GASAT: A Genetic Local Search Algorithm for the Satisfiability Problem

Frédéric Lardeux, Frédéric Saubion, Jin-Kao Hao

GASAT: A Genetic Local Search Algorithm for the Satisfiability Problem

Frédéric Lardeux lardeux@info.univ-angers.fr
Frédéric Saubion saubion@info.univ-angers.fr
Jin-Kao Hao hao@info.univ-angers.fr
LERIA, University of Angers, 2 Bd Lavoisier, F-49045 Angers Cedex 01, FRANCE

Abstract

This paper presents GASAT, a hybrid algorithm for the satisfiability problem (SAT). The main feature of GASAT is that it includes a recombination stage based on a specific crossover and a tabu search stage. We have conducted experiments to evaluate the different components of GASAT and to compare its overall performance with state-of-the-art SAT algorithms. These experiments show that GASAT provides very competitive results.

Keywords

SAT, evolutionary algorithms, tabu search, recombination operators.

1 Introduction

The satisfiability problem (SAT) (Garey and Johnson, 1979), as one of the six basic core NP-complete problems, has been the deserving object of many studies in the last two decades. In addition to its theoretical importance, SAT has a large number of practical applications such as VLSI test and verification (Biere et al., 1999), the design of asynchronous circuits (Gu and Puri, 1995), sports planning (Zhang, 2002) and so on.

An instance of the SAT problem is defined by a set of Boolean variables $\mathcal{X} = \{x_1, ..., x_n\}$ and a Boolean formula $\mathcal{F}: \{0,1\}^n \rightarrow \{0,1\}$. The formula is said to be satisfiable if there exists an assignment $v: \mathcal{X} \rightarrow \{0,1\}$ satisfying $\mathcal{F}$ and unsatisfiable otherwise. The formula $\mathcal{F}$ is in conjunctive normal form (CNF) if it is a conjunction of clauses (a clause is a disjunction of literals and a literal is a variable or its negation). Since any Boolean formula can be rewritten in CNF, CNF formulas are only considered in this paper.

SAT is originally stated as a decision problem but there are other related SAT problems that may be of interest:

- model-finding: find satisfying truth assignments,
- MAX-SAT: find an assignment which satisfies the maximum number of clauses,
- model-counting: find the number of all the satisfying truth assignments.

During the last two decades, several improved solution algorithms have been developed and important progress has been achieved. These algorithms have considerably enlarged our capacity of solving large SAT instances. Recent international challenges (Kautz and Selman, 2001; Simon et al., 2002) continue to boost the worldwide

research on SAT. These algorithms can be divided into two main classes: complete and incomplete algorithms.

A complete algorithm explores, often in an implicit way, the whole search space. Consequently, such an algorithm can be used to solve either the initial decision problem or the model-finding problem. The most powerful complete algorithms are based on the Davis-Putnam-Loveland procedure (Davis et al., 1962). They differ essentially by the underlying heuristics used for the branching rule (Dubois et al., 1996; Li and Anbulagan, 1997; Zhang, 1997). Specific techniques such as symmetry-breaking, backbone detecting or equivalence elimination are also used to reinforce these algorithms (Benhamou and Sais, 1992; Dubois and Dequen, 2001; Li, 2000). As of today, complete algorithms have an exponential complexity and the solution time may become prohibitive for large and hard instances.

An incomplete algorithm does not carry out a systematic examination of the whole search space. Instead, it explores, often in a guided way and with a limited time, some parts of the search space. Such an algorithm is appropriate for tackling the model-finding and MAX-SAT problems. Most incomplete algorithms are based on local search (Hansen and Jaumard, 1990; Selman et al., 1994; Jaumard et al., 1996; Spears, 1996) and evolutionary algorithms (EA) (De Jong and Spears, 1989; Hao and Dorne, 1994; Fleurent and Ferland, 1996; Gottlieb et al., 2002). The very simple hill-climber GSAT (Selman et al., 1992) and its powerful variant Walksat (Selman et al., 1994) are famous examples of incomplete algorithms based on local search while FlipGA (Marchiori and Rossi, 1999; Rossi et al., 2000) is a representative example of genetic algorithms for SAT. Though incomplete algorithms are of little help for proving the unsatisfiability of instances, they represent an indispensable complementary approach and a very interesting alternative with respect to the complete algorithms.

In this paper, we are interested in the development of incomplete algorithms based on the hybrid approach which combines local search and genetic search. Indeed, this general Genetic Local Search approach, also called the memetic approach (Corne et al., 1999; Hart et al., 2004), has proven to be quite successful in recent years in solving a number of well-known difficult problems such as the traveling salesman problem (Merz and Freisleben, 1997) and the graph coloring problem (Galinier and Hao, 1999). The main motivation behind this approach is to use recombination (crossover) as a guided diversification (exploration) mechanism and local search as a powerful intensification (exploitation) mechanism. A first genetic local search algorithm for SAT was reported in (Fleurent and Ferland, 1996) leading to remarkable results.

Until now, specific crossover operators have not been studied in depth for the SAT problem (Fleurent and Ferland, 1996; Marchiori and Rossi, 1999). In this work, we follow the genetic local search schema and focus on the design and study of a hybrid algorithm based on SAT specific crossover operators combined with Tabu Search (TS). The resulting algorithm is called GASAT (Genetic Algorithm for SAT). Within GASAT, specific crossover operators are used to identify particularly promising search areas while TS performs an intensified search of solutions around these areas. In such a way, we hope to be able to achieve a good compromise between intensification and diversification in the search procedure. One key point for such a hybrid algorithm is obviously the definition of the specific crossover operator which should take into account the semantic aspects of the SAT problem.

A first version of GASAT has been presented in (Hao et al., 2003). It uses a simple TS and the Corrective Clause crossover. This paper proposes a reinforced TS which uses new mechanisms, a study of four crossovers and a large panel of experimental

and comparative results.

The remainder of the paper is organised as follows. Section 2 presents our general framework and the main lines of GASAT. Section 3 provides a study of the population management. Sections 4, 5 and 6 describe our analysis of the recombination operator and of the TS. In section 7, GASAT is compared with other algorithms in order to evaluate its performance. The paper closes with our conclusions in section 8.

2 A Genetic Local Search Algorithm for SAT: GASAT

As mentioned in the introduction, GASAT is based on a genetic local search approach. It relies on the management of a population of individuals which are submitted to recombination and local search operators. The earlier version of GASAT mentioned above (Hao et al., 2003) was developed with a simple local search process. In this section, the general scheme and the main components of the improved algorithm are defined.

2.1 Representation and Search Space

The most obvious way to represent an individual for a SAT instance (as defined in the introduction) is a binary string of n bits where each bit is associated with one variable. In this representation, an individual X obviously corresponds to a truth assignment. Therefore, for a given SAT instance involving n variables, the search space is the set $\mathcal{S} = \{0,1\}^n$ (i.e. all the possible strings of n bits).

2.2 Fitness Evaluation and Associated Functions

Let $\mathcal{F}$ be a given SAT instance and X an individual, the fitness of X with respect to $\mathcal{F}$ is defined as the number of clauses of $\mathcal{F}$ which are not satisfied by X:

$$\begin{array}{rccl} eval\colon & \mathcal{S} & \rightarrow & I\!N \\ & X & \mapsto & card(\{c | \neg sat(X,c) \wedge c \in \mathcal{F}\}) \end{array}$$

where $card(A)$ denotes, as usual, the cardinality of the set A and the Boolean function $sat(X,c)$ indicates whether the clause c is true or false for X (i.e satisfied or not by the assignment corresponding to X). This fitness function induces an order $>_{eval}$ on the individuals of the population. The smallest value of this function is 0 and an individual having this fitness value corresponds to a satisfying assignment. This order will be used in the selection process.

Let $flip$ be the following function allowing us to change the value of a variable:

$$\begin{array}{rccl} flip\colon & \{0,1\} & \rightarrow & \{0,1\} \\ & \alpha & \mapsto & 1-\alpha \end{array}$$

Let $X[i \leftarrow \alpha]$ be an individual X whose i^{th} position (variable) is set to the value α. Now, the $improvement$ function is defined as follows.

$$\begin{array}{rccl} improvement\colon & \mathcal{S} \times I\!N & \rightarrow & I\!N \\ & (X,i) & \mapsto & eval(X[i \leftarrow flip(X|i)]) - eval(X) \end{array}$$

This function computes the *improvement* obtained by the flip of the i^{th} variable of X and was previously introduced in GSAT and Walksat (Selman et al., 1994; Selman et al., 1992). It corresponds to the gain of a flip according to the function $eval$ and is equal to the number of false clauses which become true by flipping the i^{th} variable minus

the number of satisfied clauses which become false. Therefore a positive number indicates an increase of the number of satisfied clauses while a negative one corresponds to an increase of the number of false clauses. This function is used in GASAT specific crossover operators and in the TS procedure.

2.3 The GASAT Algorithm

GASAT is a hybrid algorithm that combines a specific crossover operator and a TS procedure. Given a randomly generated initial population where each individual represents a truth assignment, the first step consists in selecting its best individuals according to the order $<_{eval}$. Then, two individuals (parents) are randomly selected and recombined to obtain a new individual (child). This resulting child is improved using the TS procedure and then added to the current population under certain insertion conditions. This whole process is repeated until a solution is found or until a fixed maximum number of crossovers is reached. The pseudo-code of the GASAT algorithm is described in Algorithm 1 and will be detailed in the next sections.

Data: a set of CNF clauses ϕ, $Maxflip$, $MaxNbCrossovers$
Result: the best truth assignment
begin
  CreatePopulation(P)
  $NbCrossovers \leftarrow 0$
  while *no* $X \in P$ *satisfies* ϕ *and* $NbCrossovers < MaxNbCrossovers$ **do**
    /* Selection */
    $P' \leftarrow Select(P, NbInd)$
    Choose $X, Y \in P'$
    /* Crossover */
    $Z \leftarrow Crossover(X, Y)$
    /* TS improvement */
    $Z \leftarrow TS(Z)$
    /* Insertion condition of the child */
    $P \leftarrow Replace(Z, P)$
    $NbCrossovers \leftarrow NbCrossovers + 1$
  if *there exists* $X \in P$ *satisfying* ϕ **then**
    return the corresponding assignment
  else
    return the best assignment found
end

Algorithm 1: GASAT Algorithm

3 Population Management

GASAT introduces two mechanisms to manage its population of individuals. First, a specific selection of the parents helps the crossover to produce a good child and ensures the diversity of the selected parents. Second, the children are only introduced in the population under certain insertion conditions. This mechanism acts as an intensification process by drawing aside bad individuals. Since the size of the population appears as a determinant factor for evolutionary algorithms performances, a study of this parameter has been carried out to determine an optimal size for GASAT.

