Recombination Operators for Satisfiability Problems

Frédéric Lardeux, Frédéric Saubion and Jin-Kao Hao

LERIA, University of Angers
2 Bd Lavoisier, 49045 Angers Cedex 01, France
Frederic.Lardeux@univ-angers.fr
Frederic.Saubion@univ-angers.fr
Jin-Kao.Hao@univ-angers.fr

Abstract. In this paper, we present several recombination operators that are specially designed for SAT problems. These operators take into account the semantic information induced by the structure of the given problem instance under consideration. Studies are carried out to assess the relative performance of these recombination operators on the one hand, and to show the high effectiveness of one of them when it is embedded into a hybrid genetic local search algorithm on the other hand.

1 Introduction

The satisfiability problem (SAT) [6] consists in finding a truth assignment that satisfies a well-formed Boolean expression. An instance of the SAT problem is defined by a set of boolean variables (also called atoms) $\mathcal{X} = \{x_1, ..., x_n\}$ and a boolean formula $\phi: \{0,1\}^n \rightarrow \{0,1\}$. A literal is a variable or its negation. A (truth) assignment is a function $v: \mathcal{X} \rightarrow \{0,1\}$. The formula is said to be satisfiable if there exists an assignment satisfying ϕ and unsatisfiable otherwise. The formula ϕ is in conjunctive normal form (CNF) if it is a conjunction of clauses where a clause is a disjunction of literals. In this paper, ϕ is supposed to be in CNF.

SAT is originally stated as a decision problem but we are also interesting here in the MAX-SAT problem which consists in finding an assignment which satisfies the maximum number of clauses in ϕ.

SAT and its variants have many practical applications (VLSI test and verification, consistency maintenance, fault diagnosis, planning ...). During the last decade, several improved solution algorithms have been developed and important progress has been achieved. These algorithms can be divided into two main classes:

Complete algorithms: Designed to solve the initial decision problem, the most powerful complete algorithms are based on the Davis-Putnam-Loveland procedure [2]. They differ essentially by the underlying heuristic used for the branching rule [13, 19]. Specific techniques such as symmetry-breaking, backbone detecting or equivalence elimination are also used to reinforce these algorithms [1,4,12].

Incomplete algorithms: They are mainly based on local search [8,10,11,15] and evolutionary algorithms [3,5,14,7,9]. Walksat [15] and UnitWalk [10] are famous examples of incomplete algorithms. Incomplete algorithms represent one of the most powerful approaches for finding models of very large SAT instances and for solving large scale MAX-SAT instances.

In this paper, we are interested in the hybrid genetic local search approach for SAT and in particular the design of recombination (crossover) operators. We introduce two new recombination operators taking into account the constraint structure of the given problem instance under consideration.

To assess the performance of these recombination operators, we compare them with two other crossover operators. We also study the combined effect of our crossover operators when they are embedded into a genetic local search algorithm. We show experimentally that such a hybrid algorithm is able to produce highly competitive results compared with other state of the art evolutionary algorithms as well as with WalkSat and UnitWalk, two leading SAT solvers.

In the next section, we present the hybrid evolutionary framework for SAT. We propose then an analysis of the recombination operators and of the interaction between LS and recombination operators. At last, we compare our hybrid algorithm to other well-known algorithms in order to evaluate its performance.

2 Hybrid Evolutionary Algorithm Framework

In this section, we define the main lines of a hybrid evolutionary algorithm obtained by mixing a recombination stage with a local search (LS) process. Given an initial population, the first step consists in selecting some best elements (i.e. assignments) according to a fitness function *eval*. Then, recombinations are performed on this selected population. Each child is individually improved using a local search process and inserted under certain conditions in the population. The general algorithm is described in figure 1. It is clear that this algorithm can be adjusted by changing the selection function, the recombination operator or the local search method but also by modifying the insertion conditions.

