

EagleUP: Solver Description

Oliver Gableske*

Ulm University,
Institute of Theoretical Computer Science, 89069 Ulm, Germany
`oliver.gableske@uni-ulm.de`

Abstract. This is a description of the **EagleUP** SAT Solver. It briefly covers unit propagation, the WalkSAT framework, introduces the Sparrow heuristic, RW heuristic, the Cauchy probability distribution, and explains how everything is put together to yield an SLS solver that uses unit propagation to solve large random k -SAT instances.

Keywords: EagleUP, SLS, unit propagation, Cauchy distribution, RW heuristic

1 Introduction

We will first give a brief overview of unit propagation and explain WalkSAT in general. On top of this we will describe the Sparrow heuristic that the new solver **EagleUP** uses to escape from local minima. We then give a brief overview of the recursive weight (RW) heuristic and the Cauchy probability distribution. Both are of fundamental importance to introduce unit propagation into our SLS solver. Finally, an overview of the functioning of **EagleUP** is given. There is a more elaborate description of this solver in the form of a paper¹.

2 Iterative unit propagation iUP

Let F be a k -SAT formula with n variables in $\mathcal{V} = \{x_1, \dots, x_n\}$, and m clauses in $\mathcal{C} = \{c_1, \dots, c_m\}$. Let $\alpha : \mathcal{V} \mapsto \{0, 1\}$ be a complete (reference) assignment, and let θ be a permutation of the elements from \mathcal{V} such that $\theta(x_i) = j$ denotes that variable x_i has position j in θ . iUP uses F, α , and θ as given in listing 1. The iUP algorithm will start with an empty assignment β and tries to extend it towards a satisfying assignment of F . This extension happens in two ways.

In case unit clauses are present (yet unsatisfied clauses with only one unassigned literal), iUP will force an assignment to the last remaining variable in that clause such that the clause becomes true.

* Funded by the Graduate School *Mathematical Analysis of Evolution, Information and Complexity* at the University of Ulm.

¹ http://www.uni-ulm.de/fileadmin/website_uni_ulm/iui.inst.190/Mitarbeiter/gableske/SAT11.pdf

```

iUP(CNF formula  $F$ , variable sel. heur. VAR, value sel. heur. VAL, conflictStopFlag)
  Initialize  $\beta := \{\}$ ; //start with an empty assignment
  REPEAT
    IF there is an unsatisfied clause in  $F$  that has all but one literal assigned under  $\beta$ 
      THEN assign the corresponding variable in  $\beta$  such that it satisfies the clause;
      ELSE use VAR to select a in  $\beta$  unassigned variable; use VAL to assign it in  $\beta$ ;
    UNTIL  $\beta$  assigns all variables OR (conflictStopFlag AND conflict is found)
  return  $\beta$ ;

```

Fig. 1. The algorithm performs iterative unit propagation on the CNF formula F using heuristics VAR, VAL and a flag that determines whether to stop once conflicts emerge.

In case no unit clause is present, iUP selects an unassigned variable according to a given variable selection heuristic VAR. In our case, we create an ordering of the variables θ using the RW heuristic and pick the first variable according to this ordering that has not yet been assigned in β . After a variable is selected, a value selection heuristic VAL should decide the preferred truth value for that variable. The value selection heuristic we use is phase-saving which selects the truth value based on a reference assignment α . This assignment stores for each variable the last truth value to which the variable was assigned to. In the context of our work, the reference assignment for phase-saving is provided by the SLS solver when a call to iUP is performed. The following section briefly covers how the SLS solver performs its search.

3 The SLS solver Eagle

This section briefly covers the Sparrow approach from [2] and explains the functioning of our own Sparrow-like SLS solver implementation called **Eagle**.

A Sparrow-like SLS solver is quite similar to WalkSAT solvers. Most modern WalkSAT solvers follow the approach in [4] and work in *two modes*. Starting from a random complete assignment α , they compute variable scores that basically reflect how big the improvement in terms of satisfied clauses in the formula will be when they invert a single variable assignment in α . The inversion of a single variable assignment from $x_i = 1$ to $x_i = 0$ or vice versa is called a flip. The computation of variable scores along with clause weighting schemes like PAWS [8] is complex and requires numerous data structures. For the sake of simplicity we will not cover these schemes here and refer the reader to [4] and [8].