3.1 Selection Operator and Insertion Condition

GASAT is a steady-state algorithm. The whole population is kept for the next generation except the oldest individual which is replaced by a child obtained by the crossover (if this one can be accepted in the population). Contrary to the well-known tournament selection which chooses the best individual in a subset of randomly selected individuals, the GASAT selection operator randomly chooses two individuals in a subset of the best individuals of the population.

The GASAT selection operator is a function $select: \mathcal{S} \times I\!N \rightarrow \mathcal{S}$ that takes as input a given population P and extracts a sub-population P' of size $NbInd$, which will serve as parents for the recombination stage. Two parents are randomly selected in this sub-population to be crossed. To insure an efficient search, it is necessary to keep some diversity in the population. Indeed, if the selected parents are too similar, some regions of the search space $\mathcal{S}$ may not be explored.

Data: a population P, $NbInd$
Result: a sub-population P'
begin
  $P' = \emptyset$
  $noP' = \emptyset$
  $WorstInd = -1$
  $WorstEval = 0$
  while $card(P') \neq NbInd$ **do**
    $Ind \leftarrow$ one individual of P
    $P = P - \{Ind\}$
    if *(*$card(P') < NbInd$*) and (*$Ind \notin P'$*)* **then**
      $P' = P' \cup \{Ind\}$
      if $eval(Ind) > WorstEval$ **then**
        $WorstInd = Ind$
        $WorstEval = eval(Ind)$
    else
      if $(eval(Ind) < WorstEval)$ *and (*$Ind \notin P'$*)* **then**
        $P' = P' \cup \{Ind\} - \{WorstInd\}$
        $noP' = noP' \cup \{WorstInd\}$
        $WorstEval = 0$
        for *all the individual* $tempInd \in P'$ **do**
          if $eval(tempInd) > WorstEval$ **then**
            $WorstInd = tempInd$
            $WorstEval = eval(tempInd)$
      if $card(P') \neq NbInd$ *and* $card(P) = 0$ **then**
        Complete P' with individuals of noP' randomly selected
end

Algorithm 2: Selection process

A child can be inserted according to whether its fitness value is better than the fitness value of the worst individual in the current sub-population P'. This condition accepts the insertion of an individual already in the sub-population. The diversity is insured by the suppression of the older individual. Algorithm 2 describes the selec-

tion process. Note that a particular set noP' is introduced to record individuals which are deleted from P' according to the insertion condition but which can be used later to complete P' when needed. An example of the selection and insertion processes is proposed Figure 3.1.

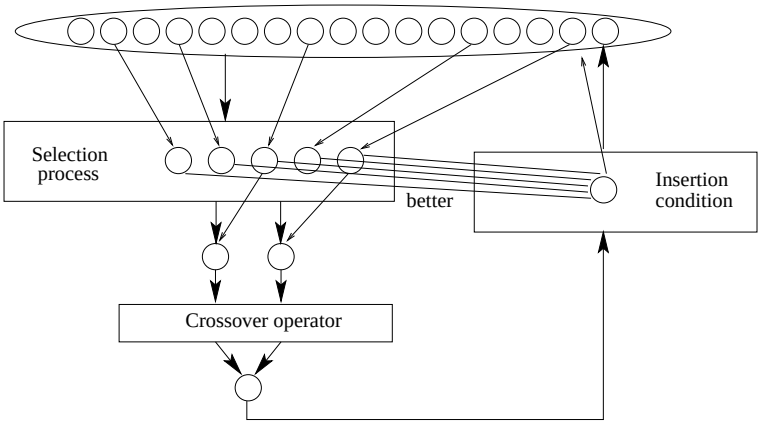

Figure 1: Selection and insertion processes. A child is obtained by a crossover between two randomly selected parents in a sub-population of the best individuals. If its fitness value is better than the value of the worst individual of the sub-population then it replaces the oldest individual of the population; otherwise the population is kept and a new selection process is executed.

3.2 Population Size

The population size is generally an important parameter for evolutionary algorithms. If the population is too small, the ability to explore the entire search space will be weak. However, if the size is too large, a lot of time will be needed to manage the population. Figure 2 shows the behavior of GASAT without TS with different population and sub-population sizes on the random 3-SAT instance `f1000` (Mitchell et al., 1992). The sub-population size is presented as a percentage of the population size. Because we are interested in both model-finding and MAX-SAT problems, the configuration quality is given by the number of false clauses in the best configuration found during an execution instead of the success rate. Each combination *population size/sub-population size* is tested 20 times. A pool of 20 populations is randomly generated and used for each run to ensure that the algorithm behavior is not due to the initial population[1]. The stopping criteria is the number of crossovers which is limited to 10^3.

As can be seen in Figure 2, when the population size increases, the configuration quality is improved and the execution time rises. The same effects are observed when the sub-population size is increased. If the purpose is to obtain very good results without taking into account the execution time, a solution consists in taking a large population and a large sub-population. On the other hand, if execution time must be very short and the configuration is not necessarily very good, it may be interesting to set a smaller population size. For GASAT, a trade-off between configuration quality and execution time has been chosen. Figure 2 suggests that a population with 100 individuals

[1] All the experiments presented in this paper which require an initial population use a pool of similar initial populations.

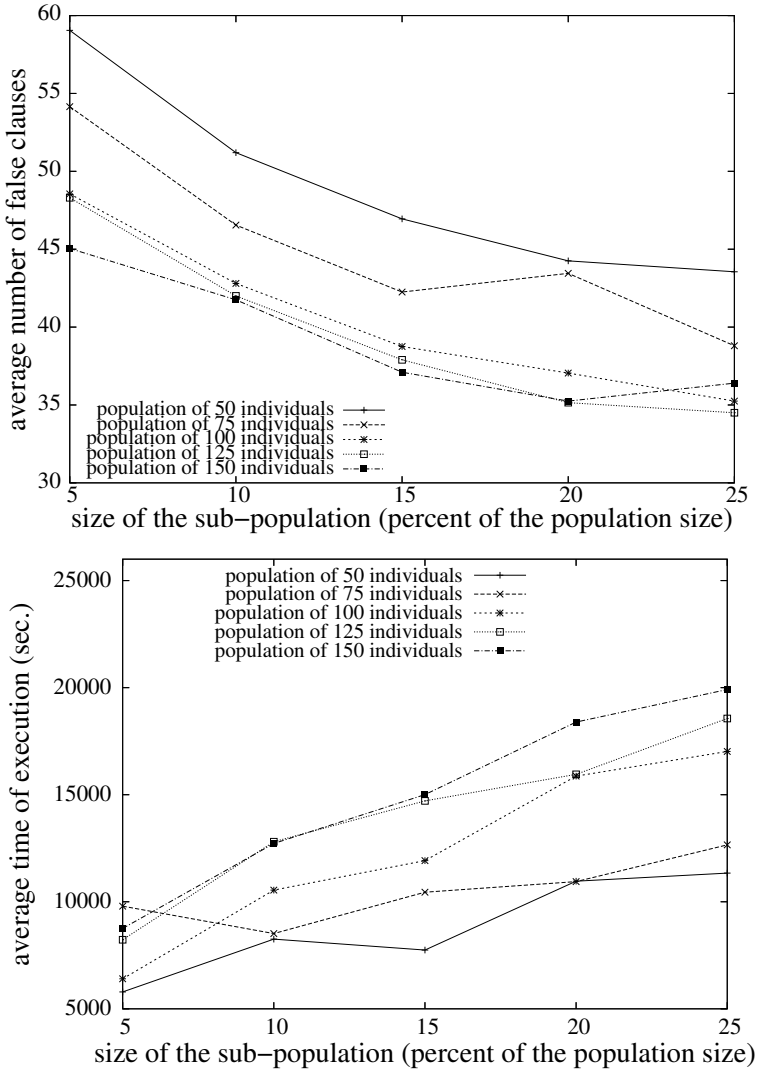

Figure 2: Influence of the population and sub-population sizes on the configuration quality (top) and the execution time (bottom).

and a sub-population composed of 15 individuals correspond to this trade-off. Similar tests have been realized on other instances and leading to similar conclusions.

4 Crossover Operators

The main goal of the crossover operator in GASAT is to create diversified and potentially promising new individuals. For this purpose, the crossover should take into account as much as possible the semantics of the individuals. In the SAT problem, the clauses of a given instance induce a constraint structure among the variables. One way to define SAT specific crossover operators is then to use this constraint structure. More specifically, based on the satisfiability of each clause with respect to the two parent in-

dividuals, one may try to create a child which benefits from both parents and satisfies a maximum number of clauses. This can be achieved by correcting false clauses and maintaining true ones. Three cases are then possible: 1) the clause is false for the two parents; 2) the clause is true for the two parents; and 3) the clause is true for one parent and false for the other.

In this section these different cases are studied and three structured crossover operators are devised. The classical uniform crossover is also presented in order to compare it with the structured ones and to assess their benefits. All these crossovers produce one child (Z) from two parents (X and Y), therefore each operator is a function $cross: \mathcal{S} \times \mathcal{S} \rightarrow \mathcal{S}$. At the beginning of all the crossovers, each variable of Z is undefined and is assigned a value along the crossover process.

4.1 Corrective Clause Crossover (CC)

When a clause c is false for both parents, a possible solution for turning c into true is to flip one of its variables[2]. However, this action may produce other false clauses. To limit the number of new false clauses, the choice of the flipping variable should be guided by the $improvement$ evaluation function. This leads to the following Corrective Clause Crossover (CC).

Data: two parents X and Y
Result: one child Z
begin
  All the variables of Z are assigned to undefined
  for *each clause* c *such that* $\neg sat(X,c) \wedge \neg sat(Y,c) \wedge \neg sat(Z,c)$ **do**
    for *all positions* i *such that the variable* x_i *appears in* c **do**
      Compute $\sigma = improvement(X,i) + improvement(Y,i)$
    Set $Z|k = flip(X|k)$ where k is the position such that σ is maximum
  All the variables of Z with no value take the value of X or Y with equal probability
end

Algorithm 3: Corrective Clause Crossover (CC)

4.2 Corrective Clause and Truth Maintenance Crossover (CCTM)

When a clause c is true for both parents, all the values of the variables appearing in c may be copied from one of the parent to the child. Unfortunately, this action would only take into account the structure of the chosen parent. To be fair, values of variables coming from both parents should be copied, but these values can be different. A solution is to select the variable whose flip has the smallest impact and to set its value such that the corresponding literal is true in c. Since only one variable is necessary to maintain this clause as true, this operation may be again guided by the $improvement$ function. This leads to the following Corrective Clause and Truth Maintenance crossover (CCTM).

4.3 Fleurent and Ferland's Crossover (F&F)

Fleurent and Ferland (1996) developed a solution for cases when a clause c is true for one parent and false for the other. Their solution is: "The corresponding variables [to

[2]Note that, if a clause is false for both parents, then all the variables appearing in this clause have necessarily the same value in both parents.

