A Hybrid Approach for SAT

Djamal Habet, Chu Min Li, Laure Brisoux Devendeville, Michel Vasquez

A Hybrid Approach for SAT*

Djamal Habet[1], Chu Min Li[2], Laure Devendeville[2], and Michel Vasquez[1]

[1] Centre LGI2P, école des Mines d'Alès, Site EERIE
Parc Scientifique Georges Besse, 30035, Cedex 01, Nîmes, France
{Djamal.Habet,Michel.Vasquez}@ema.fr
[2] LaRIA, Université de Picardie Jules Verne
5, Rue du Moulin Neuf, 80000, Amiens, France
{cli,devendev}@laria.u-picardie.fr

Abstract. Exploiting variable dependencies has been shown very useful in local search algorithms for SAT. In this paper, we extend the use of such dependencies by hybridizing a local search algorithm, Walksat, and the DPLL procedure, Satz. At each node reached in the DPLL search tree to a fixed depth, we construct the literal implication graph. Its strongly connected components are viewed as equivalency classes. Each one is substituted by a unique representative literal to reduce the constructed graph and the input formula. Finally, the implication dependencies are closed under transitivity. The resulted implications and equivalencies are exploited by Walksat at each node of the DPLL tree. Our approach is motivated by the power of the branching rule used in Satz that may provide a valid path to a solution, and generate more implications at deep nodes. Experimental results confirm the efficiency of our approach.

1 Introduction

Consider a propositional formula $\mathcal{F}$ in Conjunctive Normal Form (CNF) on a set of boolean variables $\{x_1,\ x_2, \ldots,\ x_n\}$, the satisfiability problem (SAT) consists in testing wether all clauses in $\mathcal{F}$ can be satisfied by some consistent assignment of truth values to variables.

SAT is the first known [3] and one of the most well-studied NP-complete problem. It has many applications like graph coloring, circuit designing or planning, since such problems can be encoded into CNF formulas in a natural way.

Stochastic Local Search (SLS) approaches for SAT became prominent, when independently Selman, Levesque, and Mitchell [25] as well as Gu [12] introduced algorithms based on stochastic local hill-climbing. They are considered as the most powerful incomplete methods for solving large and hard SAT instances. However, such algorithms can get stuck in the local minima of the search space, and do not integrate the structural relations between variables in their resolution. On the other hand, the complete methods, based on the Davis-Putnam-Logemann-Loveland procedure (DPLL) [5], depend on the choice of the variable to branch on. One of the best recent implementations of DPLL procedure,

* This work is partially supported by French CNRS under grant number SUB/2001/0111/DR16

Satz [19,20], uses a branching heuristic based on examining the amount of unit propagations that reduces the largest clause's number in the input SAT formula.

Improving stochastic local search on structured problems by efficiently handling variable dependencies is one of the ten challenges proposed by Selman, Kautz, and McAllester [24]. In this aim, combining systematic and stochastic search was suggested. Ginsberg and McAllester [10] have combined GSAT [25], a local search algorithm for SAT, and a dynamic backtracking [9]. The resulted algorithm allows substantial freedom of a movement in the search space but enough information is retained to ensure systematicity. Jussien and Lhomme [14] proposed a hybrid approach, for the Constraint Satisfaction Problem (CSP), performing an overall local search, based on the tabu mechanism, and using a systematic search, by filtering techniques, either to select a candidate neighbor or to prune the search space. Mazure, Saïs, and Grégoire [21] use a local search to implement the variable ordering heuristic for a systematic search. By the unit propagation process, Devendeville, Saïs, and Grégoire [2] extract variable implications, that are integrated to the tabu search mechanism of TSAT. Hirsch and Kojevnikov [13] developed a SAT solver, UnitWalk, by combining a local search and unit clause elimination.

