UnitWalk: A new SAT solver that uses local search guided by unit clause elimination*

Edward A. Hirsch[1]** and Arist Kojevnikov[2]***

[1] Russian Academy of Sciences, Steklov Institute of Mathematics at St.Petersburg, 27 Fontanka, 191011 St.Petersburg, Russia, http://logic.pdmi.ras.ru/~hirsch.
[2] St.Petersburg State University, Department of Mathematics and Mechanics, St.Petersburg, Russia, http://logic.pdmi.ras.ru/~arist.

Abstract. In this paper we present a new randomized algorithm for SAT, i.e., the satisfiability problem for Boolean formulas in conjunctive normal form. Despite its simplicity, this algorithm performs well on many common benchmarks ranging from graph coloring problems to microprocessor verification. Our algorithm is inspired by two randomized algorithms having the best current worst-case upper bounds ([PPSZ98,PPZ97] and [Sch99,SSW01]). We combine the main ideas of these algorithms in one algorithm. The two approaches we use are local search (which is used in many SAT algorithms, e.g., in GSAT [SLM92] and WalkSAT [SKC94]) and unit clause elimination (which is rarely used in local search algorithms). In this paper we do not prove any theoretical bounds. However, we present encouraging results of first computational experiments. We also prove that our algorithm is probabilistically approximately complete (PAC).

1 Introduction

Since this extended abstract is aimed at SAT Symposium, we skip most definitions and history concerning the Boolean satisfiability problem (SAT). Concerning these topics we refer the interested reader to the preliminary full version of our paper [HK01].

Weakly exponential worst-case upper bounds. "Weakly exponential" upper bounds on the worst-case running time of SAT algorithms are bounds of the form $p(n)c^n$, where $c < 2$ is a constant, n is the number of variables in the input formula, and p is a polynomial (note that SAT can be easily solved in time $p(n)2^n$, but it is non-trivial that it can be solved faster). Currently, the best known deterministic algorithm for k-SAT (i.e., when the length of a clause is bounded by k) has the bound approaching $p(n)(2 - 2/(k+1))^n$ [DGH$^+$02]. Randomized algorithms achieve even better bounds [Sch99,PPSZ98]. If a formula has exactly one satisfying assignment, it can be found even faster: in the time $p(n)1.308^n$ for a formula in 3-CNF [PPSZ98]. Concerning other results in this reach field, we refer the readers to the survey [DHIV01].

"Heuristic" algorithms. Unfortunately, the worst-case upper bounds currently known for SAT algorithms are still too large for practical purposes. Another direction of research is the design of "heuristic" algorithms. These algorithms, though they are very hard for theoretical study, show good performance in practice both on randomly generated instances and on structured instances encoding various practical problems like planning or circuit design. Theoretical knowledge about these algorithms is very limited and mostly concentrates on exponential worst-case lower bounds [Hir00], average-case bounds [GG94,KP92], or properties

* See also [HK01] for a more detailed preliminary version.
** Supported by CRDF and by grant #1 of the 6th RAS contest-expertise of young scientists projects (1999).
*** Supported by CRDF.

related to their completeness [CGH00,Hoo98,Hoo99]. Some of these algorithms are surveyed in [GPFW97,HS00a].

In their turn, "heuristic" algorithms may be complete or incomplete. A *complete* algorithm gives the correct answer with certainty. If an *incomplete* algorithm finds a satisfying assignment, it is guaranteed to be correct; however, if it fails to find a satisfying assignment, this means that either the input formula is unsatisfiable, or it is satisfiable, but appeared "too hard" for this algorithm. Note that since we have no a priori bound on the probability of error, we cannot just repeat such an algorithm sufficiently many times reducing the probability of error to any predefined constant ($O(2^n)$ iterations would do the job for almost all known algorithms, but it is unrealistic in practice). There are several quite successful SAT solvers based on complete algorithms (e.g., rel_sat [BS97], satz [LA97], SATO [Zha93,ZS00]; see comprehensive experimental results at SAT-Ex web-site [Sim]). However, in general, solvers based on incomplete algorithms are faster.