Data: two parents X and Y
Result: one child Z
begin
  All the variables of Z are assigned to undefined
  for *each clause c such that* $\neg sat(X,c) \wedge \neg sat(Y,c) \wedge \neg sat(Z,c)$ **do**
    for *all positions i such that the variable* x_i *appears in c* **do**
      Compute $\sigma = improvement(X,i) + improvement(Y,i)$
    Set $Z|k = flip(X|k)$ where k is the position such that σ is maximum
  for *each clause c such that* $sat(X,c) \wedge sat(Y,c) \wedge \neg sat(Z,c)$ **do**
    for *all positions i such that the variable* x_i *appears in c and its associated literal is true at least in one parent* **do**
      Compute $\sigma = improvement(X,i) + improvement(Y,i)$
    Value $Z|k$ such that $sat(Z,c)$ where k is the position such that σ is minimum
  All the variables of Z with no value take the value of X or Y with equal probability
end

Algorithm 4: Corrective Clause and Truth Maintenance crossover (CCTM)

this clause] are assigned values according to the parent satisfying the identified clause". This principle leads to the following definition:

Data: two parents X and Y
Result: one child Z
begin
  for *each clause c such that* $sat(X,c) \wedge \neg sat(Y,c)$ *(resp.* $\neg sat(X,c) \wedge sat(Y,c)$*)* **do**
    for *all positions i such that the variable* x_i *appears in c* **do**
      $Z|i = X|i$ (resp. $Z|i = Y|i$)
  All the variables of Z with no value take the value of X or Y with equal probability
end

Algorithm 5: Fleurent and Ferland's crossover (F&F)

It is clear that for all these crossover operators, the order in which the clauses are traversed is relevant. In our algorithm, they are traversed in the same order that they are presented in the studied instance but, of course, a specific ordering could improved the performances. Note also that CC, CCTM and F&F crossovers differ in the use of the truth values of the clauses induced by the parents. As mentioned above, the key ideas are to correct false clauses, to preserve true clauses and to maintain the structure of the parent assignments.

4.4 Uniform Crossover

In this section, the definition of the uniform crossover operator (Syswerda, 1989) is recalled. Each variable of the child is assigned by randomly taking the value of the variable of one the parents.

Data: two parents X and Y
Result: one child Z
begin
  for *each variable x* **do**
    $Z|x$ takes the value of $X|x$ or $Y|x$ with equal probability
end

Algorithm 6: Uniform Crossover

4.5 Comparison among the Crossover Operators

In order to study the characteristics of the above defined crossover operators, each of them is inserted into a simplified genetic algorithm that carries out a sequence of recombination stages on a population with or without using the selection and the insertion processes. The four crossovers presented in the previous section are used.

The experiments are presented on a random 3-SAT instance: `f500` (Mitchell et al., 1992) with 500 variables and a ratio of clauses-to-variables of 4.25 (which corresponds to hard instances (Monasson et al., 1999)) but other instances, structured (real combinatorial problems translated in SAT format) or not, have also been tested and provide similar results.

The size of the population P is 100 and the sub-population P' of possible parents has a size of 15 (according to section 3.2). A set of 20 populations has been generated and each crossover has been tested over each element of this set. The stopping criteria is the number of crossovers which is limited to 10×3. When there is no selection and insertion processes, two parents are randomly chosen in the population and all the children are inserted in the new population. Several parameters are studied: the *fitness* (i.e., the average number of false clauses) vs. *number of crossovers* and the *population diversity* (i.e. the entropy) vs. *number of crossovers*. The first comparison highlights the search power of the crossovers and the second comparison indicates the ability to keep a diversified population. The entropy, taking into account the value of each variable in each individual, allows us to measure the population diversity. It corresponds to the following function (Fleurent and Ferland, 1996) :

$$entropy(P) = \frac{-\sum_{i=1}^{n}\sum_{j=0}^{1}\frac{n_{ij}}{card(P)}log\frac{n_{ij}}{card(P)}}{nlog2}$$

where n is the number of variables and n_{ij} is the number of times the variable i is set to j in the population P. In this definition, $entropy(P) \in [0, 1]$. 0 indicates a population of identical individuals whereas 1 means that all possible assignments are almost uniformly distributed in the population.

The results are shown in Figure 3. Without the selection process, F&F and CCTM crossovers obtain the best improvement of the average number of false clauses but they also obtain the worst behavior w.r.t. the entropy. The uniform crossover does not improve the average number of false clauses and its entropy decreases whereas the CC crossover improves the average number of false clauses and maintains a high entropy.

With the selection process, CC and CCTM crossovers improve the average number of false clauses. For all the crossovers, the selection process drops the entropy rates but CC and CCTM stop these decreases sooner whereas the F&F behavior is damaged. Concerning the uniform crossover, no significant improvement of the average number

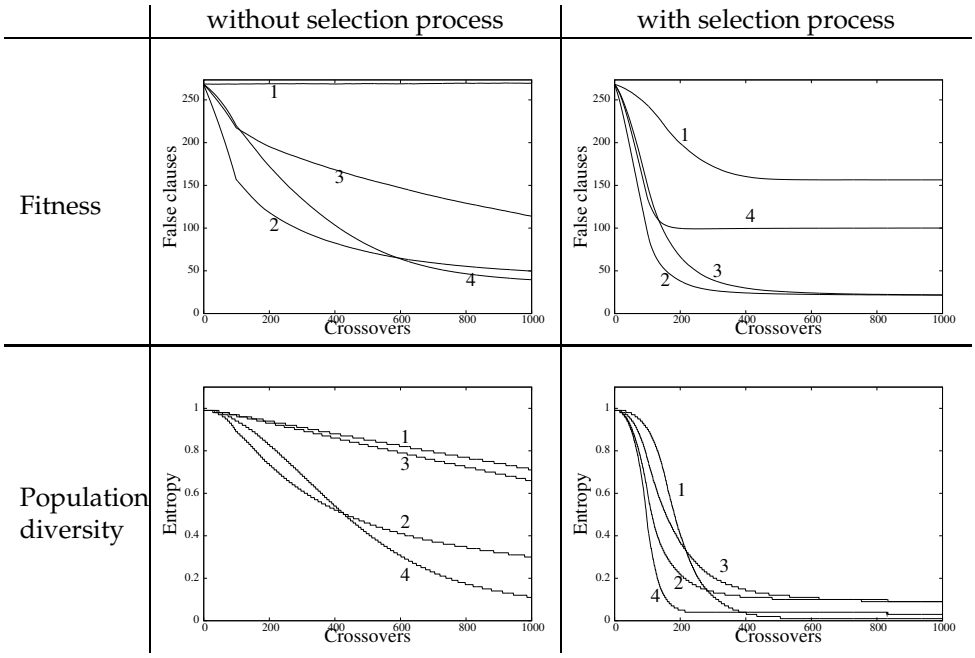

Figure 3: Evolution of the average number of false clauses and the entropy with the four crossovers (1→Uniform crossover, 2→CCTM crossover, 3→CC crossover and 4→F&F crossover).

of false clauses can be observed even with this selection mechanism. We may remark that CC and CCTM crossovers have a better behavior than the F&F crossover.

Therefore, an efficient crossover is not necessarily a crossover which quickly improves the whole population but rather which ensures a good trade-off between the quality and the diversity of the population. The diversification process allows the algorithm to benefit from a better exploration of the search space and prevents the population from stagnating in poor local optima.

5 Tabu Search (TS)

Each new individual created by the recombination stage of GASAT is improved by the TS procedure. This can be considered as an intensification stage by searching around the given individual according to a neighborhood relation.

5.1 Standard TS

TS is a local search method using a memory to avoid local optima (Glover and Laguna, 1997). TS has already been experimented for the SAT problem (Mazure et al., 1997). The principle is quite simple: it acts somewhat as a descent algorithm (at each iteration, it makes the best move), but once visited, a configuration is made tabu, that is, the algorithm is not allowed to revisit this configuration for a given number of iterations. Since the memorization of the last configuration could be costly, a common variation consists in making only the moves tabu. A move is performed if it is the best one and if it is not tabu. Once executed, the move is included in the tabu list, which acts as a fixed size FIFO queue, and is removed from this list after λ iterations. The memorization of moves instead of configurations may lead to an "over-constraint" on forbidden config-

urations. For this reason, the tabu status of a move may be disabled if the move leads to a better configuration than the best one ever found (aspiration).

TS uses an aggressive search strategy to exploit its neighborhood. Therefore, it is crucial to have special data structures and techniques which allow a fast updating of move evaluations, and reduce the effort of finding best moves.

In the SAT context, it is clear that the moves are possible flips of the values of a given assignment. The best one is selected thanks to the $choose$ function (Algorithm 7) which compares the gain provided to the current assignment by each variable flip thanks to the $improvement$ function. In order to increase efficiency, the special data structure for SAT is a matrix which provides for each variables the number of clauses becoming true and the number of clauses becoming false by its flip. It then becomes very easy to estimate the improvement due to a flip. This matrix is updated after each flip changing only the values for the variables being in the clauses of the flipped variable.

Here the TS corresponds to the TS of Mazure & al. (Mazure et al., 1997). The tabu list is a list of indexes of already performed flips. The tabu list (λ) length and the maximum number of flips are provided by an empirical study. The initial configuration given as entry to the TS is a selected child generated by crossover and TS is used to improve this child. The whole procedure is summarized in Algorithm 8.

Data: an assignment Z, the tabu list $tabu$, the best assignment found $Best$
Result: a position
begin
  for *all positions* i **do**
    if $(Z[i \leftarrow flip(Z|i)] \notin tabu) \vee (eval(Z[i \leftarrow flip(Z|i)]) < eval(Best))$ **then**
      Compute $\sigma = improvement(Z, i)$
    else
      $\sigma = -\infty$
  Return a position that is randomly selected in those which have the maximum σ
end

Algorithm 7: Choose function for the tabu search

5.2 Reinforcing TS with Refinement of the Variable Choice to Flip (RVCF)

When TS selects a variable to flip, several variables may be candidates. In order to reduce the number of possible candidate variables, a new criterion is added to this selection.

The more a clause has a significant number of true literals, the easier it is to flip one of its variables without turning this clause into false. Therefore, the notion of truth degree for a clause is introduced:

$$degree(X, c) = card(\{l | val(X, l) = 1, l \in c \in \mathcal{F}\})$$

where $\mathcal{F}$ is the set of clauses of the formula and $val(X, a)$ is the truth value of the literal a for the assignment X.