Representation The most obvious way to represent an individual for a SAT instance with n variables is a string of n bits where each variable is associated to one bit. Other representation schemes are discussed in [7]. Therefore the search space is the set $\mathcal{S} = \{0,1\}^n$ (i.e. all the possible strings of n bits) and an individual X obviously corresponds to an assignment. $X|i$ denotes the truth value of the i^{th} atom. Given an individual X and a clause c, we use the boolean function $sat(X,c)$ to denote the fact that the assignment associated to X satisfies the clause c.

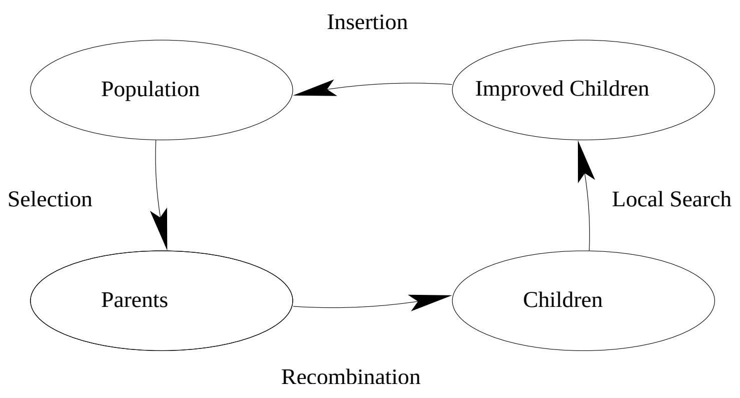

Fig. 1. Algorithm Scheme

Fitness Function Given a formula ϕ and an individual X, the fitness of X is defined to be the number of clauses which are not satisfied by X:

$$\begin{aligned} eval\colon\ & \mathcal{S} \rightarrow I\!N \\ & X \mapsto card(\{c \mid \neg sat(X, c) \wedge c \in \phi\}) \end{aligned}$$

where *card* denotes as usual the cardinality of a set. The smallest value of this function equals 0 if ϕ is satisfiable and an individual having this fitness value corresponds to a satisfying solution.

Selection Process The selection operator is a function which takes as input a given population and extracts some individuals which will serve as parents for the recombination stage. To insure an efficient search, it is necessary to keep some diversity in the population. Indeed, if the selected parents are too similar, some region of the search space $\mathcal{S}$ will not be explored.

Recombination This operator creates new assignments from two selected parent assignments. Classical random crossover may be used. However, as we show later in the paper, specific crossovers that are meaningful to the SAT problem may be much more powerful.

Local Search The local search process improves a configuration by a sequence of small changes in order to reach a local optimum according to the evaluation function, a good neighborhood function [18] and some mechanisms like tabu list, random walk, and so on.

Insertion An improved child can be inserted according to different conditions (e.g. if it is better than any individual of the population). These conditions are important with respect to the average equality and the diversity of the population. A child is inserted in the population if it is better than the worst of the sub-population of possible parents.

3 Recombination Operators

A recombination operator (also called crossover) has to take into account as much as possible the semantics of the individuals in order to control the general search process and to obtain an efficient algorithm. In SAT and MAX-SAT, the variables are the atoms and a constraint structure is induced by the clauses of the given formula. Therefore, while the representation focuses on the atoms, an efficient crossover should take into account this whole constraint structure.

We first define a function allowing to change the value of the variables:

$$\begin{aligned} \mathit{flip}\colon \{0,1\} &\rightarrow \{0,1\} \\ x &\mapsto 1-x \end{aligned}$$

Then, we define an improvement function:

$$\begin{aligned} \mathit{imp}\colon \mathcal{S} \times \mathbb{N} &\rightarrow \mathbb{N} \\ (X,i) &\mapsto \mathit{card}(\{c \mid \mathit{sat}(X[i \leftarrow \mathit{flip}(X|i)],c) \wedge \neg \mathit{sat}(X,c)\}) \\ &\quad -\mathit{card}(\{c \mid \neg \mathit{sat}(X[i \leftarrow \mathit{flip}(X|i)],c) \wedge \mathit{sat}(X,c)\}) \end{aligned}$$

This function computes the improvement obtained by flipping the i^{th} component of X and was previously introduced in GSAT and Walksat [16, 15]. It corresponds to the gain of the solution according to the function *eval* (i.e. the number of false clauses which become true by flipping the atom i minus the number of satisfied clauses which become false). Remark that if this number is negative then the number of false clauses increases if the flip is performed.