In this paper, we extend the use of variable dependencies to a full construction of implications and equivalencies between literals (a literal is either a variable or its negation). This is performed by combining two main algorithms, Walksat [22] and Satz [19,20]. At each node of the DPLL search tree, we first construct an implication graph which is ensured consistent by propagating every instantiation. Secondly, we reduce the implication graph to its collapse strongly connected components, where each component is an equivalency class represented by an unique literal. Thirdly, because of the transitivity property of the implication, we generate its transitive closure. At the end, considering these implications, we apply Walksat to the reduced formula where a tabu list is added to forbid any cycling. This process terminates if either a solution is found or a maximal fixed depth of the Satz tree is reached. These treatments allow the DPLL procedure to support the local search mechanism, strengthened by the variable dependencies.

This paper is organized as follows: sections 2 and 3 respectively present the two combined algorithms Walksat and Satz. Section 4 describes the different steps of our approach. Section 6 discusses the experimental results presented in section 5, and section 7 concludes.

2 Walksat

Originally introduced in [23], Walksat performs a greedy local search for a satisfying assignment of a set of propositional clauses in SAT format. The procedure starts with a randomly generated truth assignment. It then changes (flips) the assignment of a variable chosen under the heuristic described in algorithm 2 below. Flips are repeated until either a satisfying assignment is found or a preset maximum number of flips, *Max-Flips*, is reached. This process is repeated up to a maximum number of *Max-Tries* times.

Algorithm 1: Walksat

```
Input: SAT-formula F, Max-Tries,Max-Flips
Output: A satisfying truth assignment T of F, if found
begin
    for try=1 to Max-Tries do
        T←randomly generated truth assignment;
        for flip=1 to Max-Flips do
            if T satisfies F then return T;
            c←randomly selected clause violated under T;
            v←Heuristic(F,c);
            T←T with v flipped;
    return "Solution not found";
end;
```

Algorithm 2: Heuristic

```
Input: SAT-formula F, violated clause c
Output: Selected variable to flip, v
begin
    for each variable u appearing in c do
        Calculate score(u) equal to the violated clause number in F if u is flipped;
    if there are variables with null score then
        v←randomly select one (zero-damage-flip);
    else
        switch a probability value (noise setting) do
            case wp : v←variable with minimal score (minimal-damage-flip);
            otherwise v ←randomly select a variable from c (random walk);
    return v ;
end;
```

Compared to its predecessors, like GSAT, Walksat differs in one important aspect. In fact, while GSAT architecture is characterized by a static neighborhood relation between assignments with Hamming distance one, Walksat's one is based on a dynamically determined subset of the GSAT neighborhood relation. Effectively, the experimental results show that Walksat outperforms the existing SLS algorithms proposed before, and it is proved Probabilistically Approximately Complete (PAC property) with a noise setting $wp > 0$ [13].

3 Satz

Despite its simplicity and seniority, the Davis-Putnam-Logemann-Loveland procedure (DPLL) remains one of the best complete procedures for SAT. It essentially constructs a binary search tree and its nodes are results of recur-

sive calls. While a solution is not found, all leaves represent a dead-end where a contradiction (empty clause) is found.

Algorithm 3: DPLL

Input: SAT-formula $\mathcal{F}$
Output: SAT-decision
begin
  if $\mathcal{F}$ *is empty* **then** return "Satisfiable";
  $\mathcal{F} \leftarrow$ UnitPropagation(F);
  if $\mathcal{F}$ *contains an empty clause* **then** return "Unsatisfiable";
  Select a variable x in $\mathcal{F}$ according to a heuristic H **(Branching rule)**;
  if *DPLL($\mathcal{F} \cup \{x\}$) return "Satisfiable"* **then** return "Satisfiable";
  else return the result of calling DPLL($\mathcal{F} \cup \{\bar{x}\}$);
end;

Algorithm 4: UnitPropagation

Input: SAT-formula $\mathcal{F}$
Output: $\mathcal{F}$ simplified by unit propagations
begin
  while *there is no empty clause and a unit clause l exists in $\mathcal{F}$* **do**
    Assign *true* to l and simplify $\mathcal{F}$;
  return $\mathcal{F}$;
end;

