UNITWALK PERFORMANCE EVALUATION

by

Jari Rosti

Tik-79.194 Seminar on Theoretical Computer Science

Helsinki University of Technology

2003/04

Abstract

We evaluate the performance of UnitWalk SAT Solver and algorithm. UnitWalk-algorithm is a probabilistic local search algorithm, which is based on the Unit Clause Propagation.
The results suggest that the median of the solving time of satisfiable random 3-SAT instances has a polynomial execution time dependency on the number of variables. The behaviour of the UnitWalk is robust for random 3-SAT instances. Structural instances, primality testing and bounded model checking instances, were found to be very hard to solve with UnitWalk.

1. UnitWalk

The satisfiability problem (SAT) is the most fundamental problem in mathematical logic and computing theory. In recent years, many significant improvements have been gained for solving SAT problems /1/. The goal for the seminar course is to achieve overall view of the leading edge of the algorithmic development and actual SAT solvers. /2/

SAT can be expressed as follows:

- A set of n variables $\{x_1, x_2, \ldots, x_n\}$
- A set of literals. A literal is a variable $(Q=x)$ or a negation of a variable $(Q=\neg x)$
- A set of m distinct clauses: $\{C_1, C_2, \ldots, C_m\}$. A clause is set of disjunction of literals.

The goal of the satisfiability problem is to determine whether there exists an assignment of truth values to variables, which makes formula in Eq. 1 satisfiable i.e. it is *true*, when the assignment to variables is evaluated.

$$C_1 \wedge C_2 \wedge \ldots C_m \tag{1}$$

The topic of this work was to evaluate the performance and the usefulness of UnitWalk /3/ SAT solver. This report consists following parts:

- theoretical part, which discusses approach taken by UnitWalk SAT algorithm and compares it briefly to other methods (Chapters 2 and 3)
- experimental part, which presents measurements, where implementation of UnitWalk algorithm is evaluated (Chapter 4)

2. SAT Solvers

SAT solvers can be classified by various ways. Table 1 contains degrees of freedom and selecting one entry from each table row leads algorithm classification /1/

Table 1: algorithm classification

unconstrained	constrained
complete	incomplete
discrete	continuous
parallel	sequential

Constrained algorithms try to satisfy all possible conditions in Eq. (1). Unconstrained algorithms try to minimize e.g. number of unsatisfied clauses. Complete methods are guaranteed to find a satisfiable instance or return the result "`unsat`" (unsatisfiable) and incomplete may return "`satisfiable`" or return "`undefinite`". Discrete formulations perform search by using discrete formulation of SAT and continuous methods are based on continuous space formulations (actual implementations relying on discrete operations, of course). /1/

One classification according to Table 1 is **unconstrained-incomplete-discrete-sequential**. *Local search* is a major class of these procedures. In the local search, one defines a neighbourhood for the problem, guesses a feasible solution and then tries to improve it by selecting new solution from the neighbourhood. The procedure is repeated until the global minimum is found or the maximum number of guesses is approached.

Local minimums of the function to be minimized make greedy local search procedures perform badly. The SAT itself exhibits very complex metric and topology and thus traditional optimisation *heuristics* does not apply well to it (simulated annealing, tabu search combined with trivial neighbourhood). Thus, wide range of different heuristics is proposed for SAT. /1/

Many of these heuristics are *probabilistic* where one uses random variables in the computation when selecting new solution from the neighbourhood. A local search may be guided by a local compass, which may be accurate in a sense, that new trial solution may be closer the goal with certain finite probability. These algorithms are classified as *probabilistically approximately complete (PAC)* /7/

Definition 1: Algorithm is PAC, if for any given problem instance, which has a solution, it returns a solution with probability 1 when the number of the execution time (i.e. iteration steps) approaches infinity (Definition 2 in /7/).