In order to take into account the specific structures of a problem, we will use the relations over the variables induced by the clauses. There are three possibilities as to the satisfaction of a clause under the two parents:

- the clause is satisfied in the two parents,
- the clause is unsatisfied in the two parents,
- the clause is satisfied in only one parent.

The aim of our recombination is then to obtain a child which benefits from the parents but with the maximum number of true clauses. Therefore, one should try to correct false clauses and to maintain true ones.

When a clause c is false for the two parents, a possible solution to turn c to true is to flip a variable of c [1]. Therefore, the child receives an opposite value for this variable with respect to its parents. Unfortunately, this action may produce false clauses. To limit this number of appearing false clauses, the choice of the variable is guided by the *imp* evaluation function.

[1] If a clause is false for both parents, then all the variables appearing in this clause have necessarily the same value in both parents. This comes from the fact that a clause can be false only if all of its literals are false.

When a clause c is true for the two parents, we may copy in the child all the values assigned to the variables of c in one of the parent in the child. Unfortunately, this action would only take into account the structure of the chosen parent. To be fair, we should copy values of variables coming from both parents. But, these values can be different. A solution is to select the variable whose flip has the smallest impact and to put its value such that the corresponding literal is true in c. Since only one variable is necessary to maintain this clause true, we may again guide this operation with the *imp* function.

Finally, when a clause c is true for one parent and false for the other, the solution proposed by Fleurent and Ferland [5] is: "The corresponding variables [to this clause] are assigned values according to the parent satisfying the identified clause."

It is clear that the order in which the clauses are traversed is important. Here, they are traversed in the same order that they appear in the studied benchmark.

We now present the definitions of four crossovers. Three of them are structured operators using the previous remarks. We present also the classical uniform crossover, which is used as a reference for comparisons. Each operator is a function *cross*: $\mathcal{S} \times \mathcal{S} \rightarrow \mathcal{S}$ (i.e. two parents produce one child).

Let X and Y be two parents and Z be the resulting child.

Uniform Crossover
Each variable of Z takes the value of X or Y with equal probability.

Corrective Clause crossover (CC)
For each clause c such that $\neg sat(X,c) \wedge \neg sat(Y,c) \wedge \neg sat(Z,c)$ and for all positions i such that the variable x_i appears in c, we compute $\sigma = imp(X,i) + imp(Y,i)$ and we 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.

Corrective Clause and Truth Maintenance crossover (CCTM)
For each clause c such that $\neg sat(X,c) \wedge \neg sat(Y,c) \wedge \neg sat(Z,c)$ and for all positions i such that the variable x_i appears in c, we compute $\sigma = imp(X,i) + imp(Y,i)$ and we set $Z|k = flip(X|k)$ where k is the position such that σ is maximum. For all the clauses c such that $sat(X,c) \wedge sat(Y,c)$ and for all positions i such that the variable x_i appears in c and its associated literal is true at least in one parent, we compute $\sigma = imp(X,i) + imp(Y,i)$ and we 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.

Fleurent and Ferland crossover [5] (F&F)
For each clause c such that $sat(X,c) \wedge \neg sat(Y,c)$ (resp. $\neg sat(X,c) \wedge sat(Y,c)$) and for all the positions i such that the variable $x_i \in c$, $Z|i = X|i$ (resp. $Z|i = Y|i$). All the variables of Z with no value take the value of X or Y with equal proba-

bility.

Notice that CC, CCTM and F&F crossovers differ on the use of the truth values of the clauses induced by the parents. As mentioned above, the key idea is to correct false clauses, to preserve true clauses and to maintain the structure of the assignments. In order to study the characteristics of the different operators, we reduce the algorithm to a sequence of recombination stages on a population with or without the selection process between two consecutive stages. The tests are realized with a random 3-sat instance (`f500.cnf`) with 500 variables and a ratio of clauses-to-variables of 4.3. Two measures are used: the profile of *number of crossovers* vs. *fitness* and *number of crossovers* vs. *population entropy*. Entropy expresses the diversity of the population. The smallest value is 0, indicating that all the individual of the population are the same. The largest value is 1, indicating that all the individuals are different.