DPLL procedure performance is closely related to the selection of the branching variable. In fact, this selection affects the search tree size, and consequently the required time to solve $\mathcal{F}$. A popular and a cheap branching heuristic is the MOM[1] heuristic [8], which picks the variable that occurs the most often in the minimal size clauses. However, work realized in Posit [7,8], Tableau [4], and Satx [19] have suggested to integrate unit propagations to the heuristic H. It results in an UP[2] heuristic which examines the variable x by respectively adding the unit clauses x and $\bar{x}$ into $\mathcal{F}$, and independently making two unit propagations. The real effect of those propagations is then used to weigh x. However, since examining variables by two unit propagations is time consuming, it is necessary to reduce the number of variables examined by the UP heuristic. Taking into account the number of binary occurrences of variables, Satz gets the best restrictions on the number of examined variables. In that way, combining the MOM and UP heuristics, Satz reduces the size of the search tree by detecting failed literals as early as possible.

[1] Maximum Occurrences of clauses of Minimum size
[2] Unit Propagation

4 Hybrid Approach

4.1 Variable Dependencies

SAT encodings of structured problems, such as planning and diagnosis, often contain large numbers of variables whose values are constrained to be a simple boolean function of other variables. These variables are then dependent. Variables whose values cannot be easily determined to be a simple function of other variables are independent. For a given SAT problem, there may be many different ways to classify the variables as dependent. In this work, we use the *dependency definition* given in [17] as follows:

Definition 1. *Let Σ be a set of clauses, V the related variables, C finite conjunctions of literals such that $V_C \subseteq V$, and a variable y such that $\Sigma \models (y \Rightarrow C)$. Then we say that the variables of V_C depend on the variable y. Roughly speaking, if y is instantiated then all variables of V_C are instantiated too.*

Example 1. Consider the set of clauses $\Sigma = \{\neg l \vee a,\ \neg l \vee \neg b,\ b \vee \neg c\}$. If l is fixed then the literals a, $\neg b$, $\neg c$ are fixed too. So we have the dependencies $l \rightarrow \{a, \neg b, \neg c\}$. Such dependencies are *implications*.

The implications are naturally constructed by unit propagations, performed by Satz when looking for the variable to branch on, and are represented by a directed graph, where nodes are literals and edges relate two dependent literals. This construction is done in linear time.

Favorably, two literals mutually implied are *equivalent*. An equivalency is a stronger dependency than a simple implication. A set of equivalent literals, constituting a class, is a strongly connected component of the implication graph, where a representative literal, chosen randomly, substitutes the other element of its class. Consequently, the input formula is reduced and the implication graph is also reduced and becomes acyclic.

4.2 Dependencies Consistency

In the preprocessing of a SAT formula or when a branching occurs, Satz fixes some variables and their states become *passive*. To maintain the implications coherent, this state must be propagated through the implication graph for the satisfied literals, and through its transposed graph for the falsified literals. Those propagations are performed by a Depth-First-Search procedure in a linear time, which is also used to construct the equivalency classes restricted to the *active* variables. On the other hand, the implication relation is transitive. Once again, by the Depth-First-Search procedure, the implication closure is constructed from the reduced implication graph. At this stage, the implications are fully enumerated and are consistent.

4.3 Proposed Algorithm: *WalkSatz*

After applying the enumerated processes to each sub-formula obtained in a current node of the Satz tree, Walksat is then applied by considering the equivalency classes rather than original variables. The class to flip is selected as described in the section 2. The implications are to be integrated advantageously in the Walksat resolution. In fact, three cases are distinguished when Walksat chooses a variable to flip: a zero-damage-flip, a minimal-damage-flip, and a random walk. It is then possible to merge the implication constraint in one, two, or all the three cases, i.e., in any case, if a class is flipped then the implied ones are flipped too. We have tested the eight possible combinations under a large variety of problems, and the best integration was observed in the case of a minimal-damage-flip. Such result can be interpreted by:

1. The zero-damage-flip is the best case, but flipping the implied classes may increase the violated clause number.
2. The random walk is incompatible with the deterministic behavior of implications, and integrating the implication relation to all levels, without any improved mechanism, the search may be trapped easily in the local minima.

We should remark that a large reduction of an instance size may cause, in the undesirable cases, a cancellation of successive actions (flips) applied to a same class. To avoid this, a tabu list [11] of a tenure fixed empirically to 1, is used as follows: each flipped class, as well as the related ones by the implication relation (if flipped), are forbidden to any change during one iteration. The full steps of the hybrid approach are resumed in the developed algorithm, **WalkSatz**, which is incomplete because of the restriction on the depth of the Satz tree.

