

# Combination of Exact and Approximate Methods for SAT and MAX-SAT Problems

Frédéric Lardeux, Frédéric Saubion and Jin-Kao Hao

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

**Abstract.** Several hybridizations of exact and approximate methods for the SAT and MAX-SAT problems have been studied [16, 3, 9] but these methods are based on different search space representations. The aim of this paper is to propose a new resolution framework which introduces a third truth value *undefined* in order to unify exact and approximate methods and to improve the resolution efficiency. Using this resolution framework, we compare a classic Tabu Search algorithm with the hybridization of a Branch and Bound algorithm and a Tabu Search. Promising results are obtained and show the interest of this resolution framework.

## 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 then defined by a set of boolean variables (also called atoms)  $\mathcal{X} = \{x_1, \dots, x_n\}$  and a boolean formula  $\phi: \mathcal{B}^n \rightarrow \mathcal{B}$ . A literal is a variable or its negation. A truth assignment is a function  $v: \mathcal{X} \rightarrow \mathcal{B}$ . The formula is said to be satisfiable if there exists an assignment satisfying  $\phi$  and unsatisfiable otherwise. The maximum satisfiability problem (MAX-SAT) corresponds to the minimization of the number of false clauses. The formula  $\phi$  is in conjunctive normal form (CNF) if it is a conjunction of clauses where a clause is a disjunction of literals. In this paper,  $\phi$  is supposed to be in CNF.

Two classes of methods can be used to solve SAT and MAX-SAT problems: exact and approximate methods.

- Exact methods are able to find all the solutions of an instance or, if there is no solution, to prove its unsatisfiability. These methods are generally based on the Davis-Putnam-Loveland procedure (DPL) [4] which explores a binary search tree corresponding to the different possible assignments. The introduction of heuristics [14, 20, 16] allows the algorithms to prune this search tree and to guide their exploration in order to increase the resolution efficiency. It is also possible to introduce other specific techniques such as

symmetry-breaking, backbone detection or equivalence elimination to reinforce these algorithms [2, 13, 5]. They provide very good results but are not suitable for the MAX-SAT problem. Other exact methods [1, 21] based on a Branch and Bound (B&B) algorithm have been designed to handle the MAX-SAT problem but their performances are very limited for large instances. In these algorithms, B&B is roughly an extension of the DLP procedure.

- Approximate methods are mainly based on local search (LS) [19, 11, 15] and evolutionary algorithms [8, 10]. They expect to minimize, on the assignments space, a function corresponding to the number of false clauses. Therefore, they are naturally designed for the MAX-SAT problem and allows one to handle large instances.

Hybridizations of these two approaches have been proposed [16, 3, 9] and are mainly based on a *master-slave* combination of a local search process and an exhaustive search. Indeed, in these algorithms, LS is used as a sub-routine in a DLP-like algorithm.

Concerning such hybridizations, we have to remark that, while exact methods work on partial assignment incrementally built by the resolution process, approximate algorithms explore the set of all possible complete assignments. Moreover, approximate methods are originally designed for MAX-SAT while exact methods are more often used for SAT. We obviously consider here SAT as a subcase of MAX-SAT where the number of false clauses is 0. Therefore, our purpose is to provide a uniform framework in order to design hybrid solvers which allows the methods to share a common search space.

This is achieved by introducing a tri-valued logical framework. We propose then a hybrid algorithm for the MAX-SAT problem which uses a Tabu Search (TS) and introduces a classic Branch and Bound algorithm (B&B) as an exact resolution stage. This new framework also allows us to integrate more homogeneously diversification and intensification processes in the TS and in particular an intensification strategy by means of the B&B algorithm.

In the next section we recall the basis of Tabu Search in the SAT context. We then define our tri-valued framework and its integration in the Tabu search process. Section 5 describes how the new TTS algorithm can be tuned before being used in the hybrid algorithm presented in section 6. Experimental results are then discussed before the concluding section.