Data: an assignment Z,λ,$Maxflip$
Result: the best assignment found
begin
  Set the list $tabu$ to size λ
  $Best \leftarrow Z$
  $nbflip \leftarrow 0$
  while *not* $(eval(Best) = 0 \vee nbflip > Maxflip)$ **do**
    $i \leftarrow choose(Z)$
    if $eval(Z[i \leftarrow flip(Z|i)]) < eval(Best)$ **then**
      $Best \leftarrow Z[i \leftarrow flip(Z|i)]$
    $nbflip \leftarrow nbflip + 1$
    $Z \leftarrow Z[i \leftarrow flip(Z|i)]$
    remove the older index from $tabu$
    add i to $tabu$
  Return $Best$
end

Algorithm 8: Tabu Search (TS)

For each selected variable, its weight may be computed with the following weight function:

$$weight(X,i) = \frac{\sum_{c \in \{y | y \in \mathcal{F}, lit(i) \in y, val(X,lit(i))=1\}} degree(X,c)}{card(\{y | y \in \mathcal{F}, lit(i) \in y, val(X,lit(i)) = 1\})} + \frac{\sum_{c \in \{y | y \in \mathcal{F}, lit(i) \in y, val(X,lit(i))=0\}} degree(X,c)}{card(\{y | y \in \mathcal{F}, lit(i) \in y, val(X,lit(i)) = 0\})}$$

where $lit(i)$ represents the literal associated to the variable i. Now, the new $choose$ function (Algorithm 9) always takes as input an assignment and returns the position with the best improvement and the best weight.

Data: an assignment X
Result: a position
begin
  for *all positions* i **do**
    Compute $\sigma = improvement(X,i)$
  for *all positions* j *such that* σ *is maximum* **do**
    Compute $\alpha = weight(X,j)$
  Return a position that is randomly selected in those which have the α maximum
end

Algorithm 9: Choose function for the tabu search with the RVCF

Table 1 shows the power of the RVCF mechanism when it is added to the standard tabu search. Tests are realized on instances detailed in section 7.1.2. The number

of performed flips (number of single bit flips needed to find the best configuration) is limited to 101×10^5 (the same number is used for the experiments performed in section 7) and the tabu list length is set to 10% of the number of variables (empirical result based on (Mazure et al., 1997)). The algorithm runs 20 times on each instance.

Comparison criteria As mentioned above, since we are interested in both model-finding and MAX-SAT problems (due to the intrinsic nature of incomplete algorithms), the configuration quality is most likely related to the number of false clauses (f.c.) in the best configuration found during an execution than to the success rate. The average number of performed flips for finding the best configuration (fl.) and the average time of execution (sec.) are also used to characterize the power of the algorithms. To compare the efficiency of the algorithms, a 95% confidence Student t-test[3] has been performed in order to assess if the difference in the number of false clauses left by the two algorithms was statistically significant (Stat.). Finally, the percentage of improvement (%) in the number of false clauses due to the new mechanism is computed.

Benchmarks				TS				TS+RCVF				Stat.	Imp.
instance	var.	cls.	struc.	false clauses		fl.	sec.	false clauses		fl.	sec.	T-St.	%
				avg.	s.d.	$\times 10^3$		avg.	s.d.	$\times 10^3$		(95%)	
3blocks	283	9690	Y	1.70	0.46	1399	49	1.00	0.00	1	175	Y	+**41**
color-10-3	300	6475	Y	1.36	0.77	2881	56	0.05	0.22	3224	94	Y	+**96**
par16-4-c	324	1292	Y	4.85	0.65	4666	43	4.95	0.22	76	101	N	−**2**
par8-1	350	1149	Y	1.00	0.00	22	43	1.00	0.00	29	71	N	**0**
difp_19_0_arr_rcr	1201	6563	Y	9.25	1.34	4572	157	4.95	0.22	670	482	Y	+**46**
difp_19_99_arr_rcr	1201	6563	Y	18.35	1.62	5133	156	10.40	1.71	2302	511	Y	+**43**
color-18-4	1296	95905	Y	30.90	0.83	2653	436	26.20	1.08	4062	2556	Y	+**15**
g125.17	2125	66272	Y	17.90	1.97	793	461	16.90	2.51	1475	1414	N	+**6**
g125.18	2250	70163	Y	14.70	2.90	1117	508	13.85	2.50	1408	1358	N	+**6**
par32-5	3176	10325	Y	6.35	0.79	3533	337	5.30	0.56	1851	780	Y	+**17**
glassy-s1069116088	399	1862	N	5.00	0.00	50	59	5.00	0.00	67	185	N	**0**
glassy-s325799114	450	2100	N	8.00	0.00	2862	67	8.50	0.50	2027	207	Y	−**6**
hgen2-s1205525430	500	1750	N	0.90	0.29	361	63	0.90	0.29	937	215	N	**0**
hgen2-s512100147	500	1750	N	1.00	0.00	465	69	1.00	0.00	708	222	N	**0**
f1000	1000	4250	N	1.70	1.43	2182	108	1.00	1.7	4197	300	N	+**41**
f2000	2000	8500	N	5.50	1.88	3147	242	5.55	2.44	3464	887	N	−**1**

Table 1: Influence of the RVCF mechanism on the Tabu Search

The results presented in Table 1 show the influence of the RVCF mechanism. On structured instances, good results are observed for the RVCF mechanism but on random instances, no clear dominance appears. Random instances are more homogeneous in their constraint structures and the improvement due to the RVCF mechanism is too poor to compensate its cost: even if the RVCF improves the TS, it dramatically increases the execution time. This weakness is compensated by the good results for structured instances but this is not the case for random instances. The RVCF mechanism improves

[3]The 95% Student's t-test is based on the average, the standard deviation and the cardinality of a set of runs and used a p-value equal to 1.96. It is computed to insure that the difference of two sets is significant.

the number of false clauses but it has no impact on the standard deviation. Finally, the number of performed flips has the same order of magnitude for the two algorithms.

5.3 Reinforcing TS with Diversification

During the TS, it is not unusual to observe the number of false clauses decreasing dramatically to reach a small value which never drops to 0. The last false clauses are often the same and they block the TS which stumbles over them. These clauses are called *stumble clauses*.

To avoid this problem, a new diversification mechanism has been developed (Algorithm 10). When a stumble clause appears $MaxFalse$ times (whose typical value is 5), it is forced to become true by flipping one of its variables. This flip induces new false clauses which are forced to become true too and so on, recursively Rec times. The flipped variables cannot be flipped again before k flips which maintains these clauses true.

Data: an assignment Z, $MaxFalse$, k, Rec
Result: an assignment
begin
  $FC \leftarrow \{c | c \in clause,\ c\ appears\ false\ MaxFalse\ times\}$
  for *each* $c \in FC$ **do**
    Select the variable $i \in c$ such that $improvement(X, i)$ is maximal
    $Z \leftarrow Z[i \leftarrow flip(Z|i)]$
    i must not be flipped before k flips
    $count \leftarrow Rec$
    while $count \neq 0$ **do**
      $NFC \leftarrow \{c | c \in clause, sat(Z[i \leftarrow flip(Z|i)], c), \neg sat(Z, c)\}$
      for *each* $c \in NFC$ **do**
        Select the variable $i \in c$ such that $improvement(Z, i)$ is maximal
        $Z \leftarrow Z[i \leftarrow flip(Z|i)]$
        i must not be flipped before k flips
  Return Z
end

Algorithm 10: Diversification

This mechanism allows the search to escape from the local optima more easily than with a standard TS as highlighted by the peaks in Figure 4. Tests have been performed on several instances and the same effect has been observed. Figure 4 which is representative of the diversification mechanism effect presents the evolution of one run on a random instance (`f1000`).

This mechanism is introduced in the TS algorithm at the end of the $While$ loop, just after the introduction of the flipped variable in the tabu list.

5.4 Combination of the Previous Mechanisms

The two specific mechanisms slow down the algorithm but allow us to improve the results in terms of quality. The tests are presented on two instances using all possible combinations of the previously described mechanisms. Each instance is a representative of one family of problems (`f1000` for random instances and `color10-3` for

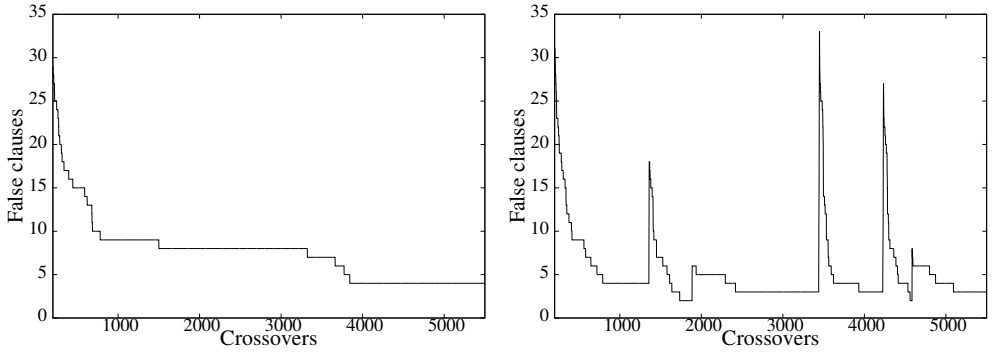

Figure 4: TS without diversity (left) and TS with diversity (right)

structured instances (see section 7.1.2)).

The parameters are a TS of 101×10^5 performed flips and a tabu list length of 10% of the number of variables. Each instance is run 20 times. A new criteria of comparison is added to characterize most precisely the power of the mechanisms. Four values are given: the success rate (%) which is the number of successful runs divided by the total number of runs, the average number of false clauses (f.c.), the number of performed flips (fl.) and the average time in second (sec.) for finding the best configuration.

	Instances							
	f1000				color10-3			
	%	f.c.	fl.$\times 10^3$	sec.	%	f.c.	fl.$\times 10^3$	sec.
Standard TS	25	1.70	2181	94.75	15	1.36	2881	56.93
TS + RVCF	60	1.00	4197	300.82	95	0.05	3224	94.54
TS + Diversification	45	0.69	2929	94.75	10	1.09	4246	53.12
TS + RVCF + Diversification	45	0.81	3081	342.85	100	0.00	3631	119.58

Table 2: Comparisons of RVCF and diversification for the Tabu Search

Table 2 shows the interaction between the two mechanisms for random and structured instances. Only two instances are presented but others were tested and similar results were obtained.

For random instances, the diversification mechanism and the RVCF mechanism improve the results but the combination of both deteriorates them (see Table 1). TS with the diversification mechanism seems to provide the best results for random instances.

For structured instances, the combination of the RVCF mechanism and the diversification mechanism improves the results. The RVCF mechanism intensifies the search whereas the diversification mechanism allows the search to escape from local optima.

6 Combining Crossover and Tabu Search

The previous section presented four crossovers and a reinforced TS. Even if crossovers provide us with good results individually, it is not clear if they would be efficient when combined with the reinforced TS (RTS). In this section, the good results of the CC crossover combined with RTS are shown. Then, results will be presented in order to show that the RTS benefits from the evolutionary algorithm.