Algorithm 5: WalkSatz

Input: SAT-formula $\mathcal{F}$
Output: A satisfying truth assignment T of $\mathcal{F}$, if found
begin
  for *each node reached by Satz, down to a fixed depth* **do**
    if $\mathcal{F}$ *is empty* **then** return "Solution T found, by Satz";
    Construct for $\mathcal{F}$, the implication graph I and its transposed I^t;
    Propagate variables state under I and I^t;
    Construct the equivalency classes and reduce $\mathcal{F}$, I, and I^t;
    Construct the implication closure for the reduced graphs I and I^t;
    Apply Walksat to $\mathcal{F}$ reduced to its equivalency classes, taking into account the reduced graphs I and I^t when minimizing the total number of violated clauses (minimal-damage-flip);
    if *the last step returns "Solution T found"* **then**
      return "Solution T, by Walksat";
  return "Solution not found";
end;

5 Experimental Results

The hybrid approach is the result of combining a systematic method and a local search one. Consequently, it is not so evident to give comparative criteria between such families. However, in order to evaluate the performances of WalkSatz, we compare it with Walksat and Satz on a broad range of benchmarks[3]. WalkSatz is coded in Linux/C++ and compiled with g++ compiler. Comparative parameters used in our experiments[4] are:

	WalkSatz	Walksat
Max-Tries	1	100
Max-Flips	10^5	15×10^5
Number of runs	100	1
Noise setting	0.5	0.5

Despite the importance of the noise parameter, no optimization is made on. However, its value is fixed to an identical value, 0.5, to both algorithms. The depth of the Satz tree is limited to 3, the root being at level 0, at most fifteen nodes are then developed, and the total number of the authorized flips is identical for WalkSatz and Walksat. The columns T%, # flips, and sec. correspond respectively to the success rate, average flips, and average run time in second. # flips and sec. are calculated for the successful executions, and the running time for Satz is limited to 7200 seconds. All the used instances are satisfiable.

5.1 Latin Square

Given a set S, a Latin square indexed by S is an $|S| \times |S|$ array such that each row and each column of the array are a permutation of the elements in S. $|S|$ is called the order of the Latin square. These instances have been contributed by H. Zhang [27].

Table 1. Experimental results for the "qg" instances

Latin square			WalkSatz			Walksat			Satz
Instance	# Vars.	# Cls.	T%	# flips.	sec.	T%	# flips.	sec.	sec.
qg1-08	512	148957	6	1307271	1499.446	0	-	-	125.220
qg2-07	343	68083	100	39719	0.901	25	693660	694.622	24.340
qg3-08	512	148957	100	238622	1.505	9	628604	148.794	0.210
qg4-09	512	10469	83	701706	9.169	1	328792	3226.650	0.750
qg5-11	1331	64054	8	277167	370.769	0	-	-	4.830
qg6-09	729	21844	100	368	0.664	0	-	-	0.830
qg7-09	729	22060	100	81	0.824	0	-	-	0.970

[3] http://www.satlib.org

[4] All experiments are on a Duron 800 Mhz machine with 256 MB of RAM.