If number of variables in each clause is exactly k, then the problem is called as k-SAT. There is an algorithm for 2-SAT, which finds satisfiable solution with probability $P \geq ½$ in polynomial number of steps. 2-SAT thus belongs to **RP** complexity class /4/. For k-SAT such algorithm does not exist if $P \neq NP$ and random walk algorithms require in the worst case larger than polynomial number of steps for reaching certainty $P \geq ½$. /4/

3. UnitWalk

The UnitWalk is an algorithm and SAT solver developed by E. Hirsch and A. Kojevnikov from St. Petersburg State University. /8/

The classification of the UnitWalk is

- unconstrained-incomplete-discrete-sequential
- based on local search
- probabilistically approximately complete
- local search is guided by unit clause elimination

The unit clause elimination is rarely used in local search algorithms, but it is common in complete-constrained-discrete methods (e.g. DPLL /5/). The unit clause elimination is following procedure:

- Pick a unit clause from C_i from the formula (a unit clause is clause, which contains one literal)

- If the clause is not satisfied by current assignment, then flip the variable value in that clause.
- Remove clauses, which are satisfied by flipping that variable.
- Remove literals of that variable, which are not satisfied

```
x1, x2, ..., xn variables
C1, C2, ..., Cn clauses
F  CNF formula C1∧C2∧...∧Cn
function flip := change value to true if it is false, change value to false if it is true

A := random truth assignment for variables {x1, x2, ..., xn}
for iteration := 1 to oo
   π := Random permutation of {1,2,...,n }
   G := F
   flipped := false
   for i := 1 to n
      while G has unit clause
         select unit clause {xk} or {¬xk} from G
         if (if selected unit clause is not satisfied in A and G does not contain unit
            clause with opposite literal {¬xk} or {xk})
               flip(A[k]) and set flipped := true
               assign A[k] to all xk in formula G, remove all clauses, that
                 assignment forces to be true, remove all literals, which are forced to
                 false
      if variable xπ[i] exists in formula G then assign A[π[i]] to variable xπ[i] in G
   if G is empty, then output A and exit
   if flipped = false then flip something in A
```

Algorithm 1: UnitWalk

Theorem 1: Algorithm UnitWalk is probabilistically approximately complete.

Proof:
A is truth assignment. S is satisfiable truth assignment. Hamming distance of A and S is d (Hamming distance := minimum amount of flips, which is needed to approach from one assignment to another). $x_{i(1)},...,x_{i(n-d)}$ are variables, which has same truth value in A and S.

We select permutation: π =[i(1), ..., i(n-d), ...].
After applying Algorithm 1 by using permutation π we are closer to satisfying truth assignment with finite probability, because:
Algorithm does not change first n-d variables in unit clause elimination. If something after i(n-d) is changed we are closer. If nothing is changed in unit clause elimination, we flip random variable. After flip we are closer to S with probability $P \geq 1/n$. Permutation is chosen with probability 1/n! and d is thus smaller with probability $P \geq 1/(n\ n!)$.
When applying d iterations (iteration is outer for loop in the algorithm) the probability of finding satisfiable solution is at least $p \geq 1/(n\ n!)^d$. Probability of not finding satisfiable solution is 1-p, which is smaller than 1. Now, when k times d iterations is performed, then the probability of not finding the solution is $(1-p)^k$, which approaches zero, when k approaches infinity.

4. Measurements

UnitWalk-0.944 implementation was used in all cases but bounded model checking instances (BMC, Chapter 4.3) /8/. BMC instances were run with UnitWalk versions 0.944 and 0.98.

The seed was initiated to 1 in all cases except data shown Figs. 4 and 5, where the seed was free parameter.

Random 3-SAT instances were generated by using MakeWFF-program /9/. Seeds for random 3-SAT problems were generated by computing MD5 sums from /dev/random-stream. Instances for primality testing were generated with *genfacbm*-program /10/.

Execution times refer to CPU-cycles counted by operating system i.e. measured with user command /usr/bin/time. Minimum time resolution of time-command was 0.01 seconds. All measurements were run in 1GHz Linux machine.

4.1. Random 3-SAT

Measurements were made by varying number of variables so that ratio c/v was 4.258. The ratio was selected so that approximately half of instances would be satisfiable and other half would be unsatisfiable and still being in hard SAT regime. /6/

Number of variables were varied from 40 to 3000 and 30 measurements were made with each number of variables. The cut-off time was increased when solving larger instances. For instances larger than 2000 variables it was 30 minutes and instances below that 15 minutes.

Results are shown in the Fig. 1.