If variables are present that have a non-negative score, WalkSAT solvers will pick the one with highest score and flip it (*greedy mode*), breaking ties in favor of the least recently flipped variable. If no such variable is present, the solver resides in a *local minimum* in which no greedy step is possible. In such a situation, the solver uses a heuristic to decide which variable to flip despite the fact that this will not yield any immediate improvement (*random mode*). There are numerous heuristics that WalkSAT solvers can use when working in random mode. A famous one is *Novelty+* [3] as used in the solver **gNovelty+** [7].

Sparrow introduced a new heuristic that helps WalkSAT solvers to overcome local minima. Let us assume the following situation in order to explain the Sparrow heuristic. The solver resides in a local minimum α . At least one clause is now unsatisfied, and the solver picks one of the yet unsatisfied clauses at random. Let this clause be $u_i = (x_{i_1} \vee \dots \vee x_{i_k})$. For all the variables contained in u_i , the solver computes a probability to flip this variable as follows:

$$p(x_{i_j}) = \frac{p_s(x_{i_j}) \cdot p_a(x_{i_j})}{\sum_{l=1}^k p_s(x_{i_l}) \cdot p_a(x_{i_l})}$$

with $p_s(x_{i_l}) = a_1^{\text{score}(x_{i_l})}$, and $p_a(x_{i_l}) = \left(\frac{\text{age}(x_{i_l})}{a_3}\right)^{a_2}$

In the above equations, $\text{score}(x_{i_l})$ is the current score of variable x_{i_l} according to the WalkSAT heuristic. All scores are negative since the solver is not in *greedy mode*, e.g., is in a local minimum. Furthermore, $\text{age}(x_{i_l})$ is the number of flips that passed since variable x_{i_l} was flipped last. The constants $a_1 = 4$, $a_2 = 4$, and $a_3 = 26500$ have been determined experimentally. Fig. 2 summarizes the functioning of **Eagle**.

Eagle(3-CNF formula F , timeout t)

```

Initialize random complete assignment  $\alpha$ ; Set flips := 0;
WHILE  $\alpha$  does not satisfy  $F$  AND timeout  $t$  is not yet reached
  calculate scores for all variables;
  IF variables with positive score exist
    THEN //greedy mode
       $x :=$  pick variable with highest score,
      breaking ties in favor of the least recently flipped;
      flip  $x$ ; flips++;
    ELSE //random mode
      pick random unsatisfied clause  $u$ ;
      compute sparrow probabilities for all  $x \in u$ ;
      randomly pick a variable according to the computed probabilities;
      flip the picked variable; flips++; adapt clause weights;
  ENDWHILE
IF  $\alpha$  satisfies  $F$  THEN output  $\alpha$ ; ELSE output UNKNOWN;

```

Fig. 2. The functioning of **Eagle** following the approach from [2].

4 Details about iUP: the VAR, VAL, and RW heuristics

RW is based on the work by Mijnders et. al. [6] as well as Athanasiou et. al. [1]. Before we explain how and why we use it, we will give a thorough explanation on how it is computed.

Given a 3-CNF formula F with n variables in \mathcal{V} and $2n$ literals in \mathcal{L} . We write $(l \vee l' \vee l'')$ to denote a clause containing *literal* $l \in \mathcal{L}$ and *two other arbitrary literals* $l', l'' \in \mathcal{L}$. We compute the RW-score of a *variable* $x \in \mathcal{V}$ using the functions $h_i : \mathcal{L} \mapsto \mathbb{R}$ and $RW : \mathcal{V} \mapsto \mathbb{R}$ up to a recursion depth of 5 as follows:

$$\begin{aligned} h_0(l) &= 1 \\ s_{i+1} &= \frac{1}{2n} \cdot \sum_{l \in \mathcal{L}} h_i(l) \\ h_{i+1}(l) &= \sum_{(l \vee l' \vee l'') \in F} \left(\frac{h_i(\neg l') \cdot h_i(\neg l'')}{s_{i+1}^2} \right) \\ RW_5(x) &= h_5(\neg x) \cdot h_5(x) \end{aligned}$$

We create a variable ordering θ_{RW_5} such that $\theta_{RW_5}(x_i) < \theta_{RW_5}(x_j) \Leftrightarrow RW_5(x_i) > RW_5(x_j)$. The ordering θ_{RW_5} has to be computed exactly once (i.e., it is static) and prioritizes variables with high impact on F .