Table 2. Experimental results for the "aim" instances

AIM			WalkSatz			Walksat			Satz
Instance	# Vars.	# Cls.	T%	# flips.	sec.	T%	# flips.	sec.	sec.
aim-50-1-6-yes1-1	50	300	100	1	0.001	6	920684	44.422	0.020
aim-50-1-6-yes1-2	50	300	100	1	< 0.001	1	64016	148.600	0.030
aim-50-1-6-yes1-3	50	300	100	1	< 0.001	34	685077	6.378	0.010
aim-50-1-6-yes1-4	50	300	100	1	< 0.001	0	-	-	0.010
aim-50-2-0-yes1-1	50	100	100	1	< 0.001	2	702	83.125	0.010
aim-50-2-0-yes1-2	50	100	100	1	< 0.001	97	22170	0.094	0.020
aim-50-2-0-yes1-3	50	100	100	1	< 0.001	100	151431	0.287	0.020
aim-50-2-0-yes1-4	50	100	100	1	< 0.001	100	137124	0.237	0.020
aim-100-1-6-yes1-1	100	160	100	1	< 0.001	0	-	-	0.020
aim-100-1-6-yes1-2	100	160	100	1	< 0.001	0	-	-	0.030
aim-100-1-6-yes1-3	100	160	100	1	< 0.001	0	-	-	0.030
aim-100-1-6-yes1-4	100	160	100	1	< 0.001	0	-	-	0.030
aim-100-2-0-yes1-1	100	200	100	1	< 0.001	0	-	-	0.020
aim-100-2-0-yes1-2	100	200	100	1	0.001	0	-	-	0.030
aim-100-2-0-yes1-3	100	200	100	1	< 0.001	0	-	-	0.030
aim-100-2-0-yes1-4	100	200	100	1	< 0.001	0	-	-	0.020
aim-200-1-6-yes1-1	200	360	100	1	0.003	0	-	-	0.050
aim-200-1-6-yes1-2	200	360	100	1	0.003	0	-	-	0.090
aim-200-1-6-yes1-3	200	360	100	1	< 0.001	0	-	-	0.050
aim-200-1-6-yes1-4	200	360	100	1	0.001	0	-	-	0.050
aim-200-2-0-yes1-1	200	400	100	1	0.001	0	-	-	0.050
aim-200-2-0-yes1-2	200	400	100	1	0.001	0	-	-	0.040
aim-200-2-0-yes1-3	200	400	100	1	0.001	0	-	-	0.100
aim-200-2-0-yes1-4	200	400	100	1	0.001	0	-	-	0.070

5.2 DIMACS Benchmarks

"aim" instances: proposed by Iwama and *al.* [1], the instances are all generated with a particular random 3-SAT instance generator.

"ssa" instances: contributed by A.V. Gelder and *al.*, the instances correspond to single-stuck-at-faults problem in circuit analysis. The used instances are selected formulas from those generated by *N*emesis [6,18], which is a test-pattern generation program.

Table 3. Experimental results for the "ssa" instances

SSA			WalkSatz			Walksat			Satz
Instance	# Vars.	# Cls.	T%	# flips.	sec.	T%	# flips.	sec.	sec.
ssa7552-160	1391	3126	100	385	2.270	100	30875	0.061	0.080
ssa7552-159	1363	3032	100	344	2.265	100	22006	0.433	0.080
ssa7552-158	1363	3034	100	255	2.218	100	26738	0.052	0.070
ssa7552-038	1501	3575	100	991	2.581	100	80673	0.167	0.120

Table 4. Experimental results for the "par8" instances

PAR			WalkSatz			Walksat			Satz
Instance	# Vars.	# Cls.	T%	# flips.	sec.	T%	# flips.	sec.	sec.
par8-1-c	64	254	100	12062	0.024	100	10628	0.025	0.02
par8-1	350	1149	100	68175	0.180	4	977367	79.366	0.01
par8-2-c	68	270	100	1	0.003	100	15504	0.036	0.02
par8-2	350	1157	100	120942	0.356	2	483886	160.791	0.02
par8-3-c	75	298	100	35293	0.069	100	37213	0.089	0.02
par8-3	350	1171	100	96110	0.272	3	538312	107.811	0.02
par8-4-c	67	266	100	1	0.000	100	41423	0.098	0.02
par8-4	350	1155	100	22966	0.681	0	-	-	0.02
par8-5-c	75	298	100	12843	0.027	100	34534	0.084	0.02
par8-5	350	1171	100	77604	0.284	1	304169	330.066	0.03

"par8" instances: contributed by J. Crawford and suggested by M. Kearns, these instances are propositional versions of the parity learning problem.

5.3 Superscalar Processor Verification

Defined by M.N. Velev [26], these instances result from the verification of exceptions, multicycle functional units, and branching prediction in superscalar microprocessors, DLX.

5.4 Beijing-Challenge Benchmarks

This set comprises the instances which have been proposed for the International Competition on SAT Testing in Beijing, 1996, including planning and scheduling problems [15,16].