6.1 Influence of the Crossover

To further assess the interaction between RTS and each crossover operator, each of the four operators are inserted in GASAT with the reinforced tabu search combining RVCF and diversification mechanisms. Each of the four hybrid algorithms is run 20 times on some selected representative instances, namely, a blocks world instance (`3blocks`), a parity function instance (`par16-4-c`), a random instance (`f1000`) and a chess-board coloring instance (`color10-3`). These instances are presented in detail in section 7.1.2. They come from four different representative families of benchmarks. The experimental conditions are the same for the four algorithms. The F&F crossover is used without the selection process since it gives better results (Figure 3). A RTS of at most 10^4 performed flips is applied to each child and the number of crossovers is limited to 10^3. The flips which have been done during the crossovers are taken into account in the total number of flips. A set of 20 populations has been generated and each combination has been tested over each element of this set.

Four comparison criteria are used in order to evaluate the different crossovers improved by RTS. The success rate (%), which is the number of successful runs divided by the total number of runs, and the average number of false clauses highlight the search power of the algorithm. The speed is evaluated using average number of crossovers (cr.) in the successful runs and the average number of performed flips for finding the best configuration.

	CC		CCTM		F&F		Uniform	
instances	%	cr.	%	cr.	%	cr.	%	cr.
3blocks	10	439	0	-	0	-	0	-
color-10-3	100	287	95	224	95	255	95	218
f1000	85	235	70	177	60	189	70	146
par16-4-c	5	788	10	206	0	-	0	-

Table 3: Comparison based on the success rate of different crossovers included in the GASAT algorithm

	CC		CCTM		F&F		Uniform	
instances	f.c.	fl.$\times 10^3$	f.c.	fl.$\times 10^3$	f.c.	fl.$\times 10^3$	f.c.	fl.$\times 10^3$
3blocks	0.90	90	1.00	0.2	1.00	0.2	1.00	0.8
color-10-3	0.00	41	0.03	934	0.03	788	0.03	1001
f1000	0.16	6417	0.32	622	0.17	775	0.23	715
par16-4-c	2.60	538	3.55	129	2.90	215	2.75	368

Table 4: Comparison based on the average number of false clauses of different crossovers included in the GASAT algorithm

Table 3 and Table 4 give the same conclusion: the combination between CC and RTS is more powerful than the others.

6.2 Improvement due to Each Process

To assess the benefit of the evolutionary process for the combination of TS and a crossover operator, Table 5 presents a comparison between the RTS and the combination of this RTS and the CC crossover (RTS+CC). In the same way, Table 6 presents

a comparison between the evolution process only with the CC crossover and the hybridization RTS+CC. Most of the results are improved in the presence of the evolutionary process.

For this combination, our RTS using the RVCF (section 5.2) and the diversification mechanism (section 5.3) is applied to each child. The number of crossovers is limited to 10^3 and the number of performed flips per child is limited to 10^4. The flips which are done during the crossovers are taken into account in the total flips number. A set of 20 populations is generated and each combination is tested over each element of this set. Concerning the reinforced RTS without evolutionary process, the number of flips is limited to 101×10^5 performed flips. The tabu list length is set to 10% of the number of variables in all the TS.
For the evolutionary process alone, the number of crossovers is limited to 10^3 and the crossover is the CC crossover.

Comparison criteria The configuration quality is given by the number of false clauses in the best configuration found during an execution. The average of this number (avg.) and its standard deviation (s.d.) are presented for 20 executions. The average number of performed flips for finding the best configuration (fl.) is also used to characterize the power of the algorithms. Again, to compare the efficiency of the algorithms, a 95% confidence Student t-test allows us to check that the number of false clauses of the two algorithms is significantly different. Finally, the percentage of improvement (%) in the average number of false clauses due to the evolutionary process (Table 5) or to the RTS (Table 6) is computed.

Benchmarks				RTS			RTS+CC			Stat.	Imp.
instance	var.	cls.	SAT	false clauses		fl.	false clauses		fl.	T-St.	%
				avg.	s.d.	$\times 10^3$	avg.	s.d.	$\times 10^3$	(95%)	
3blocks	283	9690	Y	1.00	0.00	0.1	0.90	0.30	41	N	**+10**
color-10-3	300	6475	Y	0.00	0.00	3631	0.00	0.00	926	N	**0**
par16-4-c	324	1292	Y	5.05	1.77	56	2.60	0.54	538	Y	**+49**
par8-1	350	1149	Y	1.00	0.00	10	0.50	0.50	587	Y	**+50**
difp_19_0_arr_rcr	1201	6563	Y	5.00	0.00	317	4.96	0.2	730	N	**+1**
difp_19_99_arr_rcr	1201	6563	Y	10.40	1.62	2375	4.95	0.22	8402	Y	**+52**
color-18-4	1296	95905	Y	25.70	1.14	3370	25.20	0.51	2082	N	**+2**
g125.17	2125	66272	Y	17.25	2.21	939	3.38	0.99	142	Y	**+80**
g125.18	2250	70163	Y	13.80	1.25	1006	0.2	0.4	134	Y	**+99**
par32-5	3176	10325	Y	5.00	0.71	2540	5.60	0.49	2428	Y	**−12**
glassy-s1069116088	399	1862	Y	5.00	0.00	48	5.00	0.00	157	N	**0**
glassy-s325799114	450	2100	Y	8.00	0.00	2567	8.00	0.00	601	N	**0**
hgen2-s1205525430	500	1750	Y	0.90	0.30	306	1.00	0.00	304	N	**−11**
hgen2-s512100147	500	1750	Y	1.00	0.00	716	1.20	0.40	304	Y	**−20**
f1000	1000	4250	Y	0.69	0.77	2929	0.16	0.89	6417	Y	**+77**
f2000	2000	8500	Y	3.14	2.08	2671	1.93	0.96	1912	Y	**+39**

Table 5: Influence of the evolutionary process in GASAT

From Tables 5 and 6, it appears that the combination of the evolutionary process and the RTS improves most of the time the results of RTS alone or of the evolutionary

Benchmarks				CC		RTS+CC		Stat.	Imp.
instance	var.	cls.	SAT	false clauses		false clauses		T-St.	%
				avg.	s.d.	avg.	s.d.	(95%)	
3blocks	283	9690	Y	15.70	2.83	0.90	0.30	Y	+**94**
color-10-3	300	6475	Y	7.65	1.39	0.00	0.00	Y	+**100**
par16-4-c	324	1292	Y	21.25	3.28	2.60	0.54	Y	+**88**
par8-1	350	1149	Y	20.50	3.25	0.50	0.50	Y	+**98**
difp_19_0_arr_rcr	1201	6563	Y	66.65	6.84	4.96	0.2	Y	+**93**
difp_19_99_arr_rcr	1201	6563	Y	71.15	8.17	4.95	0.22	Y	+**93**
color-18-4	1296	95905	Y	52.55	2.78	25.20	0.51	Y	+**52**
g125.17	2125	66272	Y	22.50	3.29	3.38	0.99	Y	+**85**
g125.18	2250	70163	Y	17.84	3.30	0.2	0.4	Y	+**99**
par32-5	3176	10325	Y	234.15	12.78	5.60	0.49	Y	+**98**
glassy-s1069116088	399	1862	Y	25.75	4.49	5.00	0.00	Y	+**100**
glassy-s325799114	450	2100	Y	30.10	3.53	8.00	0.00	Y	+**100**
hgen2-s1205525430	500	1750	Y	25.70	3.38	1.00	0.00	Y	+**96**
hgen2-s512100147	500	1750	Y	24.95	3.43	1.20	0.40	Y	+**95**
f1000	1000	4250	Y	37.40	5.49	0.16	0.89	Y	+**99**
f2000	2000	8500	Y	72.10	6.82	1.93	0.96	Y	+**97**

Table 6: Influence of the RTS in GASAT

Benchmarks			CC / GASAT	CCTM / GASAT
instance	var.	cls.	Performed Flips	Performed Flips
f1000	1000	4250	0.000035	0.000075
3blocks	283	9690	0.000084	0.000102
par32-5	3176	10325	0.000126	0.000215

Table 7: Cost of crossovers w.r.t. RTS

process alone.

Finally, the cost of the crossovers has been evaluated with respect to the RTS in term of flips. Table 7 shows that the CC and the CCTM crossovers do not perform a signifiant number of flips w.r.t. the number of flips performed by RTS. Note that CC and CCTM execute approximatly the same number of performed flips.

From now on, GASAT is used to denote the hybridization between RTS and CC crossover which obtains the best results.

7 Experimental Results

In this section, the GASAT algorithm is evaluated. First, GASAT is compared with five well-known evolutionary algorithms and then with two state-of-the-art SAT solvers: Walksat (Selman et al., 1994) and UnitWalk (Hirsch and Kojevnikov, 2001). Walksat is historically one of the best incomplete solvers while UnitWalk is a winner of the SAT2003 competition. Tests are realized on a cluster with Linux and Alinka (5 nodes each of them with 2 CPU Pentium IV 2.2 Ghz and 1 GB of RAM) used sequentially.

7.1 Comparisons with Evolutionary Algorithms

Results presented in (Gottlieb et al., 2002) are used to compare GASAT with some evolutionary algorithms and test FlipGA (Marchiori and Rossi, 1999) and GASAT on different benchmarks.

7.1.1 Comparisons with Results Presented in Gottlieb et al.

Gottlieb et al. proposed several evolutionary algorithms for SAT (Gottlieb et al., 2002). Results presented in that paper (Gottlieb et al., 2002) are considered and GASAT results are added w.r.t. the same comparisons criteria. To get the same search power for all the algorithms the parameters of (Gottlieb et al., 2002) are used: each instance is tested between 5 to 50 times and the number of performed flips is limited to 3×10^5. The data provided in Table 8 are: the success rate (%) which is the number of successful runs divided by the total number of runs and the average number of performed flips for successful runs.

The solvers tested in (Gottlieb et al., 2002) are:

- SAWEA (Eiben and van der Hauw, 1997): using the *Stepwise Adaptation of Weights*, it increases only weights that correspond to unsatisfied clauses and implicitly forcing the evolutionary search to focus on these difficult clauses.

- RFEA2 and RFEA2+ (Gottlieb and Voss, 2000): they use a refining function based on associating weights with each variable, high positive weights indicate that corresponding variables are favored to be true whereas negative weights express a preference to false.

- FlipGA (Marchiori and Rossi, 1999): it is an evolutionary local search algorithm which generates a child by standard genetic operators and then improves it by means of local search (*Flip Heuristic*). It shares some similarities with GASAT.

- ASAP (Rossi et al., 2000): it is obtained from FlipGA by changing the *Flip Heuristic* in a tabu search process.

All the used instances are generated with the problem generator *mkcnf* written by Allen van Gelder (Van Gelder, 1993). They are random 3-SAT instances with a ratio clauses-to-variables of 4.3:

- suite A, 4 groups of 3 instances with 30, 40, 50 and 100 variables,

- suite B, 3 groups of 50 instances with 50, 75 and 100 variables,

- suite C, 5 groups of 100 instances with 20, 40, 60, 80 and 100 variables.

Table 8 shows that GASAT does not have the best success rate for random instances with few variables. But, in the next section, we will show that, for large instances, GASAT is very competitive w.r.t. one of the best evolutionary solver (FlipGA).