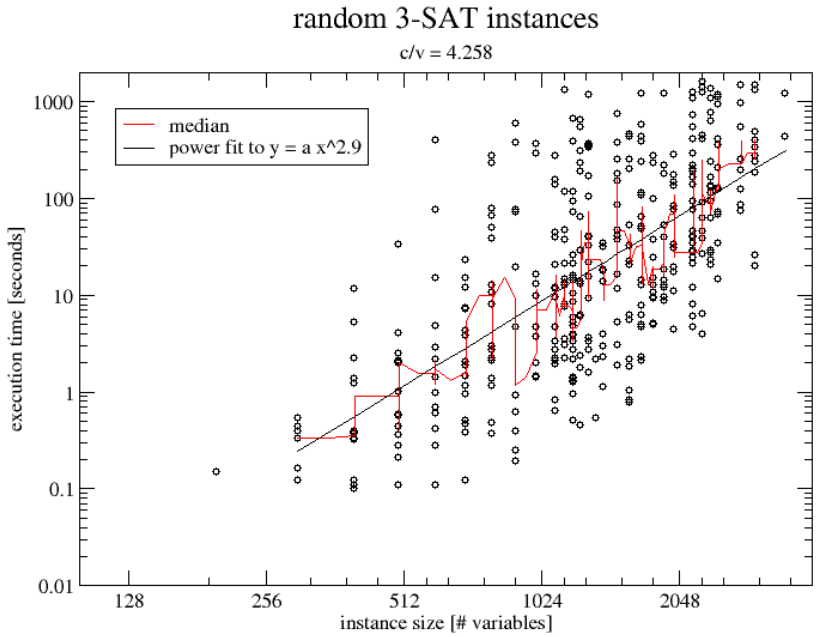

Figure 1: Random 3-SAT instances.

The function $f(x)=ax^b$ was fitted to all points in the Fig. 1. The fit parameter b was approximately 2.9. Fraction line is running median. Results suggest that the median has a polynomial time dependency on the number of variables, when the algorithm is applied for random 3-SAT instances. It was also noted that the standard deviation grows polynomially when the size of the instance is increased. However, too low upper bound and low number of measurements may twist results.

4.2. Factoring

Satisfiable instances of SAT were generated by using the *genfacbm*-program /10/. Braun-circuits were used. The width of measured instances was 15-25 bits and 10-20 instances were computed in one measurement point. Lack of time forced to limit the number of measurements, but 22 bits limit for solving the problem within 30 minutes is quite credible result.

Results are in the Fig. 2.

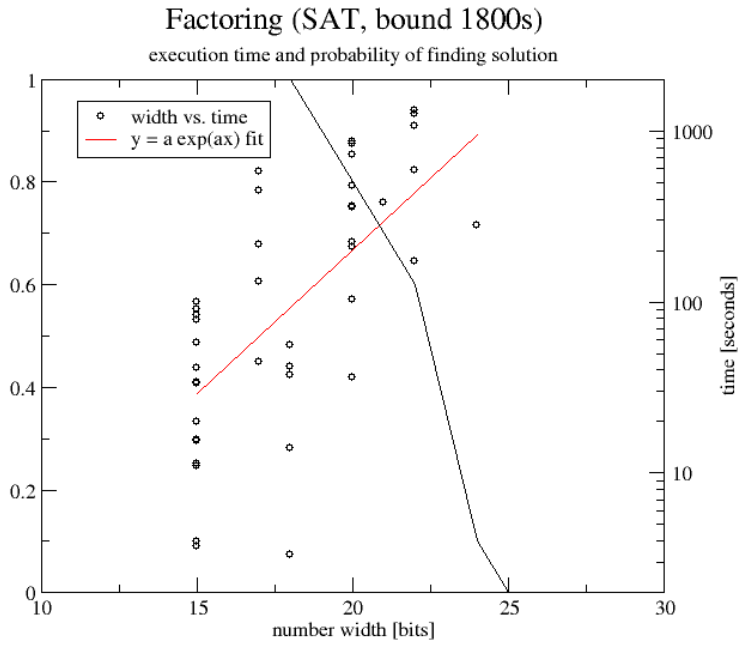

Figure 2: Factoring. Decreasing solid line corresponds probability, in which satisfiable solution was found.

4.3. Bounded Model Checking

Bounded model checking instances check if whether a given model M (typically a hardware design) satisfies a temporal property P in all paths with length less or equal to some bound k /11,12/.

Set of BMC-problems was run with cut-off time 30 minutes. Instances are presented in the table in SATLIB www-page in the Ref. /12/ (Instances #1-#13). One satisfiable solution was found for the instance *bmc-ibm-2.cnf*. Solving it produced following characteristics: Period=1091 (=iterations) Assigns=3065807, Flips=13484, Time=11.186654s.

4.4. Solving time characteristics

Clause / variable ratios c/v={4.1,4.2,4.3}were used, when 100 instances of random 3-SAT were run with 1000 variables. The probability density of running times is shown in the Fig 3. The area under solid lines and a bar represent probability densities i.e. those are normalized to unity. Both lines and bars are used in order to clarify the Figure.