When no unit clause is present, the iUP algorithm must decide what variable to propagate in the current iteration. This decision is made in the variable selection heuristic **VAR** which will pick the first variable according to θ_{RW_5} that has not yet been propagated by iUP (i.e., it is not assigned in β , see Fig. 1).

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 iUP relies less often on the reference assignment in future iterations. Given a satisfiable formula, this is helpful since this reference, currently not satisfying all clauses of the formula, must contain erroneous variable assignments. The less often iUP relies on it, the smaller the chance of propagating one of the contained errors. Fig. 3 summarizes the functioning of **VAR**.

VAR(variable ordering θ_{RW_5})

pick first variable x according to θ_{RW_5} that is unassigned in β ;
return x ; //This x has not yet been propagated in the current call of iUP.

Fig. 3. The functioning of **VAR**. See Fig. 1 for details on how it is used in iUP.

After the variable was selected by **VAR** it must be assigned to a value to complete the current iteration of iUP. The value selection heuristic **VAL** we use here is phase-saving with the current local minimum of the SLS solver as a reference. Fig. 4 summarizes the functioning of **VAL**. We have now completely

VAL(reference assignment α)

for the variable x that was picked by **VAR**, assign $\beta(x) := \alpha(x)$;

Fig. 4. The functioning of **VAL**. See Fig. 1 for details on how it is used in iUP.

explained the interior functioning of `iUP` and our `Eagle` SLS solver. In order to combine both, one needs to clarify when to perform a call to the `iUP` component during the search of the SLS solver. This is done in the next section.

5 When to perform `iUP` in the SLS search: the Cauchy distribution

In order to embed `iUP` into an SLS solver, it must be clarified when `iUP` is to take place during the local search. A reasonable approach would be to have the SLS solver perform its local search and call for `iUP` as soon as it gets stuck in a local minimum. Following this approach has two advantages. First, the SLS solver is allowed to continue its search towards a satisfying assignment without interruption as long as this search seems promising, e.g., is done greedily. Second, a call to `iUP` is only performed in situations where the SLS solver could make use of additional support in order to advance the search again, e.g., to escape from a local minimum.

However, on 3-CNF instances with a ratio of 4.2, the SLS solver will encounter a local minimum in about every third flip. Calling `iUP` in every third flip would mean that multiple `iUP` calls rely on almost identical reference assignments. Given the static variable ordering θ_{RW_3} , the chance for different outcomes of these `iUP` calls is extremely small. Therefore, two calls of `iUP` should be separated by more than three flips in order to not waste computational time. We call this number of separating flips the *cool-down period* and denote it with \mathfrak{c} .

Our approach to compute these cool-down periods is based on the Cauchy probability distribution. The *Cauchy distribution* is defined as follows. Let $\gamma \in \mathbb{R}, \gamma > 0$ and $\omega \in \mathbb{R}$. The Cauchy distribution is given by the *probability density function* (PDF)

$$c : \mathbb{R} \mapsto \mathbb{R}, c(z) = \frac{1}{\pi} \cdot \frac{\gamma}{\gamma^2 + (z - \omega)^2}$$

Its *cumulative distribution function* (CDF) is

$$C : \mathbb{R} \mapsto \mathbb{R}, C(z) = P(Z < z) = \frac{1}{2} + \frac{1}{\pi} \cdot \arctan\left(\frac{z - \omega}{\gamma}\right).$$

Since the density c is symmetric and has its mean in ω , the parameter ω is called the center of the distribution. The parameter γ is called the *width* parameter as it describes how strong the decay around the center is in the PDF. A width parameter close to 0 will give a strong decay in the PDF and a strong increase in the CDF. For more information about the Cauchy distribution see [5].