In this paper we suggest one more incomplete heuristic algorithm for SAT; therefore, throughout this paper we will speak only about satisfiable formulas. Most incomplete "heuristic" algorithms use local search paradigm (this line of research started from experiments of Gu [Gu92] and Selman, Levesque and Mitchell [SLM92] and theoretical work by Koutsoupias and Papadimitriou [KP92,Pap91]). Such an algorithm chooses a random initial assignment and then modifies it step by step until it finds a satisfying assignment. If the random walk is long enough, it is restarted from another initial assignment. There is a number of experimentally studied SAT local search algorithms, for example, GSAT [SLM92], GWSAT (aka GSAT+w) [SKC94], HSAT [GW93], HWSAT [GW95], SDF [SS00], WalkSAT [SKC94] and various algorithms within WalkSAT framework such as WalkSAT/TABU, Novelty, R-Novelty [MSK97], Novelty+ and R-Novelty+ [Hoo99].

Some of these algorithms *need* restarts because there are initial assignments such that the probability that the random walk (if allowed to run infinitely long) hits a satisfying assignment is strictly less than one (in fact, it can be zero for most of these algorithms). Such algorithms are called *essentially incomplete* [Hoo99] (e.g., GSAT, WalkSAT/TABU, Novelty, R-Novelty [Hoo98,Hoo99]). Other algorithms are *probabilistically approximately complete* (*PAC*) [Hoo99], i.e., they succeed with probability one without restarts for *every* initial assignment (e.g., Novelty+, R-Novelty+, or GWSAT with strictly positive noise parameter [Hoo98,Hoo99]).

Unit clause elimination is a common technique in complete SAT algorithms. However, it seemed hard to use it in local search algorithms. Recently, there has been such an attempt [Pre01]. Our local search algorithm also uses unit clause elimination, though in a different way.

Procedure of Paturi, Pudlák, Saks, and Zane and our algorithm. In this paper, we suggest a new incomplete SAT algorithm UnitWalk. Similarly to the experimentally best incomplete algorithms from WalkSAT family [MSK97] and the theoretically best algorithms by Schöning [Sch99] and Schuler, Schöning and Watanabe [SSW01], our algorithm uses local search. However, the heuristic for flipping a variable is motivated by another theoretically best algorithm by Paturi, Pudlák, Saks, and Zane [PPSZ98].

The core of the algorithm in [PPSZ98] is the following procedure [PPZ97]. Take a random assignment for the input formula F and a random permutation of variables of F. Consider all variables in the order determined by this permutation. For each variable v, do the following. If the value of v is forced by a unit clause, then set the forced value to the variable; otherwise, take the value from the random assignment. In any case, make the corresponding substitution in the formula, i.e., replace F by $F[v \leftarrow \texttt{True}]$ or $F[v \leftarrow \texttt{False}]$ respectively by removing all clauses containing the literal (either v or $\neg v$) having the value `True` and removing the opposite literal from the remaining clauses.

Paturi, Pudlák, and Zane [PPZ97] show that this procedure finds a satisfying assignment for a formula in 3-CNF with probability at least $O(2^{-2n/3})$ (in other words, with high probability, only $2n/3$ values in the initial assignment are essential). Therefore, repeating this procedure $O(2^{2n/3})$ times gives a constant probability of error (for any predefined constant). This algorithm can be derandomized by enumerating $2^{2n/3}$ assignments and a polynomial number of permutations ([PPZ97], see also [DHIV01] for a simpler construction).

Paturi, Pudlák, Saks and Zane [PPSZ98] give an extension of this algorithm. The new algorithm includes a preprocessing step: it adds some resolvents to the input formula. This allows to achieve a higher probability of success in one iteration and hence the improved running time $p(n)1.363^n$ (or even $p(n)1.308^n$ if there is only one satisfying assignment).

In [PPZ97,PPSZ98], the procedure described above is used for obtaining a satisfying assignment in one iteration: if a satisfying assignment is not found, the procedure is restarted with a new random assignment and a new random permutation. In our algorithm, we use this procedure to obtain an assignment that is closer to a satisfying assignment than the initial assignment (i.e., the obtained unsatisfying assignment is not dropped, but is used instead of a random assignment for the next iteration of the procedure). In fact, we use a slightly modified procedure based on the version described in [DHIV01]. Namely, we process unit clauses as soon as they appear (irrespectively of the chosen permutation of variables). We also make a random choice if more than one unit clause appears. If there are two unit clauses of the opposite signs, we do *not* flip the corresponding variable. Finally, if no variable is flipped after considering all variables, we flip a randomly chosen variable (in fact, this is a very rare situation). See Section 2 for more details about our algorithm and its relation to other heuristic algorithms such as WalkSAT/TABU [MSK97].