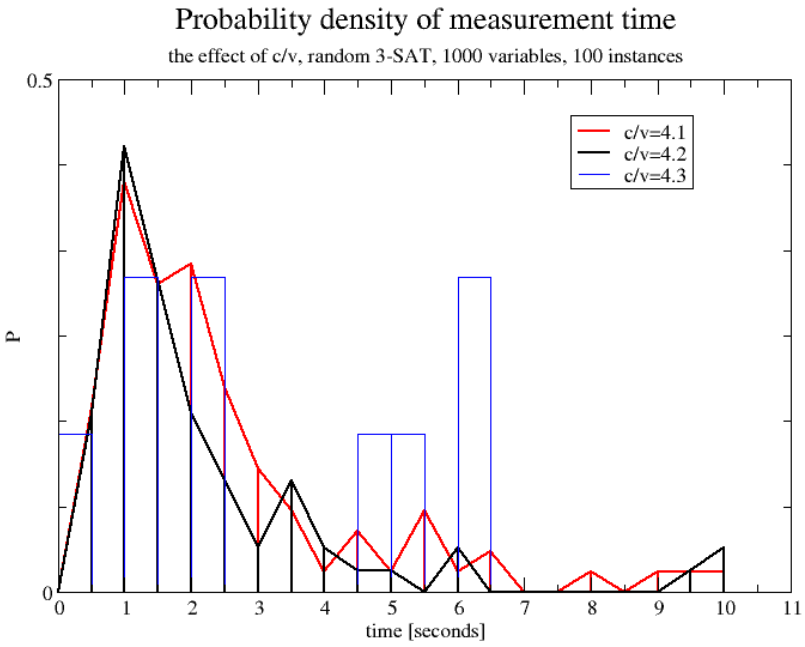

Figure 3: Probability density of the measurement time.

The number of instances was 100, 92 and 13 for 4.1, 4.2 and 4.3 respectively. Different behaviour may be expected because experimental results with

complete algorithms suggest higher execution time in the hard SAT regime close to phase transition from SAT to UNSAT /6/. However, one would expect that in the hard SAT regime instances would be harder, which is not the case, even if it is based similar heuristics than common complete methods like most DPLL solvers /5/. One would naively speculate, if the execution time for random local search based solvers is related to number of solutions, which clearly decreases when c/v increases.

Figures 4 and 5 show the probability density of the running time, when UnitWalk seed is varied. c/v is in both cases 4.258, but the instance is selected to be "easy" in 1200 variable case (Fig. 4) and "hard" but solvable in 1300 variable case (Fig. 5). The "easy case" seems to be easy despite of the starting assignment. The "hard case" is solvable quite fast in some cases, but the distribution seems to be surprisingly broad. Contrary to e.g. some instances in factoring (which were all satisfiable, but not solvable in decent time) UnitWalk solved the problem in all but two cases.

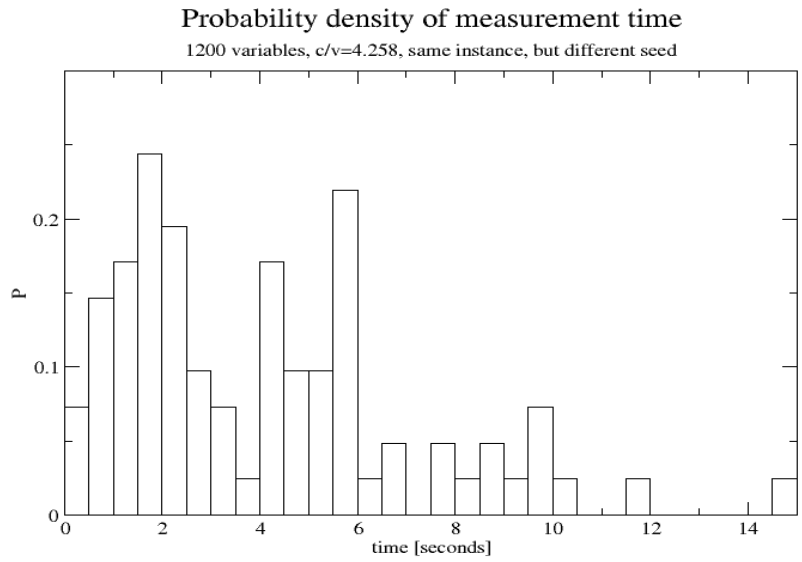

Figure 4: Probability density of measurement time.