Definition 1. *Entropy [5] Let* n_{ij} *= number of times where the variable* i *is set to* j *in the population* P.

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

The population size is 100 and the sub-population of possible parents has a size of 15. Two parents are randomly chosen in this sub-population. A child is inserted in the population if its fitness is better than the fitness of the worst individual of the sub-population.

Figure 2 shows the performances of the four crossover operators combined with or without selection according to the two measures. One observes that:

- the selection damages the F&F crossover behavior,
- the selection improves the results of the uniform crossover, the CCTM crossover and the CC crossover,
- the uniform crossover does not provide good results with or without selection,
- the CCTM crossover and the CC crossover have a better entropy than the F&F crossover,
- the CC crossover and the CCTM crossover with the selection process provide very good results with a diversity higher than all the other crossovers.

Therefore, an efficient crossover is not necessary a crossover which quickly improves the whole population.Rather, the crossover should be able to ensure a good trade-off between quality and diversification of the population. The diversification allows a better exploration of the search space and prevents the population from stagnating in poor local optima.

4 Crossovers and Local Search

A good crossover insures a meaningful diversification leading the search to promising areas. Now it is necessary to add LS to perform an intensified search of solu-

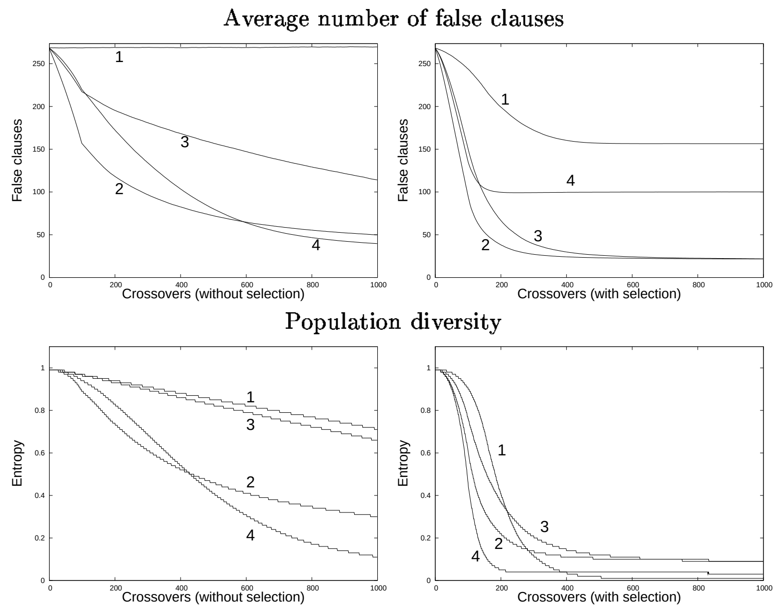

Fig. 2. The left column shows the crossovers without the selection stage and the right column shows the crossovers with the selection stage (1-Uniform crossover, 2-CCTM crossover, 3-CC crossover and 4-F&F crossover)

tions in the identified particular areas. Based on this idea, the GASAT algorithm is a hybridization of LS and a specific recombination operator [9].

To assess further the performance of each crossover operator within a whole algorithm, we insert each of this four cross operators in GASAT and employ a tabu search as local searcher. Now we run with the conditions detailed in section 5.2 each of the four hybrid algorithms on four instances. `3blocks` is a blocks world instance, `par16-4-c` is a parity function instance and `f1000` and `color10-3` are presented in details in section 5.2. Experimental conditions are the same for the four algorithms. Notice that the F&F crossover is used without the selection process since this gives better results (figure 2).

Two comparison criteria are used in order to evaluate the different crossovers improved by a LS. The first criterion is the success rate (%) which corresponds to the number of successful runs divided by the number of tries. The second criterion is the average number of crossovers (cr.) for the successful runs.