Table 5. Experimental results for the "dlx_cc" instances

DLX2_CC			WalkSatz			Walksat			Satz
Instance	# Vars.	# Cls.	T%	# flips.	sec.	T%	# flips.	sec.	sec.
dlx2_cc_bug01	1515	12808	100	989391	49.607	79	603523	24.857	> 7200
dlx2_cc_bug02	1515	12808	100	858251	43.818	80	517174	21.958	> 7200
dlx2_cc_bug03	1515	12808	100	1013661	51.013	71	723533	33.193	> 7200
dlx2_cc_bug14	1516	12811	100	458607	26.524	98	381390	10.122	> 7200
dlx2_cc_bug16	1516	12811	100	461816	25.555	92	417319	13.432	> 7200
dlx2_cc_bug33	1516	12798	100	636107	23.721	99	265186	6.820	> 7200
dlx2_cc_bug34	1516	12718	100	965240	43.048	98	369863	9.282	> 7200
dlx2_cc_bug38	1515	12783	100	652728	35.267	63	516764	1.404	> 7200
dlx2_cc_bug39	1482	12111	93	1141143	49.067	66	640518	33.861	> 7200
dlx2_cc_bug40	1520	12811	100	629230	33.210	98	358702	9.688	> 7200

Table 6. Experimental results for the Beijing instances

Beijing			WalkSatz			Walksat			Satz
Instance	#Vars.	# Cls.	T%	# flips.	sec.	T%	# flips.	sec.	sec.
3blocks	370	13732	100	26768	0.222	100	49960	0.626	0.060
4blocks	900	59285	34	519608	102.429	0	-	-	0.060
4blocksb	540	34199	100	264532	3.655	19	694517	210.913	0.650
e0ddr2-10-by-5-1	19500	108887	100	246496	1067.484	0	-	-	137.820
e0ddr2-10-by-5-4	19500	104527	98	463371	3116.833	18	1202849	132.465	185.900
enddr2-10-by-5-1	20700	111567	97	292415	2352.466	0	-	-	112.960
enddr2-10-by-5-8	21000	113729	100	261973	1463.593	2	1329010	1448.225	105.230
ewddr2-10-by-5-1	21800	118607	97	252841	1119.552	0	-	-	401.460
ewddr2-10-by-5-8	22500	123329	100	205542	846.741	1	1456023	3136.267	120.640

Table 7. Experimental results for the Kautz & Selman planning problem

Planning			WalkSatz			Walksat			Satz
Instance	# Vars.	# Cls.	T%	# flips.	sec.	T%	# flips.	sec.	sec.
logistics.a	828	6718	100	21508	0.088	100	133275	0.469	2.750
logistics.b	843	7301	100	36783	0.142	100	249287	0.887	0.090
logistics.c	1141	10719	100	77653	0.334	87	490317	2.896	0.420
logistics.d	4713	21991	100	286041	155.530	74	624500	6.990	507.490
bw_large.a	459	4675	100	5146	0.199	100	14359	0.057	0.070
bw_large.b	1087	13772	100	96838	1.899	86	564590	6.249	0.240
bw_large.c	3016	50157	100	198672	22.738	0	-	-	2.740

5.5 Kautz & Selman Planning Problems

Proposed by H. Kautz and B. Selman [15,16], these instances correspond to the planning logistic problem.

6 Experimental Result Discussion

Globally and when Compared to Walksat, WalkSatz largely reduces the required flips number to get a solution, and has a good behavior when solving hard instances. These statements may be justified as follows:

1. The equivalency classes reduce the problem size and eliminate the cyclic implications. On the other side, the implication transitive closure permit to enumerate all the dependent literals, and so, exploit fully their advantage.
2. The implication relation can be viewed as a tabu which restricts the visited neighborhood by the local search. Furthermore, checking the implication consistency allows to verify implicit constraints, then reduce the number of flips required to reach a solution.
3. The branching rule used by Satz gives two sub-formulas completely disjoint. So, the space search explored by Walksat is diversified.