We prove that our algorithm is probabilistically approximately complete. We also implemented a SAT solver based on our algorithm. This "rough" implementation (not optimized in any sense) already shows experimental results that are competitive with other SAT solvers. We provide the experimental results we obtained so far on various benchmarks.

Organization of the paper. The paper is organized as follows. Our algorithm is described in Section 2, where it is also shown that the algorithm is probabilistically approximately complete, and its relation to other local search algorithms is discussed. In Section 3 we present experimental data describing the execution of our algorithm on various benchmarks. Finally, in Section 4 we summarize our results and point directions for further research.

2 Algorithm UnitWalk

2.1 Description of the algorithm

As a typical local search algorithm, our algorithm generates an initial assignment at random and then modifies it step by step. The main difference from other local search algorithms is that during this walk our algorithm modifies also the input formula.

The random walk is divided into *periods*. During one period at least one (usually, much more) flip is made. A period starts with choosing a random permutation of variables. Then

Fig. 1. Algorithm UnitWalk

Input: A formula F in CNF containing n variables $x_1, \ldots, x_n$.
Output: A satisfying assignment for F, or "Not found".
Method:

    For $t := 1$ to MAX_TRIES(F) do
        $A :=$ random truth assignment for n variables;
        For $p := 1$ to MAX_PERIODS(F) do
            $\pi :=$ random permutation of $1..n$;
            $G := F$;
            $f := 0$;
            For $i := 1$ to n do
                While G contains a unit clause, repeat
                • Pick a unit clause $\{x_j\}$ or $\{\neg x_j\}$ from G at random;
                • If this clause is not satisfied by A, and G does not contain the opposite unit clause, then flip $A[j]$ and set $f := 1$;
                • $G := G[x_j \leftarrow A[j]]$;
                If variable $x_{\pi[i]}$ still appears in G, then $G := G[x_{\pi[i]} \leftarrow A[\pi[i]]]$.
            If G contains no clauses (i.e., $G \equiv$ True), then output A and exit;
            If $f = 0$, choose j at random from $1..n$ and flip $A[j]$.
    Output "Not found".

---

algorithm takes the input formula and modifies it step by step, sometimes also modifying the current assignment. At each step, the algorithm substitutes the value of one variable in the current formula, i.e., replaces a formula G by the formula $G[v \leftarrow t]$ for a variable v and a truth value t. If there are unit clauses, then v is taken from one of them; if the value of v does not satisfy the unit clause and satisfies no other unit clause, it is flipped before the substitution. If there are no unit clauses, the algorithm substitutes the value to the next variable in the chosen permutation (taking the value from the current assignment).

If a period finishes (i.e., all variables are processed), but no variable was flipped during it, the algorithm chooses a variable at random and flips it (in fact, this is a very rare situation). After period finishes, the algorithm chooses a new random permutation, replaces the current formula (which is trivial now) by the input formula, and starts a new period. The number of periods is limited to MAX_PERIODS(F) which may be a function of certain syntactic characteristics of the input formula, e.g., of the number of variables. After the last period finishes, the random walk is restarted from another random initial assignment. If a satisfying assignment is not found after taking MAX_TRIES(F) initial assignments, the algorithm outputs the answer "Not found". One of the practical choices for MAX_PERIODS and MAX_TRIES is to set MAX_TRIES(F) to 1 and MAX_PERIODS(F) to $+\infty$ (clearly, in this case the algorithm will never stop if given an unsatisfiable formula).

Note that instead of choosing a random permutation in the beginning of each period, we could generate it "on the fly", i.e., choose a variable at random every time it is needed (from the set of all variables that had not yet been substituted by their values). However, we formulate our algorithm using permutations to stress its relation to the procedure of Paturi, Pudlák, Saks and Zane [PPZ97,PPSZ98].

A more formal description of the algorithm is given in Fig. 1

2.2 Probabilistic approximate completeness

The following theorem shows that our algorithm is probabilistically approximately complete, i.e., if we set `MAX_PERIODS`(F) to $+\infty$ and `MAX_TRIES`(F) to 1, then for every satisfiable formula and *every* initial assignment, UnitWalk finds a satisfying assignment with probability one.

Theorem 1. *Algorithm UnitWalk is probabilistically approximately complete.*

Proof. We now prove that for any A and for any satisfying assignment S for the formula F during one period either the Hamming distance[1] between A and S decreases with probability bounded from below or the algorithm outputs a satisfying assignment.