Table 1 shows a slight dominance of the CC crossover. These results show the interaction between CC and LS is more powerful than the interactions between the other crossovers and LS operator.

	CC		CCTM		F&F		Uniform	
instances	%	cr.	%	cr.	%	cr.	%	cr.
3blocks	10.00	439.00	*(1 cl.)*		*(1 cl.)*		*(1 cl.)*	
color-10-3	100.00	287.00	93.33	224.06	96.66	255.20	93.33	218.00
f1000	90.35	235.31	66.66	177.38	83.33	189.90	80.00	146.08
par16-4-c	5.00	788.00	10.00	206.00	*(2.80 cl.)*		*(2.75 cl.)*	

Table 1. Comparison of different crossovers included in the GASAT algorithm (if no assignment is found then the average number of false clauses is given)

5 Experimental Results with the CC crossover

In this section, we evaluate the GASAT algorithm with the CC crossover on different classes of benchmarks. We compare first GASAT with five evolutionary algorithms presented in [7]. Then we show comparisons with the well known Walksat [15] and UnitWalk [10], a winner of the SAT2003 competition. All our tests are realized on a cluster with Linux and Alinka (5 nodes with each of them 2 CPU Pentium IV 2, 2 Ghz and 1 GB of RAM) used in sequential run.

5.1 Comparisons with evolutionary algorithms from [7]

The purpose of this section is to compare GASAT with several state-of-the-art evolutionary algorithms presented in [7]. To make the comparison as fair as possible, we run GASAT under the same condition as that used in [7]. More precisely, each problem instance is solved between 5 and 50 times, each run being limited to 3×10^5. The benchmarks used in this experimentation are:

- suite A, 4 groups of 3 instances with 30, 50, 75 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.

All these instances are random 3-sat instances with a ratio clauses-to-variables of 4.3.

Results of GASAT are shown in table 2, together with the results of the five evolutionary algorithms reported in [7]. One observes that on these random instances GASAT with CC crossover does not have the best success rate but requires in general much fewer flips to obtain a solution. Next section will show more results of GASAT on much larger random instances as well as structured instances.

5.2 Comparison with Walksat and UnitWalk

Due to the incomplete and non deterministic character of GASAT, Walksat and UnitWalk, each algorithm has been run 20 times on each benchmark with an execution time limited to 1 hour. These algorithms are used with their standard parameters. The number of flips is limited to 101×10^5 for the three algorithms.

Instances			GASAT		SAWEA		RFEA2		RFEA2+		FlipGA		ASAP	
Suite	nb.	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 2. Comparison between evolutionary algorithms (%-success rate , fl.-average number of flips of successful runs)

We provide now the precise parameters for each algorithm.

GASAT: GASAT works with a population of 10^2 individuals. During the generation of this population, a LS of 10^3 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 a LS (tabu search) of at most 10^4 flips is applied on each child. The size of the tabu list is set to 10% of the number of variables in the problem.

Walksat[2]**:** We use the version v41. The number of tries is 10 with at most 101×10^4 flips for each try. Walksat uses the "novelty" heuristic with a noise set to 0.5.

UnitWalk[3]**:** We use UnitWalk version 0.981. The maximum number of flips is 101×10^5 with 1 run.

Benchmarks Two classes of instances have been used: structured instances and random instances. Some instances are satisfiable and others are unsatisfiable. These instances[4] were used by the SAT2002 [17] or SAT2003 competition.

– structured instances:
  · `color-10-3`, `color-18-4`, `color-22-5` (chessboard coloring problem),
  · `difp_19_0_arr_rcr`, `difp_19_99_arr_rcr` (integer factorization),

[2] Walksat is available : `http://www.cs.washington.edu/homes/kautz/walksat/`
[3] UnitWalk is available : `http://logic.pdmi.ras.ru/ arist/UnitWalk/`
[4] Available at `http://www.info.univ-angers.fr/pub/lardeux/SAT/benchmarks-EN.html`

· `g125.17, g125.18` (graph coloring instances),
· `mat25.shuffled`, `mat26.shuffled` (matrix multiplication),
· `par32-5`, `par32-5-c` (problems of learning the parity function).