## 2 Standard Tabu Search for Satisfiability

Tabu Search (TS) [7] is a local search meta-heuristics whose aim is to optimize a function  $f$  on a search space  $\mathcal{S}$ . The exploration of the search space is achieved by moving from a configuration to one of its neighbors. These moves are guided by a fitness function which evaluates their benefit. In order to avoid problems induced by local optima, TS uses a list containing informations on previously visited configurations to forbid some possible loops in the search process. TS has already been studied for the SAT problem in [12, 15].

In this SAT context, the search space is the set of all possible truth assignments and the moves are clearly possible flips of the values of a given assignment. The flip of a variable  $i$  in an assignment  $X$  is the swap of its truth value (*true* to *false* or *false* to *true*) denoted  $flip(X|i)$ . The best flip is selected thanks to a *choose* function which returns the variable whose flip provides the best improvement (i.e., maximizes the number of false clauses which become true after the flip minus the number of satisfied clauses which become false). The number of satisfied clauses is given by the *eval* function:

$$\begin{aligned} eval: \mathcal{S} &\rightarrow \mathbb{N} \\ X &\mapsto |\{c | sat(X, c) \wedge c \in \mathcal{F}\}| \end{aligned}$$

where  $sat(X, c)$  means that the clause  $c$  is satisfied by the assignment  $X$ . A standard TS algorithm for SAT is described in Algorithm 1.

We now introduce our tri-valued framework in order to extend TS to partial assignments.

### 3 A Tri-valued Framework

As mentioned before, exact methods construct their solutions along the search process and therefore generate successive partial assignments to reach a solution. To unify exact and approximate methods, a possible approach consists in extending approximate methods in order to use partial assignments. To introduce such partial assignments in a local search process, we add a truth value *undefined* to the classic *true* and *false* values. This new value induces some changes in the standard logical rules. Two approaches can be used to define these new rules: an optimistic approach and a pessimistic approach.

- The *optimistic approach* is the most commonly used. Indeed, all the exact methods exploring a search tree are based on this approach. A clause is considered undefined if none of its literal is true and at least one of its literal is undefined. This approach corresponds to the following simplification rules:

$$\begin{aligned} true \vee undefined &\rightarrow true \\ false \vee undefined &\rightarrow undefined \\ undefined \vee undefined &\rightarrow undefined \end{aligned}$$

- We propose here an alternative *pessimistic approach* which relies on the fact that as soon as a clause has a false literal and no true literal, it is considered as false. An undefined clause must have all these literals set to *undefined*. This approach corresponds to the following rules:

$$\begin{aligned} true \vee undefined &\rightarrow true \\ false \vee undefined &\rightarrow false \\ undefined \vee undefined &\rightarrow undefined \end{aligned}$$

We may now adapt TS to this tri-valued context.

