Walksat in the 2004 SAT Competition

Henry Kautz
Dept. Computer Science & Engineering
University of Washington
Seattle, WA USA

Bart Selman
Dept. Computer Science
Cornell University
Ithaca, NY USA

David McAllester
Toyota Technical Inst. at Chicago
Chicago, IL USA

The first convincing demonstration that local search could be used to solve challenging satisfiability problems was provided by the GSAT algorithm [9]. GSAT performs gradient descent search in the space of complete truth assignments, where adjacent assignments differ on a single variable and the objective function is the number of clauses not satisfied by the assignment. Like all local search routines GSAT can become trapped in a local minima. One technique for reducing this problem is to randomly alternate between greedy minimizing moves and "noisy" moves that are randomly selected from the variables that appear in unsatisfied clauses [7].

The Walksat algorithm [8] is based on the insight that such noisy moves could be made the basis for local search. Rather than trying to globally determine the best move, Walksat first randomly chooses an unsatisfied clause, and then selects a variable to flip within the clause. Because Walksat may overlook the best global move it is said to perform hill-climbing rather than gradient descent. The fact that a clause is unsatisfied means that at least one of the variables in the clause *must* be flipped in order to reach a global solution. If variables are chosen randomly from the clause and the clause length is bounded by a constant k it is easy to see that each flip as a $1/k$ or better chance of being correct. When $k = 2$ a pure random walk strategy will solve a satisfiable formula over n variables with high probability in $O(n^2)$ time [5].

For larger values of k the only worst-case guarantees for pure random walk are exponential. In practice, therefore, the variable to be flipped is chosen from the (randomly selected) unsatisfied clause by some greedy heuristic. A number of such heuristics have been studied.

The original Walksat heuristic, denoted "SKC" after the initials of the authors [8], employs the notion of the *breakcount* of a variable, which is the number of clauses that are currently satisfied that would become unsatisfied if the variable were to be flipped. Similarly, the *makecount* of a variable is the number of clauses current unsatisfied that would become satisfied. The SKC variable selection heuristic is as follows: (1) If there are variables with breakcount=0, choose one such variable at random. (2) Otherwise, with some fixed probability p select a variable randomly from the clause. (3) Otherwise, pick a variable with minimum breakcount; if there are several such, pick one at random.

The first step, checking for any breakcounts of 0, is crucial for good performance of the SKC heuristic. If this step is eliminated and variables are chosen to minimize (breakcount - makecount), then the algorithm is similar to GSAT, modulo the fact that variable selection is not done globally, but only after clause selection.

The breakcounts and makecounts of all variables can be maintained incrementally, an implementation decision that is crucial for practical efficiency of Walksat. At the start of the algorithm the values are initialized. When a variable x is flipped from true to false, the breakcounts can be updated by *(i)* visiting each clause that contains x positively; and if the clause now contains single true literal l, incrementing the breakcount of the variable of l; and *(ii)* visiting each clause containing x negatively; and if the clause now contains exactly one other true literal l, decrementing the the breakcount of the variable of l. Similar operations are performed for the case of flipping from false to true or cal-

culating the makecount (if the heuristic makes use of the makecount).

The best value for p depends upon the particular problem instance. Random 3-SAT problems typically are solved most quickly with $p = 0.5$. Structured instances often require a lower p value in order to be solved. There is evidence that the optimal p value is close to the value at which the quantity μ/σ is minimized, where μ is the average number of unsatisfied clauses during the run, and σ is the standard deviation of this quantity [4]. A version of Walksat called "AutoWalksat" tries to find the best p value for a given instance by computing this ratio for various settings of p during a series of initial short partial runs [6].

One of the most powerful Walksat heuristics, called "Rnovelty", modifies the random walk strategy by trying to avoid repeatedly flipping the same small set of variables [4]. Every time a variable is flipped the timestamp of the flip is recorded. Once an unsatisfied clause is chosen Rnovelty proceeds as follows:

- Sort the variables in the clause by (breakcount - makecount).
- If two or more variables have the best score, pick one that is *not* the variable from the clause that was most recently flipped.
- Otherwise consider the best and second-best variable under the sort. If the best variable is not the most recently flipped variable in the clause, then select it.
- Otherwise when the difference in the score between the best and second best is greater than 1, pick the best; otherwise pick the second best.

This description of Rnovelty is a simplified case when a parameter $p = 0.5$; for full details see [4]. Rnovelty (unlike SKC) can, on rare occasion, become stuck in deterministic cycles of flips. Such cycling can be prevented by injecting a pure random-walk flip every 100 steps; the resulting heuristic is called Rnovelty+ [2].

Instead of running until a solution is found or a hard time-out limit is reached, Walksat can be run repeatedly from different random starting states. Whether such restarts are beneficial depends upon the run-time distribution of random runs of Walksat for the particular problem instance. If the tail of the distribution drops off at an exponential or higher rate, then restarts are unnecessary. Otherwise, a good restart strategy could provide expected exponential savings.

Choosing a practical restart strategy is not an easy matter. A strategy that interleaves short and increasingly longer runs is provably within a constant times a log factor of optimal for any stochastic process [3]. Starting with runs of length 1, however, is rarely practical; so to usefully apply this strategy one still must choose an initial cutoff c. Recent work on restart strategy for backtracking search algorithms suggests it may be possible to learn syntactic features of formulas that predict good cutoff values [1], but a definitive approach for local search is an open problem.

We entered the three variants of Walksat described above in the 2004 SAT competition: SKC, AutoWalksat, and Rnovelty+. Walksat SKC was run with $p = 0.4$ and Rnovelty+ with $p = 0.5$. All versions we set to perform a single long run on each problem instance: that is, no restart strategy was employed. While this simplified preparing our programs for the competition, it means that negative results for any of the versions are not conclusive. After the competition we will rerun the solvers on the competition problems with a variety of restart strategies to see if doing so significantly improves their performance.

References

[1] Pascal Van Hentenryck, editor. *Restart Policies with Dependence among Runs: A Dynamic Programming Approach*, volume 2470 of *Lecture Notes in Computer Science*. Springer, 2002.

[2] H. Hoos. On the run-time behaviour of stochastic local search algorithms for sat. In *Proc. AAAI-99*, 1999.

[3] Michael Luby, Alistair Sinclair, and David Zuckerman. Optimal speedup of las vegas algorithms. In *Israel Symposium on Theory of Computing Systems*, pages 128–133, 1993.

[4] David McAllester, Bart Selman, and Henry Kautz. Evidence for invariants in local search. In *Proc. AAAI-97*, pages 321–326, 1997.

[5] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization*. Prentice-Hall, 1982.

[6] Donald J. Patterson and Henry Kautz. Auto-walksat: a self-tuning implementation of walksat. *Electronic Notes in Discrete Mathematics (ENDM)*, 9, 2001.

[7] Bart Selman, Henry Kautz, and Bram Cohen. Noise strategies for improving local search. In *Proc. AAAI-94*, pages 337–343, 1994.

[8] Bart Selman, Henry Kautz, and Bram Cohen. Local search strategies for satisfiability testing. In David S. Johnson and Michael A. Trick, editors, *Cliques, Coloring, and Satisfiability: the Second DIMACS Implementation Challenge,* pages 521–532. AMS, 1996.

[9] Bart Selman, Hector J. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proc. AAAI-92*, pages 440–446, 1992.