Excepting "qg1-08", "qg5-11", and "4blocks" instances, the high rate success of WalkSatz shows the robustness of this hybrid algorithm. Furthermore, whenever Satz works well, WalkSatz takes advantage in robustness, and when Satz works less well, for example for the "dlx2_cc" instances, there is a small improvement in robustness at the expense of computation time. On random SAT problems containing few variable dependencies, Walksatz and Walksat essentially have the same behavior. The experimental results are not presented here.

As said in the section 5, it is less evident to make an adjusted comparison, and the used parameters are retained to compare the behavior of WalkSatz and WalkSat with a same number of authorized flips. However, other parameters are to underline: in the above experiments, if WalkSatz fails in a given node of the Satz tree, no restart is done on (only one try is authorized), but other experiments show that WalkSatz works better with more tries: a restart is an useful aspect in a local search. The depth of the Satz tree is also prominent, and its optimized value can be found for each problem class. However, a tarde-off should be respected: it is look not so interesting to develop many nodes, in the Satz tree, to let the local search reaching a solution. At the end, the tabu tenure and list, used by WalkSatz, are to be well handled: because of the instance reduction, these tabu parameters can be a function of the number/size of the equivalency classes.

The main weakness of WalkSatz is the computation time requirements. It is mainly caused by the repeated Depth-First-Search procedures at every node of the Satz search tree. In fact, the version elaborated in this paper is not optimized. This lack is easily improved by an incremental construction of the implications and equivalencies at each branching and backtracking done by Satz. Despite this drawback, the experimental results prove the efficiency of our approach.

7 Conclusion

In this paper, we presented a hybrid approach between two efficient algorithms Walksat and Satz. Our work is motivated by the performances of such algorithms. In fact, by its powerful branching rule, Satz provides an optimal search tree, and by its variable selection heuristic, Walksat outperforms the precedent SLS algorithms. On the other hand, SAT formulas contain implicit variable dependencies that are so useful in a SAT resolution, especially in a local search process. To well exploit those elements, WalkSatz was elaborated. It is the result of numerous experiments in the aim to get a good hybridization that increases the contribution of implication and equivalency relationships in the resolution mechanism. The obtained results support our approach.

As perspective works, more powerful variants of Walksat, such as Novelty and R-Novelty [22], are also indicated for such approach. The variable dependencies, as well as tabu, noise setting, and other WalkSatz parameters, are to be further studied to increase their effects. Finally, to improve the computation time, these dependencies must be constructed *incrementally*.