**Data** : a formula  $\phi$ , an assignment  $Z$ ,  $\lambda$ ,  $\text{Maxflip}$   
**Result**: the best assignment found

```

begin
  Set the list tabu to size  $\lambda$ ;
   $Best \leftarrow Z$ ;
   $nbflip \leftarrow 0$ ;
  while not ( $eval(Best) = nb \text{ clauses in } \phi \vee nbflip > \text{Maxflip}$ ) do
     $i \leftarrow choose(Z)$  (*);
    if  $i \notin tabu$  then
      if  $eval(Z[i \leftarrow flip(Z|i)]) > eval(Best)$  then
         $Best \leftarrow Z[i \leftarrow flip(Z|i)]$ ;
      end
    else
      /* Aspiration Criteria */ ;
      if  $eval(Z[i \leftarrow flip(Z|i)]) > eval(Best)$  then
         $Best \leftarrow Z[i \leftarrow flip(Z|i)]$ ;
      else
        go to (*) with the condition that  $i$  is different than the
        actual  $i$ 
      end
    end
     $nbflip \leftarrow nbflip + 1$  ;
     $Z \leftarrow Z[i \leftarrow flip(Z|i)]$  ;
    remove the older index from tabu ;
    add  $i$  to tabu ;
  end
  Return  $Best$  ;
end

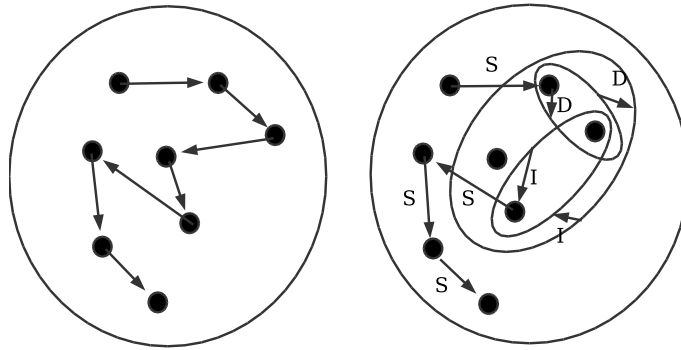
```

**Algorithm 1:** Tabu Search (TS)

## 4 TTS: a Tri-valued Tabu Search

The third truth value will allow TS to diversify its search without the addition of an external heuristic: whereas a classic TS with two truth values explores the search space moving from an assignment to another, TS with three truth values (TTS) introduces moves across areas of the search space thanks to partial assignments, since a partial assignment can be actually considered as a set of complete assignments. For instance  $T T F U T U$  represents  $\{ T T F F T F, T T F F T T, T T F T T F, T T F T T T \}$ .

The general TS and TTS processes share common principles. The main difference is that the valuation of a variable to *undefined* induces a diversification step whereas the valuation to *true* or to *false* of an undefined variable induces an intensification step (Fig.1). It is important to remark that contrary to the TS, TTS can start its search with an empty assignment. The influence of the initial assignment in the results is then avoided.



**Fig. 1.** TS (left) and TTS (right). S represents a standard move with two truth values. A diversification (D) step occurs when a variable is valued to *undefined* whereas an intensification step (I) occurs when a variable undefined is valued to *true* or *false*.

The introduction of the third truth value requires to redefine the *eval* function and the *choose* function of section 2 since the number of undefined clauses must be taken into account. The new evaluation function  $eval'$  returns the number of true clauses as well as the number of undefined clauses. In this context, an assignment is better than another if it satisfies more clauses. If the number of true clauses is equal for two assignments, the one generating the greatest number

of undefined clauses will be considered as the best assignment.

$$\begin{aligned}
eval': \mathcal{S} &\rightarrow (\mathbb{N}, \mathbb{N}) \\
X &\mapsto (|\{c \mid sat(X, c) \wedge c \in \mathcal{F}\}|, \\
&|\{c \mid c \in \mathcal{F}\}| - |\{c \mid sat(X, c) \wedge c \in \mathcal{F}\}| - |\{c \mid \neg sat(X, c) \wedge c \in \mathcal{F}\}|)
\end{aligned}$$

We define the order  $>_{eval'}$  as the lexicographic extension ( $>, >$ ) of the order  $>$  on the pair constructed by the  $eval'$  function.

The purpose of the search process is to increase the number of true clauses. When this is not possible, it is then preferable to increase the number of undefined clauses and thus to reduce the number of false clauses. We propose two different *choose* functions:

- The *choose1* function (Algorithm 2) maximizes the number of true clauses and minimizes the number of false clauses (increasing the number of undefined clauses if necessary).
- The *choose2* function (Algorithm 3) increases the number of true clauses but, if it is not possible, it maximizes the difference between the number of false clauses and the number of true clauses, keeping a maximum number of true clauses.