Instances			GASAT		SAWEA		RFEA2		RFEA2+		FlipGA		ASAP	
Suite	n.b.	var.	%	fl.	%	fl.	%	fl.	%	fl.	%	fl.	%	fl.
A	3	30	99	1123	100	34015	100	3535	100	2481	100	25490	100	9550
A	3	40	100	1135	93	53289	100	3231	100	3081	100	17693	100	8760
A	3	50	91	1850	85	60743	100	8506	100	7822	100	127900	100	68483
A	3	100	95	7550	72	86631	99	26501	97	34780	87	116653	100	52276
B	50	50	96	2732	-	-	100	12053	100	11350	100	103800	100	61186
B	50	75	83	6703	-	-	95	41478	96	39396	82	29818	87	39659
B	50	100	69	28433	-	-	77	71907	81	80282	57	20675	59	43601
C	100	20	100	109	100	12634	100	365	100	365	100	1073	100	648
C	100	40	100	903	89	35988	100	3015	100	2951	100	14320	100	16644
C	100	60	97	9597	73	47131	99	18857	99	19957	100	127520	100	184419
C	100	80	66	7153	52	62859	92	50199	95	49312	73	29957	72	45942
C	100	100	74	1533	51	69657	72	68053	79	74459	62	20319	61	34548

Table 8: Comparison among evolutionary algorithms ($n.b.$ is the number of instances and $var.$ is the number of variables)

7.1.2 Comparison between GASAT and FlipGA

To compare GASAT with an existing evolutionary algorithm on large instances, FlipGA, which is one of the ones most similar to GASAT, is used. Due to the specific local search (hill-climbing) of FlipGA, it is more convenient to limit the number of crossovers instead of the number of flips. In fact, since the number of crossovers is not controlled in FlipGA, all the flips can be done during the local search process. Thus FlipGA and GASAT are limited to 10^2 crossovers. Each instance is tested 20 times.

Tests are performed on two classes of instances: structured instances and random instances including satisfiable and unsatisfiable problems. All these instances[4] were presented at the SAT2002 (Simon et al., 2002) or SAT2003 competitions.

- structured instances (real problems translated in SAT format):
  · `3blocks` (a blocks world problem),
  · `color-10-3`, `color-18-4`coloring problems (Beresin et al., 1989)),
  · `difp_19_0_arr_rcr`, `difp_19_99_arr_rcr` (integer factorization problems),
  · `mat25.shuffled`, `mat26.shuffled` ($n \times n$ matrix multiplication with m products (Li et al., 2002)),
  · `par16-4-c`, `par32-5`, `par32-5-c` (problems of learning the parity function).

- random instances:
  · `f1000`, `f2000` (DIMACS instances(Mitchell et al., 1992)),
  · 2 instances of 500 variables generated by `hgen2` with seeds 1205525430 (hgen2-a) and 512100147 (hgen2-b),
  · 2 instances generated by `glassy` one with 399 variables and seed 1069116088 (glassy-a) the other with 450 variables and seed 325799114 (glasst-b).

FlipGA specificity : FlipGA works only on exact-3-SAT instances. Therefore, all the instances must be transformed by the following mechanism:

[4] Available at `http://www.info.univ-angers.fr/pub/lardeux/SAT/benchmarks-EN.html`

$$
\begin{array}{ll}
(x) & \rightarrow (x \vee X_1 \vee X_2) \wedge (x \vee \neg X_1 \vee X_2) \\
 & \wedge (x \vee X_1 \vee \neg X_2) \wedge (x \vee \neg X_1 \vee \neg X_2) \\
(x_1 \vee x_2) & \rightarrow (x_1 \vee x_2 \vee X_1) \wedge (x_1 \vee x_2 \vee \neg X_1) \\
(x_1 \vee x_2 \vee x_3) & \rightarrow (x_1 \vee x_2 \vee x_3) \\
(x_1 \vee x_2 \vee x_3 \vee x_4) & \rightarrow (x_1 \vee x_2 \vee X_1) \wedge (\neg X_1 \vee x_3 \vee x_4) \\
(x_1 \vee x_2 \vee \ldots \vee x_i \vee \ldots \vee x_{n-1} \vee x_n) & \rightarrow (x_1 \vee x_2 \vee X_1) \wedge \ldots \wedge (\neg X_{i-2} \vee x_i \vee X_{i-1}) \\
 & \wedge \ldots \wedge (\neg X_{n-3} \vee x_{n-1} \vee x_n)
\end{array}
$$

This transformation increases the number of clauses and the number of variables but preserves the minimal number of false clauses for unsatisfiable instances. Now the precise parameters are provided for each algorithm.

GASAT: GASAT uses the CC crossover with a population of 10^2 individuals. During the initialization of this population, a RTS of 10^3 performed flips is applied to each individual. The selection process for the parents and the insertion condition for the children are activated. The number of possible parents for the crossover is limited to 15 different individuals. The number of allowed crossovers is 10^2 and a RTS of at most 10^4 performed flips is applied to each child. The size of the tabu list is set to 10% of the number of variables in the problem. The RVCF mechanism is applied only on structured instances. The diversification mechanism is activated when a unique false clause remains unsatisfied during 5 performed flips. False clauses are recursively set to true 10 times. The number of times that a flipped variable must wait before it can be flipped again is set to 10% of the number of variables.

FlipGA: Here a version provided by Claudio Rossi is used. The pool size is equal to 10 because larger populations affect the efficiency of FlipGA as it was indicated in (Marchiori and Rossi, 1999). The generation process stops after 10^2 generations and uses the convergence mechanism. FlipGA uses a uniform crossover and the mutation is disabled. The other parameters are standard values stored in a configuration file which is provided with the FlipGA algorithm.

Comparison criteria Four comparison criteria are used to evaluate GASAT versus FlipGA. For each instance, the average number (avg.) of false clauses is indicated for the best configuration obtained after 20 runs, the standard deviation (s.d.) and the average number of performed flips and the average running time for finding the best configuration. Success rate is not mentioned because for many instances, no solution is found. The statistical value (Stat.) of the Student t-test is also mentioned (see section 5.2). Finally, the percentage of improvement (%) in the number of false clauses due to the use of GASAT is computed.

Table 9 shows a clear dominance in term of quality (i.e., f.c. and fl.) of GASAT on structured instances and on random instances with more than 400 variables whereas the execution time is larger. The high number of flips executed by FlipGA confirm the fact that few crossovers would be applied if the number of flips was limited.

7.2 Comparison among GASAT, Walksat and UnitWalk

Due to the incomplete and non-deterministic character of GASAT, Walksat (Selman et al., 1994) and UnitWalk (Hirsch and Kojevnikov, 2001), each algorithm has been run 20 times on each benchmark. Tests are performed on a subset of the SAT2003

Benchmarks			FlipGA				GASAT				Stat.	Imp.
instances	var.	cls.	f.c. avg.	f.c. s.d.	fl. $\times 10^3$	sec.	f.c. avg.	f.c. s.d.	fl. $\times 10^3$	sec.	T-St. (95%)	%
par16-4-c.3SAT	446	1414	10.15	0.85	1026	9	5.85	1.01	137	35	Y	**+42**
mat25.shuffled.3SAT	1388	2768	10.10	2.41	674	28	7.60	0.80	113	161	Y	**+25**
mat26.shuffled.3SAT	1704	3424	12.89	3.54	9143	44	8.00	0.00	41	749	Y	**+38**
par32-5-c.3SAT	1711	5722	27.40	1.68	5746	77	19.60	1.69	311	129	Y	**+28**
3blocks.3SAT	2783	12190	7.20	0.40	11631	79	5.25	1.09	197	436	Y	**+27**
difp_19_0_arr_rcr.3SAT	5157	10560	87.60	7.23	20340	109	84.25	6.13	657	661	N	**+4**
difp_19_99_arr_rcr.3SAT	5157	10560	87.95	9.75	20172	107	81.40	7.14	639	658	Y	**+7**
par32-5.3SAT	6458	13748	50.65	3.35	62149	290	41.25	5.02	755	813	Y	**+19**
color-10-3.3SAT	6675	12850	41.65	1.80	24664	86	46.53	3.08	18	1343	Y	-12
color-18-4.3SAT	97200	191808	2064.35	363.65	2818	1150	248.50	0.50	27	33128	Y	**+88**
glassy-a.3SAT	399	1862	7.60	1.06	411	16	5.00	0.00	153	18	Y	**+34**
glassy-b.3SAT	450	2100	11.45	1.28	479	21	8.95	0.22	166	86	Y	**+22**
hgen2-a.3SAT	500	1750	6.24	1.19	579	16	1.40	0.49	294	22	Y	**+78**
hgen2-b.3SAT	500	1750	7.00	1.34	575	18	1.80	0.68	268	22	Y	**+74**
f1000.3SAT	1000	4250	8.90	1.67	1480	47	2.30	0.90	408	45	Y	**+74**
f2000.3SAT	2000	8500	16.90	2.07	3641	122	7.35	1.80	709	97	Y	**+57**

Table 9: Comparison between FlipGA and GASAT

competition instances in which each family is represented by at least one instance. The search effort of Walksat and UnitWalk is essentially defined by the number of allowed flips. Therefore, when GASAT is compared with Walksat and UnitWalk, their number of performed flips are limited to 101×10^5. As in the SAT2003 competition, the running time is limited to one hour for each run. We will now provide the precise parameters for each algorithm.

Walksat: Walksat is a randomized local search algorithm. It tries to determine the best move by randomly choosing an unsatisfied clause and selecting a variable to flip within it. The version v41[5] is used. The number of tries (Maxtries) is set to 10 with at most 101×10^4 performed flips for each try. When one solution is found, the search stops. Walksat uses the "novelty" heuristic with a noise set to 0.5 (its default value).

UnitWalk: UnitWalk is an incomplete randomized solver. It is a combination of unit clause elimination and local search. UnitWalk version 0.981[6] is used. A function which provides the best assignment found has been included, since the standard UnitWalk solver only returns the last assignment found. The maximum number of allowed flips is 101×10^5 for each try.

GASAT: GASAT uses the CC crossover which works with a population of 10^2 individuals. During the initialization of this population, a RTS of 10^3 performed flips is applied to each individual. The selection process for the parents and the insertion condition for the child are activated. The number of possible parents for the crossover is limited to 15 different individuals. The number of allowed crossovers is 10^3 and an RTS of at most 10^4 performed flips is applied to each child. So, the maximum number of performed flips allowed is 101×10^5. The size of the tabu list is set to 10%

[5]Walksat is available: `http://www.cs.washington.edu/homes/kautz/walksat/`
[6]UnitWalk is available: `http://logic.pdmi.ras.ru/~arist/UnitWalk/`

of the number of variables in the problem. The RVCF mechanism is applied only to structured instances. The diversification mechanism is activated when a unique false clause remains unsatisfied for 5 iterations. False clauses are recursively set to true 10 times. The number of times that a flipped variable must wait before it can be flipped is set again to 10% of the number of variables.