– random instances:
· `f1000`, `f2000` (DIMACS instances),
· 2 instances of 500 variables generated by **hgen2** with 2 differents seeds,
· 2 instances generated by **glassy** one with 399 and 450 variables.

For each instance, the number of clauses is given in the column *cl* and the number of variables is given in the column *var*. The random seed is also given when it is known.

Evaluation Three comparison criteria are used to evaluate GASAT and to compare it with UnitWalk and Walksat. The first criterion is the success rate (%) which corresponds to the number of success divided by the number of runs. This criterion is important since it highlights the search power of the algorithm. The two other criteria are the average number of flips (fl.) and the average time in seconds (sec.) for the successful runs. The number of flips for GASAT include the flips performed during the crossover operations.

Benchmarks				GASAT			Walksat			UnitWalk [5]		
instances	var	cls	sat	%	fl.[6]	sec.	%	fl.[6]	sec.	%	fl.[6]	sec.
color-10-3	300	6475	Y	100	1046	93.33	100	404	24.41	(*13.00 clauses*)		
color-18-4	1296	95905	?	(*25.95 clauses*)			(*38.00 clauses*)			(*65.00 clauses*)		
color-22-5	2420	272129	?	(*9.10 clauses*)			(*13.60 clauses*)			(*57.00 clauses*)		
difp_19_0_arr_rcr	1201	6563	Y	(*6.75 clauses*)			(*21.90 clauses*)			100	10100	319.53
difp_19_99_arr_rcr	1201	6563	Y	(*9.25 clauses*)			(*21.25 clauses*)			(*88.00 clauses*)		
g125.17	2125	66272	Y	(*3.00 clauses*)			35	458	251.46	(*28.00 clauses*)		
g125.18	2250	70163	Y	70	5501	860.81	100	9	2.40	(*23.00 clauses*)		
mat25.shuffled	588	1968	N	(*3.35 clauses*)			(*3.00 clauses*)			(*28.00 clauses*)		
mat26.shuffled	744	2464	N	(*3.20 clauses*)			(*2.55 clauses*)			(*25.00 clauses*)		
par32-5-c	1339	5350	Y	(*5.90 clauses*)			(*12.70 clauses*)			(*60.00 clauses*)		
par32-5	3176	10325	Y	(*4.15 clauses*)			(*12.10 clauses*)			10	10100	191.65

Table 3. Structured instances (if no assignment is found then the average number of false clauses is given between parentheses, ? indicates the satisfiability of an instance is unknown)

Results Analysis The results given in table 3 and table 4 do not show a clear supremacy of one algorithm over other ones. However, GASAT is generally more

[5] When no assignment is found UnitWalk does not give the best number of false clauses but only the last found.
[6] You must multiply by 10^3 to obtain the real number of flips.

Benchmarks					GASAT			Walksat			UnitWalk [5]		
gen	seed	var	cls	sat	%	fl.[6]	sec.	%	fl.[6]	sec.	%	fl.[6]	sec.
glassy	1069116088	399	1862	Y	*(5.00 clauses)*			*(5.60 clauses)*			*(43.00 clauses)*		
glassy	325799114	450	2100	Y	*(8.10 clauses)*			*(10.70 clauses)*			*(38.00 clauses)*		
f1000	-	1000	4250	Y	90	2485	35.75	100	367	9.17	100	1386	6.45
f2000	-	2000	8500	Y	5	3417	207.00	70	586	26.65	100	1232	10.41
hgen2	1205525430	500	1750	Y	*(1.00 clause)*			*(1.00 clause)*			*(8.00 clauses)*		
hgen2	512100147	500	1750	Y	*(1.00 clause)*			*(1.00 clause)*			*(21.00 clauses)*		

Table 4. Random instances (if no assignment is found then the average number of false clauses is given between parentheses, ? indicates the satisfiability of an instance is unknown)

efficient on structured instances than on random instances. These results are not surprising since the CC crossover is based on the idea of exploring clause structure. Taking the two tables together, one may conclude GASAT gives very competitive results on the tested instances.