**Data** : an assignment  $S$

**Result**: a variable, a truth value

**begin**

$E = analyze(S);$

Select randomly a triplet among the triplets of  $E$  having the higher  $maxT$  value and the higher  $maxU$  value;

Return the variable as well as the truth value having allowed to obtain this triplet;

**end**

**Algorithm 2:** *Choose1* function for TS with three truth values

These two functions use the *analyze* function (see Algorithm 4) which computes the impact of each move on the number of true, false and undefined clauses. In this function,  $Cl_{X \rightarrow Y}[i \leftarrow Z]$  denotes the number of clauses which truth value  $X$  becomes  $Y$  when the variable  $i$  is valued with the truth value  $Z$ .

At this step, we have to tune our TTS in order to prepare its hybridization with B&B.

**Data** : an assignment  $S$   
**Result**: a variable, a truth value  
**begin**  
     $E = \text{analyze}(S)$ ;  
    Select randomly a triplet among the triplets of  $E$  having the higher  $\text{max}T$  value and the higher  $\text{max}U$  value;  
    **if**  $\text{max}T > \text{number of true clauses with } S$  **then**  
        Return the variable as well as the truth value having allowed to obtain this triplet;  
    **else**  
        Select randomly a triplet among the triplets of  $E$  maximizing  $(\text{number of false clauses with } S - \text{max}F) - (\text{number of true clauses with } S - \text{max}T)$ ;  
        Return the variable as well as the truth value having allowed to obtain this triplet;  
    **end**  
**end**

**Algorithm 3:** *Choose2* function for TS with three truth values

## 5 Tuning TTS for Hybridization

As described before, TTS provides a partial assignment which represents indeed a subset of assignments. To evaluate the quality of this partial assignment, it is necessary to explore exhaustively the corresponding subset in order to find the best complete assignment that it contains. This can be achieved thanks to a B&B algorithm which performs an exhaustive search by exploring a restricted search tree whose root is the partial assignment provided by TTS.

Table 1 compares the different approaches concerning logical rules and the different selection functions. Tests are realized on random 3-sat instances with 500, 1000 and 2000 variables (F500.cnf, F1000.cnf and F2000.cnf). The length of the tabu list is set to 10% of the number of variables and 5000 iterations are executed by TTS. Along the TTS process, the number of undefined variables (*nb. var. U*) and the number of clauses with one, two and three undefined variables ( $(U, UU, UUU)$ ) are memorized for the last partial assignment found ( $(T, U, F)$  after TS). To estimate the potential quality of the partial assignment found by TTS, a B&B algorithm is executed and provides the best complete assignment ( $(T, F)$  after B&B) by assigning the undefined variables.

Table 1 shows a clear dominance of the *choose1* selection function. Using this selection function, the best approach is our pessimistic approach (section 3). The quality of the partial assignment found during the TTS is better because the number of true clauses is greater than the number obtained with the other combinations. Since our motivation is to hybridize TTS with an exact method,