Consider any satisfying assignment S and an assignment A at distance d of the assignment S. We now construct a permutation π such that if it is chosen in the beginning of the period starting with A (note that every permutation is chosen with probability $\frac{1}{n!}$), then the assignment obtained in the end of the period will be closer to S than A.

Let $\pi[1] = i_1, \ldots, \pi[n-d] = i_{n-d}$, where $x_{i_1}, \ldots, x_{i_{n-d}}$ are the variables on which A agrees with S. Clearly, the values of these variables will not be changed during the period. The remaining variables have different values in A and S. Thus, if at least one of them is flipped during the period, then we are done. Note that these are the only variables whose flippings can be forced by unit clauses. However, if no variable is flipped during the period and satisfying assignment is not found, then after the period finishes, UnitWalk chooses a variable at random and flips it. With probability at least $1/n$, it chooses a variable whose values in A and S are different.

Therefore, every period decreases the Hamming distance between A and S with probability at least $\frac{1}{n \cdot n!}$. The probability of outputting a satisfying assignment in at most n periods is thus at least $b(n) = \frac{1}{(n \cdot n!)^n}$ (irrespectively of the initial assignment). Hence, the overall probability of outputting a satisfying assignment is at least $b(n) + (1 - b(n))b(n) + (1 - b(n))^2 b(n) + \ldots = 1$. $\square$

3 Experimental data

In this section we briefly survey our experiments; see the table below for the numerical data. We refer the reader to [HK01] for more detailed tables and their descriptions.

Benchmarks. The main source where we took benchmarks was SATLIB [HS] (cf. [HS00b]); see references there for the description and original sources of these benchmarks. In addition, we ran our algorithm on Velev's microprocessor verification benchmarks [Vela,Velb,Velc,Veld]. Clearly, we selected only satisfiable benchmarks from all these series.

Our experiments and other sources of data. All our experiments were made on a 466MHz Intel Celeron Pentium II running under Linux. The DIMACS hardware benchmark program *dfmax r500.5* [JT96] takes 43,71 seconds on the machine. The data for other algorithms is partially taken from other sources [Hoo98,HS00a,Pre01]; in order to find their

[1] The *Hamming distance* between two assignments A and B is the number of variables having different values in A and B.