References

1. Y. Asahiro, K. Iwama, and E. Miyano. Random Generation of Test Instances with Controlled Attributes. In *D. S.Johnson and M. A.Trick, editors, Cliques, Coloring, and Satisfiability; The Second DIMACS Implementation Challenge*, volume 26, pages 377–394, 1996. 179
2. L. Brisoux Devendeville, L. Saïs, and E. Grégoire. Recherche locale: vers une exploitation des propriétés structurelles. In *proceedings of JNPC'2000*, pages 243–244, Marseille, France, 2000. 173
3. S. Cook. The Complexity of Theorem Proving Procedures. In *Proceeding oh the Third Annual ACM Symp. on Theory of Computing*, pages 151–158, 1971. 172
4. J. M. Crawford and L. D. Auton. Experimental Results on the Crosover Point in Random 3-SAT. *Artificial Intelligence Journal*, 81(1-2):31–57, 1996. 175
5. M. Davis, G. Logemann, and D. Loveland. A Machine Program for Theorem Proving. In *Communication of ACM Journal*, volume 5(7), pages 394–397, July 1962. 172
6. F. J. Ferguson and T. Larrabee. Test Pattern Generation for Realistic Bridging Faults in CMOS ICS. In *Proceedings of the International Testing Conference*, pages 492–499, 1991. 179
7. J. W. Freeman. *Improvements to Propostional Satisfiability Search Algorithms*. PhD thesis, Departement of Computer and Information Science, University of Pennsylvania, Philadelphia, PA, 1995. 175
8. J. W. Freeman. Hard Random 3-SAT Problems and the Davis-Putnam Procedure. In *Artificial Intelligence Journal*, pages 81:183–198, 1996. 175
9. M. L. Ginsberg. Dynamic backtracking. *Journal of Artificial Intelligence Research*, 1:25–46, 1993. 173
10. M. L. Ginsberg and D. A. McAllester. GSAT and Dynamic Backtracking. In P. Torasso, J. Doyle, and E. Sandewall, editors, *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning KR'94*, pages 226–237. Morgan Kaufmann, 1994. 173
11. F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997. 177
12. J. Gu. Efficient Local Search for Very Large-Scale Satisfiability problems. In *ACM SIGART Bulletin*, pages 3(1):8–12, 1992. 172
13. H. H. Hoos and T. Stutzle. Local search algorithms for SAT: An empirical evaluation. *Journal of Automated Reasoning*, 24(4):421–481, 2000. 173, 174
14. N. Jussien and O. Lhomme. Local search with constraint propagation and conflict-based heuristics. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI'2000)*, pages 169–174, Austin, TX, USA, August 2000. 173
15. H. Kautz, D. McAllester, and B. Selman. Encoding Plans in Propositional Logic. In *Proceedings of the 4th International Conference on the Principle of Knowledge Representation and Reasoning, KR'96*, pages 374–384, 1996. 180, 181
16. H. Kautz and B. Selman. Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search. In Howard Shrobe and Ted Senator, editors, *Proceedings of the 13th National Conference on Artificial Intelligence and the 8th Innovative Applications of Artificial Intelligence Conference*, pages 1194–1201, Menlo Park, California, 1996. 180, 181
17. J. Lang and P. Marquis. Complexity Results for Independence and Definability in Propositional Logic. In A. G. Cohn, L. K. Schubert, and S. C. Shapiro, editors, *Proceedings of the Sixth International Conference on Principles of Knowledge Representataion and Reasoning, KR'98*, pages 356–367, 1998. 176

18. T. Larrabee. Test Pattern Generation Using Boolean Satisfiability. In *IEEE Transactions on Computer-Aided Design*, pages 11(1):6–22, 1992. 179
19. C. M. Li. Exploiting Yet More the Power of Unit Clause Propagation to solve 3-SAT Problem. In *ECAI'96 Workshop on Advances in Propositional Deduction*, pages 11–16, Budapest, Hungray, 1996. 173, 175
20. C. M. Li and Anbulagan. Heuristic Based on Unit Propagation for Satisfiability. In *Proceedings of CP'97, Springer-Verlag, LNCS 1330*, pages 342–356, Austria, 1997. 173
21. B. Mazure, L. Saïs, and E. Grégoire. Boosting Complete Techniques Thanks to Local Search. *Annals of Mathematics and Artificial Intelligence*, 22(3-4):319–331, 1998. 173
22. D. McAllester, B. Selman, and H. Kautz. Evidence for Invariants in Local Search. In *Proceedings of the 14th National Conference on Artificial Intelligence, AAAI'97*, pages 321–326, Providence, Rhode Island, 1997. MIT Press. 173, 182
23. B. Selman, H. Kautz, and B. Cohen. Noise Strategies for Improving Local Search. In MIT press, editor, *Proceedings of the 12th National Conference on Artificial Intelligence AAAI'94*, volume 1, pages 337–343, 1994. 173
24. B. Selman, H. Kautz, and D. McAllester. Ten Challenges in Propositional Reasoning and Search. In *Proceedings of IJCAI'97*, pages 50–54, Nagoya, Aichi, Japan, August 1997. 173
25. B. Selman, H. J. Levesque, and D. Mitchell. A New Method for Solving Hard Satisfiability Problems. In Paul Rosenbloom and Peter Szolovits, editors, *Proceedings of the 10th National Conference on Artificial Intelligence, AAAI'92*, pages 440–446, Menlo Park, California, 1992. 172, 173
26. M. N. Velev and R. E. Bryant. Superscalar processor verification using efficient reductions of the logic of equality with uninterpreted functions to propositional logic. In *Correct Hardware Design and Verification Methods, CHARME'99*, 1999. 180
27. H. Zhang and M. E. Stickel. Implementing the davis-putnam method. *Journal of Automated Reasoning*, 24(1):277–296, 2000. 178