**Data** : an assignment  $S$   
**Result**: a set  $E$  of triplets

```

begin
   $E = \{\}$ ;
  for all the variables  $i$  do
    /* Compute the number of true, false and undefined clauses after the
    valuation to TRUE of the variable  $i^*$ */;
    if  $value(i) \neq TRUE$  then
       $maxT = \text{number of true clauses with } S + Cl_{F \rightarrow T}[i \leftarrow T] + Cl_{U \rightarrow T}[i \leftarrow T] - Cl_{T \rightarrow F}[i \leftarrow T] - Cl_{T \rightarrow U}[i \leftarrow T]$ ;
       $maxF = \text{number of false clauses with } S + Cl_{T \rightarrow F}[i \leftarrow T] + Cl_{U \rightarrow F}[i \leftarrow T] - Cl_{F \rightarrow T}[i \leftarrow T] - Cl_{F \rightarrow U}[i \leftarrow T]$ ;
       $maxU = \text{number of undefined clauses with } S + Cl_{T \rightarrow U}[i \leftarrow T] - Cl_{U \rightarrow T}[i \leftarrow T] - Cl_{U \rightarrow F}[i \leftarrow T]$ ;
    end
    /* Compute the number of true, false and undefined clauses after the
    valuation to FALSE of the variable  $i^*$ */;
    if  $value(i) \neq FALSE$  then
       $maxT = \text{number of true clauses with } S + Cl_{F \rightarrow T}[i \leftarrow F] + Cl_{U \rightarrow T}[i \leftarrow F] - Cl_{T \rightarrow F}[i \leftarrow F] - Cl_{T \rightarrow U}[i \leftarrow F]$ ;
       $maxF = \text{number of false clauses with } S + Cl_{T \rightarrow F}[i \leftarrow F] + Cl_{U \rightarrow F}[i \leftarrow F] - Cl_{F \rightarrow T}[i \leftarrow F] - Cl_{F \rightarrow U}[i \leftarrow F]$ ;
       $maxU = \text{number of undefined clauses with } S + Cl_{T \rightarrow U}[i \leftarrow F] - Cl_{U \rightarrow T}[i \leftarrow F] - Cl_{U \rightarrow F}[i \leftarrow F]$ ;
    end
    /* Compute the number of true, false and undefined clauses after the
    valuation to UNDEFINED of the variable  $i^*$ */;
    if  $value(i) \neq UNDEFINED$  then
       $maxT = \text{number of true clauses with } S - Cl_{T \rightarrow F}[i \leftarrow U] - Cl_{T \rightarrow U}[i \leftarrow U]$ ;
       $maxF = \text{number of false clauses with } S + Cl_{T \rightarrow F}[i \leftarrow U] - Cl_{F \rightarrow U}[i \leftarrow U]$ ;
       $maxU = \text{number of undefined clauses with } S + Cl_{T \rightarrow U}[i \leftarrow U] + Cl_{F \rightarrow U}[i \leftarrow U]$ ;
    end
    Add the triplet  $(maxT, maxF, maxU)$  to the set  $E$ ;
  end
  Return  $E$ ;
end

```

**Algorithm 4:** *Analyze* function which computes the impact of each move

Inst.	Observations	Optimistic approach		Pessimistic approach	
		<i>choose1</i>	<i>choose2</i>	<i>choose1</i>	<i>choose2</i>
500 var.	(T,U,F) after TTS	(2139,1,10)	(2130,8,12)	(2145,0,5)	(2093,1,56)
	nb. var. U	20	16	18	41
	(U,UU,UUU)	(188,2,0)	(167,7,0)	(177,4,0)	(419,26,1)
	(T,F) after B&B	(2140,10)	(2135,15)	<b>(2145,5)</b>	(2129,21)
1000 var.	(T,U,F) after TTS	(4227,5,18)	(4221,10,19)	(4233,0,17)	(4146,1,103)
	nb. var. U	33	45	40	73
	(U,UU,UUU)	(356,5,0)	(462,19,0)	(394,16,0)	(804,39,1)
	(T,F) after B&B	(4232,18)	(4228,22)	<b>(4234,16)</b>	(4205,45)
2000 var.	(T,U,F) after TTS	(8455,1,44)	(8435,24,41)	(8465,0,35)	(8306,0,194)
	nb. var. U	69	99	78	130
	(U,UU,UUU)	(662,16,0)	(983,26,0)	(774,20,0)	(1390,77,0)
	(T,F) after B&B	(8456,44)	(8449,51)	<b>(8465,35)</b>	(8397,103)

**Table 1.** Comparisons among the different approaches and the different selection functions.

this combination of pessimistic approach and selection function *choose1* seems to be particularly suitable.

At this step, we have to remark that, on the one hand, B&B acts as an intensification process in the tabu search by exploiting exhaustively a subset of assignments in order to extract the best one from a MAX-SAT point of view. On the other hand, TTS can be viewed as a heuristics to point out interesting nodes in the search tree usually associated to the B&B algorithm, restricting thus the search to subparts of this whole tree.