	UnitWalk		GSAT	WalkSAT	Novelty	R-Novelty	SDF	IDB
	substitutions	flips(time)	flips(time)	flips(time)	flips(time)	flips(time)	flips(time)	time
uf100-430	13,500	2,547(0.029)	*	3,652(0.009)	15,699(0.047)	1,536(0.005)	876(0.01)	
par8-5	13,078	1,022(0.022)	*	*	*$2.0 \cdot 10^6$(3.875)	*271,632(0.577)	$1.0 \cdot 10^6$(19.1)	
par16-5-c	$13 \cdot 10^6$	$3.1 \cdot 10^6$(33.85)	*	*	$86 \cdot 10^6$(236.6)	$50 \cdot 10^6$(145.2)	*	
flat50-115	1,521	261(0.002)	*16,857(0.10)	3,896(0.008)	*24,421(0.027)	*9,373(0.019)	773(0.01)	
flat100-239	18,833	3,500(0.051)	*752,775(4.63)	44,900(0.164)	17,893(0.036)	12,345(0.027)	6,983(0.15)	
flat150-360	208,719	46,096(0.499)	*	254,188(0.562)	130,359(0.296)	*731,825(1.429)	39,156(0.86)	
aim100	24,059	4,020(0.048)	*	*251,618(0.445)	*269,700(0.453)	*265,862(0.456)	*116,467(1.89)	
ii8	1,468	203(0.021)	*169,366(7.09)	495(0.021)	5,593(0.180)	*3,166(0.010)	3,946(0.80)	
ii16	9,093	1,086(0.273)	*	6,201(0.769)	*91,031(10.07)	*85,742(4.036)	7,180(1.35)	0.305
ii32	5,989	854(0.227)	*	2,284(1.040)	*88,779(6.775)	*77,589(3.736)	*	1.17
jnh	3,625	484(0.026)	*291,729(2.59)	5,449(0.049)	3,329(0.031)	1,943(0.020)	732(0.04)	
ssa	6,992	707(0.023)	*	45,181(0.171)	*650,156(7.024)	*$1.4 \cdot 10^6$(13.76)	27,238(2.15)	0.31
logistics.a	$40 \cdot 10^6$	$1.2 \cdot 10^6$(920.7)	*	95,205(0.432)	55,748(0.332)	45,220(0.259)		
logistics.b	$1.5 \cdot 10^6$	25,344(17.12)						
logistics.c	$27 \cdot 10^6$	519,865(364.8)						
logistics.d	103,686	4,120(0.940)	*	472,513(3.013)	136,938(1.187)	$1.1 \cdot 10^6$(12.63)	74,090(56.7)	
ais8	56,370	2,875(0.229)	*	28,528(0.152)	*	*$1.2 \cdot 10^6$(6.529)	4,645(0.17)	
ais10	432,191	18,405(6.077)	*	173,422(1.978)	*	*	20,870(1.52)	
ais12	$6.0 \cdot 10^6$	222,967(142.5)	*	*$2.9 \cdot 10^6$(34.28)	*	*	*154,249(18.6)	
bw_large.a	15,174	1,215(0.159)	*	19,158(0.093)	9,699(0.050)	6,911(0.039)	2,917(0.184)	0.425
bw_large.b	215,889	11,848(3.562)	*	480,187(3.793)	203,813(2.005)	347,827(4.443)	38,927(6.44)	7.75
bw_large.c	$25 \cdot 10^6$	$1.0 \cdot 10^6$(610)	*	$14 \cdot 10^6$(151.8)	$6.4 \cdot 10^6$(200.8)	*	*	
2bitadd_11	6,762	297(0.011)	32,517(0.56)	1,174(0.005)	*99,489(0.357)	*139,062(0.498)	166,089(11.1)	0.026
2bitadd_12	4,566	254(0.009)	23,554(0.44)	751(0.004)	*99,478(0.411)	*53,076(0.194)	111,328(8.07)	0.028
3bitadd_31	753,820	4,778(7.164)	*	28,511(1.090)	*	210,907(19.54)	*	11.0
3bitadd_32	416,921	3,743(4.757)	*	14,209(0.628)	*	*	*	8.6
3blocks	116,808	3,193(4.851)	*	33,256(0.643)	13,696(0.270)	8,304(0.179)	10,561(2.60)	
qg5-11	777,304	35,198(3,327)						
qg3-08	86,016	6,539(1,992)						
qg2-07	40,240	3,363(8,446)						
qg1-07	60,745	4,791(6,938)						

Table 1. An asterisk means that MAX_FLIPS(F) has been reached in one or more runs. An asterisk alone means that the algorithm failed in all tries. A completely empty cell means that we do not have data.

CPU time, we also had to make some experiments ourselves, by taking the implementations from `http://www.cs.washington.edu/homes/kautz/walksat/` and `http://logos.uwaterloo.ca/~dale/`. The data for IDB is taken from [Pre01] where the experiments were made on a 400 Mhz Intel Pentium II.

The number of restarts for most algorithms in [Hoo98,HS00a] as well as for our algorithm was set to one, i.e., an algorithm performs just one (long) random walk. For other algorithms we could not present some data unless MAX_FLIPS would be set to a finite number, because they could occasionally loop forever (especially, essentially incomplete algorithms). We set MAX_FLIPS $= 10,000,000$ for `par8`, `par16-c` and `ais12` series and MAX_FLIPS $= 1,000,000$ on other benchmarks.

Conclusion concerning our experiments. Our experiments show that even the current "rough" implementation of UnitWalk is very competitive on many benchmarks. Note that UnitWalk makes substantially less flips on the average than other algorithms on `par`, `aim`, `ii`, `jnh`, `ssa`, `logistics.d`, `bw_large.a`, `bw_large.b` and some of the Bejing benchmarks. On `par8-5`, `par16-5-c`, `aim` and `ssa` it even makes less substitutions than the number of flips made by the other algorithms. Regarding CPU time, UnitWalk "wins" on `aim`, `par8-5`, `par16-5-c` and some of the Bejing benchmarks. It also shows competitive results on many other benchmarks. Note that UnitWalk solved 4 of Zhang's QG instances [ZH94], all satisfiable Velev's SSS-SAT.1.0 [Vela], SSS.1.0 [Velb] and SSS.1.0a [Velc] instances, and 84 out of 100 VLIW-SAT.1.1 [Veld] instances.

