# EagleUP: Solving Random 3-SAT using SLS with Unit Propagation

Oliver Gableske[1] and Marijn Heule[2]

[1] Institute of Theoretical Computer Science, Ulm University, Germany
[2] Algorithmics Group, Delft University of Technology, The Netherlands

While the application of unit propagation (UP) is of vital importance in systematic search solvers to solve structured problems of the SAT competitions [8], its application in stochastic local search (SLS) solvers is rare. Examples for combining UP with SLS solvers are `UnitWalk` [4] and `QingTing` [6] and both solvers show strong performance on structured instances. Despite the success on structured formulas, the application of UP in SLS only seemed to weaken the performance on random $k$-CNF formulas. The approach described in this abstract briefly presents how UP can be embedded in SLS solvers to boost their performance on random 3-CNF formulas as well.

In summary, several questions must be answered in order to embed UP into a given SLS solver: When to call for UP in the ongoing SLS search? What variable assignments is UP supposed to propagate in what order? And finally, how to use the result of a UP call? For the following explanations, we assume the ongoing search on a CNF formula $F$ of a G2WSAT solver [5] with current assignment $\alpha$.

**When to call for UP?** It is reasonable to interrupt the SLS search as soon as it cannot improve its current assignment $\alpha$ anymore. This is usually the case as soon as it has no further promising variables to flip (we call this a *dead end*).

**What variable assignments is UP supposed to propagate in what order?** The idea is to use the dead end assignment $\alpha$ and propagate all the assignments made herein, because such an assignment usually satisfies a large portion of the clauses of $F$. Propagation stops as soon as UP runs into a conflict. The result is a (partial) assignment $\beta$, that hosts all variable assignments that could be propagated by UP without running into a conflict.

The variable ordering we use is computed according to a recursive weight heuristic. The general idea of recursive weight heuristics is to help systematic search SAT solvers identify variables with strong impact on the formula. Such solvers usually pick a single variable in every node of their search tree and assign it to a not yet explored value. Picking variables with strong impact will then give a large reduction of the remaining formula, and therefore, a strong reduction in the size of the remaining search space.

To create a variable ordering, we use the recursive weight heuristic RW [3, 1]. We create a variable ordering $\theta_{\mathrm{RW}_5}$ such that with $x_i, x_j \in \mathrm{VAR}(F)$: $\theta_{\mathrm{RW}_5}(x_i) < \theta_{\mathrm{RW}_5}(x_j) \Leftrightarrow \mathrm{RW}_5(x_i) > \mathrm{RW}_5(x_j)$. The ordering $\theta_{\mathrm{RW}_5}$ has to be computed exactly once (i.e., it is static) and prioritizes variables with high impact on $F$.

The reason why we prefer variables with high impact is that assigning these variables first creates new unit clauses sooner. Creating new unit clauses sooner

then means that UP relies less often on the dead end assignment $\alpha$ in future iterations. Given a satisfiable formula, this is helpful since this dead end, currently not satisfying all clauses of the formula, must contain erroneous variable assignments. The less often UP relies on it, the smaller the chance of propagating one of the contained errors.

**How to use the result of a UP call?** Comparing $\alpha$ and $\beta$ on all variables that are assigned in $\beta$ can provide a set of variables that has changed its assignment during the unit propagation. Let us call this set $\mathcal{M}$. In short $\mathcal{M} = \{x \in \mathcal{V} \mid \beta(x) \text{ is defined and } \alpha(x) \neq \beta(x)\}$. The idea is to multi-flip all variables in $\mathcal{M}$ and continue the SLS search from this new position in the search space. In general, applying UP helps the SLS solver to escape from the current dead end to an assignment that has increased consistency regarding the formula.

There is, however, an additional problem that arises from the application of the static variable ordering $\theta_{\mathrm{RW}_5}$. Empirical tests show that a G2WSAT solver will encounter a dead end in about every third flip on a 3-CNF formula. Calling for UP in every third flip will most likely give similar results, and is therefore considered to be a waste of computational time. In order to overcome that problem, it is sufficient to increase the number of flips between two calls to UP (we call this a cool-down period and denote it by $\mathfrak{c}$).

Our approach to compute these cool-down periods uses the Cauchy probability distribution. Let $\gamma \in \mathbb{R}, \gamma > 0$ and $\omega \in \mathbb{R}$. The *cumulative distribution function* of the Cauchy distribution is $C : \mathbb{R} \mapsto \mathbb{R}$, $C(z) = P(Z < z) = 0.5 + 1/\pi \cdot \arctan((z-\omega)/\gamma)$. In summary, whenever UP was performed, a random number $a \in [0, 1)$ is picked and the next cool-down period is then computed as $\mathfrak{c} = \min\{z | C(z) \geq a\}$.

We have performed an empirical study to test the feasibility of our approach. The results show an average speedup of 18% when using a UP enhanced G2WSAT solver in comparison to its pure SLS variant for large size random 3-CNF formulas. Preliminary tests on crafted and application formulas suggest that our approach to combine UP with SLS can be of advantage here as well. The results are available at [7]. This abstract is available as a full paper at [2].

## References

1. Athanasiou, D., Fernandez, M.A.: Recursive Weight Heuristic for Random $k$-SAT. Technical report from Delft University. `http://www.st.ewi.tudelft.nl/sat/reports/RecursiveWeightHeurKSAT.pdf`, 2010.
2. Full paper: `www.uni-ulm.de/in/theo/mitarbeiter/olivergableske.html`
3. Mijnders, S., De Wilde, B., Heule, M.J.H.: Symbiosis of search and heuristics for random 3-SAT. In LaSh'10, 2010.
4. Hirsch, E.A., Kojevnikov, A.: UnitWalk: A New SAT Solver that Uses Local Search Guided by Unit Clause Elimination. AMAI 43(1-4):91-111. Kluwer 2005.
5. Li, C.M., Huang, W.Q.: Diversification and Determinism in Local Search for Satisfiability. In SAT'05, LNCS 3569:158-172. Springer 2005.
6. Li, X.Y., Stallmann, M.F., Brglez, F.: QingTing: A fast SAT solver using local search and efficient unit propagation. In SAT'03, LNCS 2919:452-467. Springer 2003.
7. The results of our study: `http://edacc.informatik.uni-ulm.de/EDACC3/index`.
8. The SAT competition homepage: `http://www.satcompetition.org`.