This close interaction between the two methods leads us to propose an homogeneous hybrid algorithm in which each resolution stage acts at the same level in a complementary process.

## 6 TTS-BB: a Hybrid Algorithm

The algorithms [1, 3] based on a B&B method obtain good results on instances with few variables but, as the number of variables grows up, the performances of these algorithms decrease dramatically. The interest of the hybridization between TTS with the pessimistic approach and B&B relies on the fact that TTS selects subsets of the search space and B&B explores them exhaustively. In our algorithm, this whole process is repeated until a solution is found (0 false clause for SAT instances) or until a maximum number of visited assignments is reached. To avoid to be stuck in an already studied space, all the undefined variables are added to the tabu list. To control the allowed search power of each method, it is necessary to impose that at least a certain number  $s$  of steps (empirically fixed to 5000) are performed by TTS process between two calls to B&B. On the other hand, B&B is executed only if the number of unassigned variables is not greater than a bound  $b$  (empirically fixed to 100). The general process is sketched in

algorithm 5.

- 1) TTS executes at least  $s$  steps (except if a solution is found) until a partial assignment with at maximum  $b$  undefined variables is found.
- 2) B&B returns the best complete assignment contained in the partial assignment obtained in the step 1).
- 3) The undefined variables before the B&B process are added to the tabu list.
- 4) Return to 1).

**Algorithm 5:** General process of TTS-BB

## 7 Experimental Results

In this section, we compare the performances of TTS-BB with a classic TS algorithm. No comparison is done with B&B alone since the studied instances have too many variables. The benchmarks used are well-known instances. However, our aim is not to improve the state of the art results but to show that a hybridization of standard exact and approximate methods using the three truth values framework with our pessimistic approach and choose function provides competitive results.

### 7.1 Experimental Conditions

Due to the approximate and non-deterministic nature of TS, each algorithm runs 20 times on each benchmark. The maximum number of allowed local search steps is  $10^6$  and the value  $s$  is set to  $5 \times 10^3$ . For our hybridization, we consider that a backtrack has the same cost as a local search step. The length of the tabu list is fixed to 10% of the number of variables of the studied problem.

### 7.2 Benchmarks and Evaluation Criteria

All studied instances have been used for SAT competitions [17, 18] and contain satisfiable and unsatisfiable instances. To compare the two algorithms, four criteria are used: the average number of false clauses (avg.) and its standard deviation (s.d.) , the average number of steps (including LS steps and B&B backtracks) to obtain the best result (steps) and the percentage of improvement (Imp.) of TTS-BB with respect to the classic TS.

Benchmarks			TS			TTS-BB			Imp.
instances	var	cls	false clauses		steps	false clauses		steps	%
			avg.	s.d.		avg.	s.d.		
color-10-3	300	6475	2.60	0.50	219359	2.45	0.76	256962	+5.8
color-15-4	900	45675	5.30	0.52	263991	5.20	0.57	341175	+1.9
f1000	1000	4250	5.58	2.35	365002	3.58	0.79	497987	+35.8
f2000	2000	8500	14.58	4.17	684653	8.67	2.01	543274	+40.5
Mat25.shuffled	588	1968	21.83	14.80	43198	5.83	0.57	371227	+73.3
Mat26.shuffled	744	2464	8.00	0.00	170516	6.83	0.39	513994	+14.6
par32-5-c	1339	5350	14.70	11.38	303836	11.70	6.66	219720	+20.4
par32-5	3176	10325	23.70	30.04	361385	10.20	1.32	473116	+57.0
ssa7552-038	1501	3575	10.00	13.06	407185	8.55	1.88	553387	+14.5
term1mul.miter	3504	22229	154.92	10.47	190158	63.67	4.38	444574	+58.9