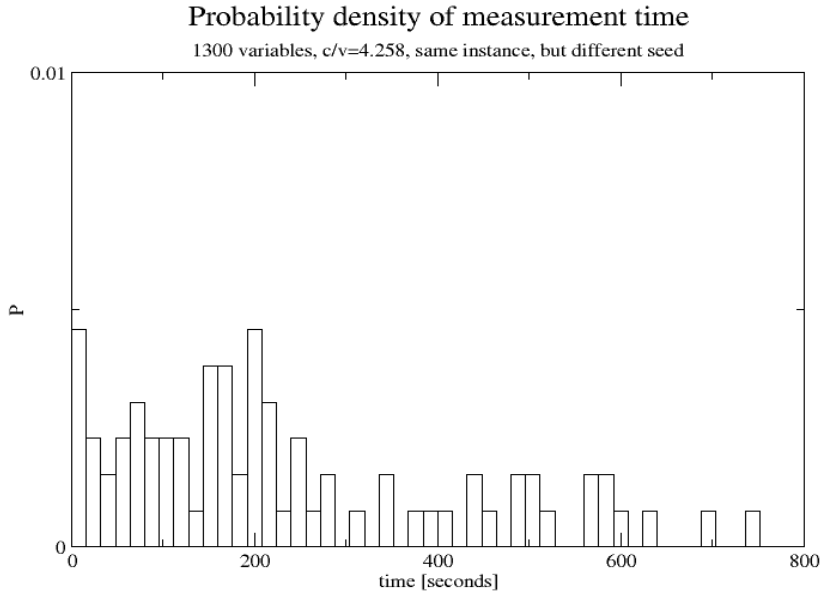

Figure 5: Probability density of running time.

5. Conclusions

UnitWalk is quite robust randomised heuristic for solving SAT problem. It was not well suited for "realistic" problems tested during this evaluation (factoring and BMC). However, it showed quite satisfiable results for random 3-SAT instances and reliable convergence to the solution without restarts.

The results suggest that the median of the solving time of satisfiable random 3-SAT instances has a polynomial execution time dependency on the number of variables with an exponent approximately 3.

REFERENCES

/1/ Gu, J., Purdom, P.W., Franco, J., and Wah, B.W., *Algorithms for Satisfiability (SAT) Problem: A Survey*, Discrete Mathematics and Theoretical Computer Science: Satisfiability (SAT) Problem, American Mathematical Society, Vol. 35, pp. 19--152, (1997)

/2/ Seminar in Theoretical Computer Science, http://www.tcs.hut.fi/Studies/T-79.194/, Helsinki University of Technology (2003)

/3/ E. A. Hirsch, A. Kojevnikov, *Solving Boolean satisfiability using local search guided by unit clause elimination*, 7th International Conference on Principles and Practice of Constraint Programming, CP2001, LNCS 2239, pp. 605-609, November (2001).

/4/ C. Papadimitriou *Computational Complexity,* Addison-Wesley (1994)

/5/ M. Davis, G. Logemann, D. Loveland, A *Machine Program For Theorem Proving,* CACM 5, pp 394-397, *(*1962)

/6/ J.M. Crawford and L.D. Auton. *Experimental Results on the Crossover Point in Random 3SAT*, Artificial Intelligence 81, pp 31-57, (1996)

/7/ Holger H. Hoos: *On the Run-time Behaviour of Stochastic Local Search Algorithms for SAT*, Proc. of AAAI-99, pp.661-666, MIT Press, (1999)

/8/ A. Kojevnikov, *Unit Walk Home Page*, http://logic.pdmi.ras.ru/~arist/UnitWalk/index.html

/9/ B. Selman, H. Kautz, *MakeWFF Home Page*, http://www.cs.washington.edu/homes/kautz/walksat/

/10/ T. Pyhälä*, genfacbfm,* http://www.tcs.hut.fi/%7Etpyhala/genfacbm.tgz

/11/ A. Biere, A. Cimatti, E. Clarke and Y. Zhu, *Symbolic Model Checking without BDDs,* Proc. of the Workshop on Tools and Algorithms for the Construction and Analysis of Systems, TACAS99, lncs, springer, (1999)

/12/ Bounded Model Checking instances, http://www.intellektik.informatik.tu-darmstadt.de/SATLIB/Benchmarks/SAT/BMC/description.html, from SATLIB