6 Conclusion

We have presented two new recombination operators for SAT and compared them with two previous known operators. The most powerful crossover is the Corrective Clause crossover. Inserted in the GASAT algorithm, it gives very interesting results. By their global and local nature, the crossover and the LS operators act interactively to ensure a good compromise between exploration and exploitation of the search space. Moreover, a selection mechanism is also employed to preserve the diversity of the population.

GASAT with the CC crossover is evaluated on both structured and random instances. Its performances are compared with two well-known algorithms (Walksat and UnitWalk) and several state-of-the-art evolutionary algorithms. The experimentations show that GASAT gives globally very competitive results, in particular, on structured instances. Meanwhile, it seems that GASAT behaves less well on some random instances.

The new crossovers reported in this paper are a first step. Studies are on the way to have a better interactivity between crossovers and LS. We are also working on other hybridizations such as CC and complete algorithms.

Acknowledgments We would like to thank the anonymous referees for their helpful comments and remarks.

References

1. Belaid Benhamou and Lakhdar Sais. Theoretical study of symmetries in propositional calculus and applications. In *CADE'92*, pages 281–294, 1992.

2. Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, Jul 1962.
3. Kenneth A. De Jong and William M. Spears. Using genetic algorithm to solve NP-complete problems. In *Proc. of the Third Int. Conf. on Genetic Algorithms*, pages 124–132, San Mateo, CA, 1989.
4. Olivier Dubois and Gilles Dequen. A backbone-search heuristic for efficient solving of hard 3-SAT formulae. In Bernhard Nebel, editor, *Proc. of the IJCAI'01*, pages 248–253, San Francisco, CA, 2001.
5. Charles Fleurent and Jacques A. Ferland. 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, 1994.
6. Michael R. Garey and David S. Johnson. *Computers and Intractability , A Guide to the Theory of NP-Completeness.* W.H. Freeman & Company, San Francisco, 1978.
7. Jens Gottlieb, Elena Marchiori, and Claudio Rossi. Evolutionary algorithms for the satisfiability problem. *Evolutionary Computation*, 10(1):35–50, 2002.
8. Pierre Hansen and Brigitte Jaumard. Algorithms for the maximum satisfiability problem. *Computing*, 44(4):279–303, 1990.
9. Jin-Kao Hao, Frédéric Lardeux, and Frédéric Saubion. Evolutionary computing for the satisfiability problem. In *Applications of Evolutionary Computing*, volume 2611 of *LNCS*, pages 259–268, 2003.
10. Edward A. Hirsch and Arist Kojevnikov. 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, 2001.
11. Brigitte Jaumard, Mihnea Stan, and Jacques Desrosiers. 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, 1994.
12. Chu Min Li. Integrating equivalency reasoning into davis-putnam procedure. In *Proc. of the AAAI'00*, pages 291–296, 2000.
13. Chu Min Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proc. of the IJCAI'97*, pages 366–371, 1997.
14. Elena Marchiori and Claudio Rossi. A flipping genetic algorithm for hard 3-SAT problems. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 393–400, 1999.
15. Bart Selman, Henry A. Kautz, and Bram Cohen. Noise strategies for improving local search. In *Proc. of the AAAI, Vol. 1*, pages 337–343, 1994.
16. Bart Selman, Hector J. Levesque, and David G. Mitchell. A new method for solving hard satisfiability problems. In *Proc. of the AAAI'92*, pages 440–446, San Jose, CA, 1992.
17. Laurent Simon, Daniel Le Berre, and Edward A. Hirsch. The SAT2002 competition. *Technical report, Fifth International Symposium on the Theory and Applications of Satisfiability Testing, May 2002.*
18. Mutsunori Yagiura and Toshihide Ibarak. Efficient 2 and 3-flip neighborhood search algorithms for the MAX SAT: Experimental evaluation. *Journal of Heuristics*, 7(5):423–442, 2001.
19. Hantao Zhang. SATO: An efficient propositional prover. In *Proc. of the 14th International Conference on Automated deduction*, volume 1249 of *LNAI*, pages 272–275, Berlin, 1997.