**Table 2.** Comparisons between classic TS and TTS-BB

### 7.3 Results

Table 2 shows that the TTS-BB algorithm improves the number of false clauses provided by the classic TS algorithm. This improvement can be weak for some instances (+1.9% for `color-15-4`) but really more significant for other instances (+73.3% for `Mat25.shuffled`). All the TS results are improved by TTS-BB. Moreover, the small standard deviations of the results show that the hybridization provides reliability to the algorithm. The greater number of steps performed by TTS-BB compared with TS is related to the quality of the solution found. TS stops very quickly its improvement while TTS-BB continues its search, generating more steps. The good performances of TTS are due to the new tri-valued framework which allows the algorithm to benefit from a unified combination of the two resolution techniques.

## 8 Conclusion

We have presented a resolution framework to improve hybridization between exact and approximate methods for the MAX-SAT problem. To obtain a suitable resolution framework we have introduced a third truth value *undefined* which allows the TS to explore the search space using both complete and partial assignments. With this new value, exact and approximate methods are able to share the same search space representation.

Results show that, without any specific improvement of the basic methods, the hybridization is very competitive w.r.t. a classic TS. This new resolution framework being easily flexible, other hybridizations can be created using the best exact and approximate solvers. Our future works will consist in integrating new hybridizations within this framework.

## References

1. Teresa Alsinet, Felip Many, and Jordi Planes. Improved branch and bound algorithms for MAX-SAT. In *Proc of the 6th International Conference on the Theory and Applications of Satisfiability Testing. SAT2003*, pages 408–415, 2003.
2. Belaid Benhamou and Lakhdar Sais. Theoretical study of symmetries in propositional calculus and applications. In *CADE'92*, pages 281–294, 1992.
3. Brian Borchers and Judith Furman. A two-phase exact algorithm for MAX-SAT and weighted MAX-SAT problems. *Journal of Combinatorial Optimization*, 2:299–306, 1999.
4. Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, Jul 1962.
5. 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.
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. Fred Glover and Manuel Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.
8. Jens Gottlieb, Elena Marchiori, and Claudio Rossi. Evolutionary algorithms for the satisfiability problem. *Evolutionary Computation*, 10(1):35–50, 2002.
9. Djamal Habet, Chu Min Li, Laure Devendeville, and Michel Vasquez. A Hybrid Approach for SAT. In *International Conference on Constraint Programming (CP), LNCS*, 2002.
10. 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, University of Essex, England, UK, 14-16 April 2003.
11. Wengi Huang and Renchao Jin. Solar, a quasi physical algorithm for sat. *Science in China (Series E)*, 2(27):179–186, 1997.
12. 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, 1996.
13. Chu Min Li. Integrating equivalency reasoning into davis-putnam procedure. In *Proc. of the AAAI'00*, pages 291–296, 2000.
14. Chu Min Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proc. of the IJCAI'97*, pages 366–371, 1997.
15. Bertrand Mazure, Lakhdar Sais, and Eric Grégoire. Tabu search for SAT. In *Proc. of the 14th National Conference on Artificial Intelligence and 9th Innovative Applications of Artificial Intelligence Conference (AAAI-97/IAAI-97)*, pages 281–285, Providence, Rhode Island, 1997.
16. Bertrand Mazure, Lakhdar Sais, and Eric Grégoire. Boosting complete techniques thanks to local search methods. *Annals of Mathematics and Artificial Intelligence*, 22:319–331, 1998.
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. Laurent Simon, Daniel Le Berre, and Edward A. Hirsch. The SAT2003 competition. Technical report, Sixth International Conference on the Theory and Applications of Satisfiability Testing, May 2003.

19. William M. Spears. 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, 1996.
20. 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.
21. Hantao Zhang, Haiou Shen, and Felip Many. Exact algorithms for MAX-SAT. In Ingo Dahn and Laurent Vigneron, editors, *Electronic Notes in Theoretical Computer Science*, volume 86. Elsevier, 2003.