The benchmarks that appeared the most hard (but still solvable) for UnitWalk were logistics.a, bw_large.c, ais12 and some of the microprocessor verification benchmarks. UnitWalk was unable to solve f (LRAN), hanoi and bmc series, some of qg, 9vliw and large par benchmarks.

4 Conclusion and Further Research

In this paper we suggested a new local search algorithm for SAT which we call *UnitWalk*. The main difference of our algorithm from other local search algorithms for SAT is the use of unit clause elimination (which is widely used in complete algorithms but seemed hard to combine with local search). Surprisingly, even the first implementation of our algorithm shows very competitive results; the experimental data we present in the paper show that our algorithm dominates on some sets of benchmarks and is able to solve some very hard SAT instances. It shows substantially worse behaviour on some other benchmarks. In these cases sometimes modifications of our algorithm (like adding resolvents or limiting the number of periods) help.

However, unit clause elimination is not the only heuristic that can be used for simplifying formulas and for forcing the values of variables (cf. the general scheme [DHIV01] of a [PPSZ98]-like algorithm). We can also add pure literal rule, bounded resolution rule (after each substitution or similarly to [PPSZ98] where it is used for preprocessing) and various heuristics based on processing the 2-clauses appearing in the formula. *We are currently experimenting with these heuristics and hope to present new results at the symposium.* The first encouraging results are that these heuristics substantially improve the behaviour of the algorithm on logistics benchmarks, e.g., the running time on logistics.a becomes three times smaller; the numbers of flips and substitutions reduce even more.

We refer the reader to [HK01] for other directions of the research. The C source code of an implementation of UnitWalk is available from http://logic.pdmi.ras.ru/~arist/UnitWalk/.

References

[BS97] R. Bayardo and R. Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the 14th National Conference on Artificial Intelligence, AAAI'97*, pages 203–208, 1997.

[CGH00] J. Culberson, I. P. Gent, and H. H. Hoos. On the probabilistic approximate completeness of WalkSAT for 2-SAT. Technical Report APES-15A-2000, Department of Computer Science, University of Strathclyde, 2000.

[DGH+02] E. Dantsin, A. Goerdt, E. A. Hirsch, R. Kannan, J. Kleinberg, C. Papadimitriou, P. Raghavan, and U. Schöning. A deterministic $(2-2/(k+1))^n$ algorithm for k-SAT based on local search. *Theoretical Computer Science*, 2002. To appear.

[DHIV01] E. Dantsin, E. A. Hirsch, S. Ivanov, and M. Vsemirnov. Algorithms for SAT and upper bounds on their complexity. Technical Report 01-012, ECCC, 2001. Electronic address: ftp://ftp.eccc.uni-trier.de/pub/eccc/reports/2001/TR01-012/index.html. A Russian version appears in *Zapiski Nauchnykh Seminarov POMI*, vol. 277, pp. 14–46, 2001.

[GG94] J. Gu and Q.-P. Gu. Average time complexity of the SAT1.2 algorithm. In *Proceedings of the 5th Annual International Symposium on Algorithms and Computation, ISAAC'94*, volume 834 of *Lecture Notes in Computer Science*, pages 146–154. Springer, 1994.

[GPFW97] J. Gu, P. W. Purdom, J. Franco, and B. W. Wah. Algorithms for satisfiability (SAT) problem: A survey. In D. Du, J. Gu, and P. M. Pardalos, editors, *Satisfiability Problem: Theory and Applications (DIMACS*

Workshop March 11-13, 1996), volume 35 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 19–152. AMS, 1997.

[Gu92] J. Gu. Efficient local search for very large-scale satisfiability problems. *ACM SIGART Bulletin*, 3(1):8–12, 1992.

[GW93] I. P. Gent and T. Walsh. Towards an understanding of hill-climbing procedures for SAT. In *Proceedings of the 11th National Conference on Artificial Intelligence, AAAI'93*, pages 28–33, 1993.

[GW95] I. P. Gent and T. Walsh. Unsatisfied variables in local search. In J. Hallam, editor, *Hybrid Problems, Hybrid Solutions*, pages 73–85. IOS Press, 1995.

[Hir00] E. A. Hirsch. SAT local search algorithms: Worst-case study. *Journal of Automated Reasoning*, 24(1-2):127–143, February 2000.

[HK01] 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.