Comparison criteria All the instances of the SAT 2003 competition are very hard to satisfy (when they are satisfiable). Therefore, it seems to be more interesting and wiser to use the number of false clauses in the best configuration found during an execution than success rate. The two first criteria are the average number of clauses (avg.) and its standard deviation (s.d.) for 20 runs. The third criterion is the average number of performed flips (fl.) to obtain this best configuration. Although the CPU time is not given in the tables, it is in the same order of magnitude for both UnitWalk and Walksat whereas GASAT is more or less 20 % slower. Here, statistical values of the Student t-test are not mentioned because, for each instance, the results obtained by the best algorithm are, most of the time, significantly different with respect to the others.

Results

For each instance, a ranking based on the performance of each solver is proposed taking into consideration the average number of false clauses and, if this number is the same for several solvers, the average number of performed flips is taken into account. The first place is attributed to the best solver and the third place for the worst. Then, the average ranking is computed for each solver. This scoring scheme is not perfect because it does not take into account the distances among the three results but it does let us highlight the algorithms winning the most instances as in the SAT 2003 competition.

Bold type is used in the three tables to emphasize the best average number of false clauses except when these results are similar for the three solvers.

Handmade instances (theoretical problems coded in SAT):

The results given in Table 10 show that GASAT is competitive on the handmade instances. It obtains an average rank of 1.75 whereas UnitWalk obtains 1.88 and Walksat 2.38.

Random instances:

Table 11 also shows that GASAT provides similar and sometimes better results than Walksat and UnitWalk on the random instances. GASAT obtains an average rank of 1.54 whereas Walksat obtains 2.00 and UnitWalk 2.46. We observe that GASAT gets results with a very small standard deviation compared with the other solvers.

Industrial instances:

From Table 12, we observe that the results of GASAT are not as interesting as for the handmade and random instances. Indeed, the average rank for GASAT is 2.3, the one of Walksat is 1.6, and the one of UnitWalk is 2.1. The relatively bad behavior of GASAT is particularly visible for four large instances (k2fix_gr_2p-invar_w9.shuffled-as.sat03-436, cnt10.shuffled-as.sat03-418, dp11u10.shuffled-as.sat03-422, frg1mul.miter.shuffled-as.sat03-351). Analyzing the RCVF mechanism of GASAT, we suspect that the high number of clauses in these instances is a possible explanation for this "bad" performance. Indeed, the degree function requires the examination of a lot of clauses and is as a result a time-consuming process. However, the one hour

Benchmarks			GASAT			Walksat			UnitWalk		
instances	var.	cls.	f.c. avg.	f.c. s.d.	fl. $\times 10^3$	f.c. avg.	f.c. s.d.	fl. $\times 10^3$	f.c. avg.	f.c. s.d.	fl. $\times 10^3$
2000009987nc.-shuffled-as.sat03-1665	2756	10886	43.70	6.32	1063	89.70	8.75	4847	**10.50**	1.53	367
bqwh.33.381.shuffled-as.sat03-1642	1555	9534	50.05	7.32	985	88.10	3.95	4143	**38.80**	3.74	533
clus-1200-4800-20-20-001.shuffled-as.sat03-1085	1200	4800	1.05	0.67	899	**0.00**	0.00	63	9.15	11.23	2338
color-18-4.-shuffled-as.sat03-1486	1296	95904	**26.15**	0.73	330	38.05	1.02	3412	52.90	1.79	36
dodecahedron.-shuffled-as.sat03-1429	30	80	1.00	0.00	0.007	1.00	0.00	0.008	1.00	0.00	0.009
ezfact64_1.shuffled-as.sat03-1517	3073	19785	46.85	8.65	384	69.55	6.05	4999	**18.00**	3.11	129
genurq4Sat.shuffled-as.sat03-1510	64	298	0.00	0.00	0.058	0.00	0.00	0.181	0.00	0.00	0.173
hypercube5.shuffled-as.sat03-1435	80	512	1.00	0.00	0.010	1.00	0.00	0.012	1.00	0.00	0.015
hwb-n22-01-S1917708524.shuffled-as.sat03-1612	144	688	1.00	0.00	0.638	1.00	0.00	0.484	1.00	0.00	0.219
hwb-n32-01-S1491788039.shuffled-as.sat03-1637	226	1078	1.00	0.00	3.388	1.00	0.00	1.317	1.00	0.00	0.552
marg5x5.shuffled-as.sat03-1455	105	512	1.00	0.00	0.040	1.00	0.00	0.027	1.00	0.00	0.024
par32-5.shuffled-as.sat03-1540	3176	10325	**5.25**	0.89	2512	12.85	1.39	4288	7.40	1.24	216
pyhala-braun-unsat-35-4-03.shuffled-as.sat03-1543	7383	24320	80.15	53.88	2008	196.45	5.39	5229	**33.05**	2.40	324
SGI_30_80_18_90_3-log.shuffled-as.sat03-194	90	27978	**1.05**	0.22	51399	1.45	0.50	2073	3.70	0.46	449
SGI_30_60_28_50_8-log.shuffled-as.sat03-129	140	65504	**4.55**	0.50	492	5.05	0.22	3521	12.00	1.10	421
x1_36.shuffled-as.sat03-1589	106	282	1.00	0.00	0.056	1.00	0.00	0.067	1.00	0.00	0.035

Table 10: SAT2003 competition handmade benchmarks: comparison among GASAT, Walksat and UnitWalk. The running time is limited to 1 hour and the number of authorized performed flips is 101×10^5. GASAT obtains the best average rank.

cut-off limit used in the stop condition does not allow GASAT to reach the search limit fixed by the number of flips (101×10^5). In order to check this hypothesis, we re-ran the three algorithms on the four large instances with the same condition except the "one hour cut-off limit". Therefore, these algorithms stop only when the limit of 101×10^5 performed flips is reached. Results of this additional experiment are given in Table 13. From this table, we observe that the results of GASAT are greatly improved

Benchmarks			GASAT			Walksat			UnitWalk		
instances	var.	cls.	f.c. avg.	s.d.	fl. $\times 10^3$	f.c. avg.	s.d.	fl. $\times 10^3$	f.c. avg.	s.d.	fl. $\times 10^3$
gencnf-k4-r9.88-v200-c1976-03-S153-9378038.shuffled-as.sat03-1745	200	1976	1.00	0.00	75	1.00	0.00	43	3.85	1.56	4964
gencnf-k7-r88.7-v85-c7539-01-S962370-643.shuffled-as.sat03-1783	85	7539	**0.30**	0.46	20312	0.50	0.50	2152	7.20	2.25	3740
gencnf-k8-r180-v61-c10980-02-S893013-537.shuffled-as.sat03-1794	61	10980	1.00	0.00	16	1.00	0.00	65	5.15	0.85	1667
gencnf-k9-r357-v46-c16422-03-S1437098-472.shuffled-as.sat03-1810	46	16422	0.00	0.00	29	0.00	0.00	108	0.85	1.42	766
gencnf-k10-r720-v38-c27360-01-S1687864-750.shuffled-as.sat03-1718	38	27360	1.00	0.00	4	1.00	0.00	8	2.55	0.86	258
glassyb-v399-s732-524269.shuffled-as.sat03-1680	399	1862	**5.00**	0.00	360	5.55	0.50	2364	8.00	1.18	4174
hardnm-L19-02-S125896754.shuffled-as.sat03-916	361	1444	12.25	0.83	971	16.55	1.16	4064	**4.15**	0.65	3215
hardnm-L32-02-S964513274.shuffled-as.sat03-941	1024	4096	53.85	1.28	1078	78.60	2.91	4079	**20.20**	1.96	3481
hgen6-4-24-n390-02-S1171124847.shuffled-as.sat03-836	390	1653	**1.30**	0.95	48317	1.35	0.91	1852	4.65	2.54	5231
hgen8-n120-03-S196-2183220.shuffled-as.sat03-877	120	193	1.00	0.00	0.053	1.00	0.00	0.060	1.00	0.00	0.057
hidden-k3-s0-r5-n700-03-S1609-878926.shuffled-as.sat03-972	700	3500	**17.65**	0.48	451	27.65	0.96	3340	46.45	4.89	4993
hidden-k3-s1-r6-n500-03-S408319111.-shuffled-as.sat03-1022	500	3000	0.00	0.00	2.791	0.00	0.00	0.884	0.00	0.00	2.176
hidden-k3-s2-r4-n500-01-S1373-238829.shuffled-as.sat03-1035	500	2000	0.00	0.00	92	0.00	0.00	3	0.00	0.00	23

Table 11: SAT2003 competition random benchmarks: comparison among GASAT, Walksat and UnitWalk. The running time is limited to 1 hour and the number of authorized performed flips is 101×10^5. GASAT obtains the best average rank.

Benchmarks			GASAT			Walksat			UnitWalk		
instances	var.	cls.	f.c. avg.	f.c. s.d.	fl. $\times 10^3$	f.c. avg.	f.c. s.d.	fl. $\times 10^3$	f.c. avg.	f.c. s.d.	fl. $\times 10^3$
ferry11u.shuffled-as.sat03-381	3480	25499	1.20	0.40	54731	**0.80**	0.40	1327	2.70	0.90	35
homer19.shuffled-as.sat03-430	330	2340	8.00	0.00	0.135	8.00	0.00	0.167	8.00	0.00	0.171
k2fix_gr_2p-invar_w9.shuffled-as.sat03-436	5028	307674	174.55	10.50	553	**88.25**	8.99	4905	783.75	445.78	9090
am_4_4.shuffled-as.sat03-360	433	1458	1.15	0.65	295	1.00	0.00	5	1.00	0.00	0.458
cnf-r4-b1-k1.1-comp.shuffled-as.sat03-416	2424	14812	20.80	2.98	135	20.95	1.43	4976	**19.20**	1.57	68
cnt10.shuffled-as.sat03-418	20470	68561	163.65	22.39	966	**32.50**	5.79	5624	577.50	40.37	78
dp11u10.shuffled-as.sat03-422	9197	25271	79.67	9.52	873	**1.05**	0.22	3158	14.15	2.01	113
frg1mul.miter.-shuffled-as.sat03-351	3230	20575	5.89	0.94	164	1.00	0.00	6	1.00	0.00	2
gripper11u.-shuffled-as.sat03-391	3084	26019	8.15	0.36	1710	**2.95**	0.38	2528	4.20	0.87	29

Table 12: SAT2003 competition industrial benchmarks: comparison among GASAT, Walksat and UnitWalk. The running time is limited to 1 hour and the number of authorized performed flips is 101×10^5. Walksat obtains the best average rank.