[Hoo98] H. H. Hoos. *Stochastic Local Search — Method, Models, Applications.* PhD thesis, Department of Computer Science, Darmstadt University of Technology, 1998.

[Hoo99] H. H. Hoos. On the run-time behaviour of stochastic local search algorithms for SAT. In *Proceedings of the 16th National Conference on Artificial Intelligence, AAAI'99*, pages 661–666, 1999.

[HS] H. H. Hoos and T. Stützle. SATLIB. A collection of SAT-related resources, `http://www.satlib.org/`.

[HS00a] H. H. Hoos and T. Stützle. Local search algorithms for SAT: An empirical evaluation. *Journal of Automated Reasoning*, 24(4):421–481, 2000.

[HS00b] H. H. Hoos and T. Stützle. SATLIB: An online resource for research on SAT. In *Highlights of Satisfiability Research in the Year 2000*, volume 63 of *Frontiers in Artificial Intelligence and Applications*, pages 283–292. IOS Press, 2000.

[JT96] D. S. Johnson and M. A. Tricks, editors. *Cliques, Coloring and Satisfiability*, volume 26 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science.* AMS, 1996.

[KP92] E. Koutsoupias and C. H. Papadimitriou. On the greedy algorithm for satisfiability. *Information Processing Letters*, 43(1):53–55, 1992.

[LA97] C. M. Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 366–371, 1997.

[MSK97] D. McAllester, B. Selman, and H. Kautz. Evidence in invariants for local search. In *Proceedings of the 14th National Conference on Artificial Intelligence, AAAI'97*, pages 321–326, 1997.

[Pap91] C. H. Papadimitriou. On selecting a satisfying truth assignment. In *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science, FOCS'91*, pages 163–169, 1991.

[PPSZ98] R. Paturi, P. Pudlák, M. E. Saks, and F. Zane. An improved exponential-time algorithm for k-SAT. In *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science, FOCS'98*, pages 628–637, 1998.

[PPZ97] R. Paturi, P. Pudlák, and F. Zane. Satisfiability coding lemma. In *Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science, FOCS'97*, pages 566–574, 1997.

[Pre01] S. D. Prestwich. Local search and backtracking vs non-systematic backtracking. In *AAAI 2001 Fall Symposium on Using Uncertainty within Computation*, 2001. To appear.

[Sch99] U. Schöning. A probabilistic algorithm for k-SAT and constraint satisfaction problems. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science, FOCS'99*, pages 410–414, 1999.

[Sim] L. Simon. The SAT-Ex Site. A collection of experimental results about complete SAT solvers, `http://www.lri.fr/~simon/satex/satex.php3`.

[SKC94] B. Selman, H. A. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proceedings of the 12th National Conference on Artificial Intelligence, AAAI'94*, pages 337–343, 1994.

[SLM92] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the 10th National Conference on Artificial Intelligence, AAAI'92*, pages 440–446, 1992.

[SS00] D. Schuurmans and F. Southey. Local search characteristics of incomplete SAT procedures. In *Proc. of AAAI'2000*, pages 297–302, 2000.

[SSW01] R. Schuler, U. Schöning, and O. Watanabe. An improved randomized algorithm for 3-SAT. Technical Report TR-C146, Dept. of Mathematical and Computing Sciences, Tokyo Inst. of Tech., 2001.

[Vela] M. N. Velev. Benchmark suite SSS-SAT.1.0. Available from: `http://www.ece.cmu.edu/~mvelev/`.

[Velb] M. N. Velev. Benchmark suite SSS.1.0. Available from: `http://www.ece.cmu.edu/~mvelev/`.

[Velc] M. N. Velev. Benchmark suite SSS.1.0a. Available from: `http://www.ece.cmu.edu/~mvelev/`.

[Veld] M. N. Velev. Benchmark suite VLIW-SAT.1.1. Available from: `http://www.ece.cmu.edu/~mvelev/`.

[ZH94] H. Zhang and J. Hsiang. Solving open quasigroup problems by propositional reasoning. In *Proceedings of International Computer Symposium, Hsinchu, Taiwan*, 1994.

[Zha93] H. Zhang. A decision procedure for propositional logic. *Assoc. Automated Reasoning Newslett.*, 22:1–3, March 1993.

[ZS00] H. Zhang and M. Stickel. Implementing the Davis-Putnam method. *Journal of Automated Reasoning*, 24(1-2):277–296, 2000.