Benchmarks			GASAT			Walksat			UnitWalk		
instances	var.	cls.	f.c. avg.	f.c. s.d.	fl. $\times 10^3$	f.c. avg.	f.c. s.d.	fl. $\times 10^3$	f.c. avg.	f.c. s.d.	fl. $\times 10^3$
k2fix_gr_2p-invar_w9.shuffled-as.sat03-436	5028	307674	**78.40**	7.40	2586	88.25	8.99	4905	783.75	445.78	9090
cnt10.shuffled-as.sat03-418	20470	68561	59.75	7.63	1028	**32.50**	5.79	5624	577.50	40.37	78
dp11u10.shuffled-as.sat03-422	9197	25271	79.05	9.93	910	**1.05**	0.22	3158	14.15	2.01	113
frg1mul.miter.-shuffled-as.sat03-351	3230	20575	1.00	0.00	312	1.00	0.00	6	1.00	0.00	2

Table 13: Large SAT2003 competition industrial benchmarks where the GASAT search power is restricted by the time. The running time is NOT LIMITED and the number of authorized performed flips is 101×10^5.

on three of the four instances. The results of Walksat and UnitWalk remain the same since they had reached their maximum number of flips before the one hour cut-off limit.

All the instances:

The results presented in Tables 10, 11 and 12 show that GASAT behaves reliably and that it provides competitive results on average. For the three tables, its overall rank

is 1.87 whereas Walksat obtains 1.98 and UnitWalk 2.15. A lot of studied instances are not solvable and therefore GASAT appears very efficient for the MAX-SAT problem.

7.3 Remarks

Tables 8 and 11 show that evolutionary algorithms give very competitive results on random instances with few variables in comparison with local search algorithms. Evolutionary algorithms explore more quickly small search spaces with the crossover process than the local search algorithms with their step by step process. But usually, incomplete solvers are used on large instances since, due to the size of the problem, complete solvers become impractical. GASAT is not compared with complete solvers since some of the used instances are too large and seem to be UNSAT. For these instances, we are more interested in solving the MAX-SAT problem in order to provide fair comparisons with other incomplete solvers.

Computation time:

For all the tests, we have noticed that GASAT needs more time to execute the same number of flips than the other solvers. This may be due to the RCVF mechanism which needs to compute the weight of each possible variable. To compare it with state-of-the-art solvers, we have submitted it to the SAT 2004 competition where the execution time is limited but not the number of moves. GASAT ranks fourth [7] (for random instances) w.r.t. more than 50 SAT solvers, outperforming even UnitWalk and Walksat. Although GASAT makes less flips than other solvers within the same time, it obtains very good results in terms of quality.

8 Conclusion

In this paper, we presented the GASAT algorithm, a hybrid genetic algorithm for the SAT (and MAX-SAT) problem. GASAT includes a crossover operator which relies on the structure of the clauses and a Tabu Search with specific mechanisms. These two processes are complementary and they allow GASAT to explore the search space and to exploit particular interesting areas in a more effective manner. Moreover, some mechanisms have been added in order to insure a sufficient diversity in the involved populations. GASAT has been evaluated on both random and structured instances and has been compared with evolutionary algorithms like FlipGA and with two state-of-the-art algorithms: Walksat and UnitWalk. Experimental results show that GASAT is very competitive compared with existing evolutionary algorithms, Walksat and UnitWalk. GASAT also provides very interesting results and appears very effective for the MAX-SAT problem. These performances were confirmed during the SAT 2004 competition.

Our future work will consist of developing a better understanding of GASAT behavior with respect to the different families of benchmarks. This will enable us to improve our CC crossover with a dynamic approach and to develop more efficient interactivity between crossovers and RTS. We are also working on other hybridizations including complete resolution techniques.

Acknowledgments

The work presented in this paper is partially supported by the CPER COM program. We would like to thank the referees of the paper for their useful comments and Claudio Rossi who has provided us with the FlipGA source code.

[7] The first three algorithms were `adaptnovelty`, `saps` and `walksatrnp` and all based on local search.

References

Benhamou, B. and Sais, L. (1992). Theoretical study of symmetries in propositional calculus and applications. In *CADE'92*, pages 281–294.

Beresin, M., Levine, E., and Winn, J. (1989). A chessboard coloring problem. *The College Mathematics Journal*, 20(2):106–114.

Biere, A., Cimatti, A., Clarke, E. M., Fujita, M., and Zhu, Y. (1999). Symbolic model checking using SAT procedures instead of BDDs. In *Proc. of the Design Automation Conference (DAC'99)*, pages 317–320.

Corne, D., Dorigo, M., Glover, F., Dasgupta, D., Moscato, P., Poli, R., and Price, K. V., editors (1999). *New Ideas in Optimization (Part 4: Memetic Algorithms)*. McGraw-Hill.

Davis, M., Logemann, G., and Loveland, D. (1962). A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397.

De Jong, K. A. and Spears, W. M. (1989). Using genetic algorithm to solve NP-complete problems. In *Proc. of the 3rd International Conference on Genetic Algorithms (ICGA'89)*, pages 124–132, Virginia,USA.

Dubois, O., André, P., Boufkhad, Y., and Carlier, J. (1996). SAT versus UNSAT. In *Second DIMACS Implementation Challenge: Cliques, Coloring and Satisfiability*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 415–436.

Dubois, O. and Dequen, G. (2001). A backbone-search heuristic for efficient solving of hard 3-SAT formulae. In Nebel, B., editor, *Proc. of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01)*, pages 248–253, San Francisco, CA.

Eiben, A. E. and van der Hauw, J. K. (1997). Solving 3-SAT by GAs adapting constraint weights. In *Proc. of The IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, pages 81–86.

Fleurent, C. and Ferland, J. A. (1996). Object-oriented implementation of heuristic search methods for graph coloring, maximum clique, and satisfiability. In *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 619–652.

Galinier, P. and Hao, J.-K. (1999). Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3(4):379–397.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability , A Guide to the Theory of NP-Completeness*. W.H. Freeman & Company, San Francisco.

Glover, F. and Laguna, M. (1997). *Tabu Search*. Kluwer Academic Publishers.

Gottlieb, J., Marchiori, E., and Rossi, C. (2002). Evolutionary algorithms for the satisfiability problem. *Evolutionary Computation*, 10(1):35–50.

Gottlieb, J. and Voss, N. (2000). Adaptive fitness functions for the satisfiability problem. In Hans-Paul Schwefel, M., editor, *Parallel Problem Solving from Nature - PPSN VI 6th International Conference*, Paris, France. Springer Verlag. LNCS 1917.

Gu, J. and Puri, R. (1995). Asynchronous circuit synthesis with boolean satisfiability. *IEEE Transactions on Computer-Aided Design*, 14(8):961–973.

Hansen, P. and Jaumard, B. (1990). Algorithms for the maximum satisfiability problem. *Computing*, 44(4):279–303.

Hao, J.-K. and Dorne, R. (1994). A new population-based method for satisfiability problems. In *Proc. of the 11th European Conf. on Artificial Intelligence*, pages 135–139, Amsterdam.

Hao, J.-K., Lardeux, F., and Saubion, F. (2003). Evolutionary computing for the satisfiability problem. In *Applications of Evolutionary Computing*, volume 2611 of *LNCS*, pages 258–267, University of Essex, England, UK.

Hart, W. E., Krasnogor, N., and Smith, J. E., editors (2004). *Recent Advances in Memetic Algorithms and Related Search Technologies*. Springer-Verlag.

Hirsch, E. A. and Kojevnikov, A. (2001). UnitWalk: A new SAT solver that uses local search guided by unit clause elimination. PDMI preprint 9/2001, Steklov Institute of Mathematics at St. Petersburg.

Jaumard, B., Stan, M., and Desrosiers, J. (1996). Tabu search and a quadratic relaxation for the satisfiability problem. In *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 457–478.

Kautz, H. A. and Selman, B. (2001). Workshop on theory and applications of satisfiability testing (SAT2001). In *Electronic Notes in Discrete Mathematics*, volume 9.

Li, C. M. (2000). Integrating equivalency reasoning into davis-putnam procedure. In *Proc. of the AAAI'00*, pages 291–296.

Li, C. M. and Anbulagan, A. (1997). Heuristics based on unit propagation for satisfiability problems. In *Proc. of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97)*, pages 366–371.

Li, C. M., Jurkowiak, B., and Purdom, P. W. (2002). Integrating symmetry breaking into a dll procedure. In *Fifth International Symposium on the Theory and Applications of Satisfiability Testing (SAT2002)*, pages 149–155.

Marchiori, E. and Rossi, C. (1999). A flipping genetic algorithm for hard 3-SAT problems. In *Proc. of the Genetic and Evolutionary Computation Conference*, volume 1, pages 393–400.

Mazure, B., Sais, L., and Grégoire, E. (1997). Tabu search for SAT. In *Proc. of the AAAI-97/IAAI-97*, pages 281–285, Providence, Rhode Island.

Merz, P. and Freisleben, B. (1997). Genetic local search for the TSP: New results. In *IEEE-CEP: Proc. of The IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, pages 159–164.

Mitchell, D. G., Selman, B., and Levesque, H. J. (1992). Hard and easy distributions for SAT problems. In *Proc. of AAAI'92*, pages 459–465.

Monasson, R., Zecchina, R., Kirkpatrick, S., Selman, B., and Troyansky, L. (1999). Determining computational complexity from characteristic 'phase transitions'. *Nature*, 400(8):133–137.

Rossi, C., Marchiori, E., and Kok, J. N. (2000). An adaptive evolutionary algorithm for the satisfiability problem. In *Proc. of the ACM Symposium on Applied Computing (SAC '00)*, pages 463–470. ACM press.

Selman, B., Kautz, H. A., and Cohen, B. (1994). Noise strategies for improving local search. In *Proc. of the AAAI'94, Vol. 1*, pages 337–343.

Selman, B., Levesque, H. J., and Mitchell, D. G. (1992). A new method for solving hard satisfiability problems. In *Proc. of the AAAI'92*, pages 440–446, San Jose, CA.

Simon, L., Berre, D. L., and Hirsch, E. A. (2002). The SAT2002 competition. Technical report, Fifth International Symposium on the Theory and Applications of Satisfiability Testing.

Spears, W. M. (1996). Simulated annealing for hard satisfiability problems. In *Second DIMACS Implementation Challenge: Cliques, Coloring and Satisfiability*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 533–558.

Syswerda, G. (1989). Uniform crossover in genetic algorithms. In *Proc. of the 3rd International Conference on Genetic Algorithms (ICGA'89)*, pages 2–9, Virginia, USA.

Van Gelder, A. (1993). Problem generator mkcnf.c. DIMACS, Challenge archive.

Zhang, H. (1997). SATO: An efficient propositional prover. In *Proc. of the 14th International Conference on Automated Deduction*, volume 1249 of *LNAI*, pages 272–275, Berlin.

Zhang, H. (2002). Generating college conference basketball schedules by a SAT solver. In *Proc. of 5th International Symposium on the Theory and Applications of Satisfiability Testing*, pages 281–291.