Actual computations use a discretized CDF of the Cauchy distribution. The idea is to pick a number $a \in [0, 1)$ uniformly at random and identify the smallest cool-down length \mathfrak{c} with $C(\mathfrak{c}) \geq a$. The values for \mathfrak{c} are then distributed according to the Cauchy distribution.

Our tests revealed that the interval $[0, 2.7n]$ is reasonable for discretization, meaning that the cool-down periods have lengths ranging from 0 flips up to $2.7n$ flips. Further, we determined the values $\gamma = 1500$ and $\omega = 2n$ to be suitable.

In summary, whenever `iUP` has been performed a random number $a \in [0, 1)$ is picked. The length of the next cool-down period is then computed as $\mathbf{c} = \min\{z | C(z) \geq a\}$.

Now that it is clear how `iUP` works and when it is called in the SLS solvers search we will summarize the complete functioning of `EagleUP` in the next section.

6 EagleUP: Enhancing Eagle with iUP

The previous sections explained how the SLS solver performs its search, how `iUP` is performed, when `iUP` is performed. The following section summarizes these explanations and presents how `iUP` is embedded into `Eagle` to yield `EagleUP`.

`EagleUP` initializes a set of data structures before the actual search starts. Given the 3-CNF formula F with n variables, it first initializes a complete random assignment α for the formula that is used by the local search. Furthermore, it initializes the static variable ordering θ_{RW_5} using the RW heuristic. Additionally it pre-computes the CDF of the Cauchy distribution in $[0, 2.7n]$ using the experimentally determined parameters $\omega = 2n$, $\gamma = 1500$. The first cool-down period is set to $\mathbf{c} = \omega$.

After the initialization phase, `EagleUP` performs local search as explained in the previous sections. Local search continues in greedy mode until it gets stuck in a local minimum. Once arrived at a local minimum, the solver switches to random mode. In contrast to `Eagle`, the solver now checks if the current cool-down period \mathbf{c} is over. If not, it continues in random mode as done in `Eagle`, following the Sparrow heuristic to decide what variable to flip next. If the cool-down period is over, a call to `iUP` is performed.

A call to `iUP` is performed using the current local minimum α as a reference assignment. After the call to `iUP`, the solver performs two tasks. First, the length of the next cool-down period is computed using the Cauchy distribution. Second, the current local minimum α is partially overridden with the result β from the `iUP` call.

The above scheme is repeated until the solver either finds a satisfying assignment for the formula, or if a given timeout is reached. Fig. 5 summarizes the functioning of `EagleUP`.

7 Summary

This description presented the functioning of iterative unit propagation `iUP` and our new Sparrow-like SLS solver `Eagle`.

Concerning `iUP` we have given brief explanations on how unit propagation works, and that it needs a variable selection heuristic `VAR` and a value selection heuristic `VAL`. We explained that our implementation of `iUP` makes use of the RW heuristic to create a variable ordering which is then used by `VAR` such that the variables with highest impact are assigned first. We furthermore explained that

```

EagleUP(3-CNF formula  $F$ , timeout  $t$ )
  Initialize random complete assignment  $\alpha$ ; Set flips := 0; Set lastIUPCall := 0;
  Initialize  $\theta_{RW_5}$  using the RW heuristic; //  $\theta_{RW_5}$  is not modified again, i.e., is static
  Compute the Cauchy CDF  $C(z), z \in [0, 2.7n]$  with  $\omega := 2n$  and  $\gamma = 1500$ ;
  Set  $c := \omega$ ;
  WHILE  $\alpha$  does not satisfy  $F$  AND timeout  $t$  is not yet reached
    calculate scores for all variables;
    IF variables with positive score exist
      THEN //greedy mode
         $x :=$  pick variable with highest score,
          breaking ties in favor of the least recently flipped;
        flip  $x$ ; flips++;
      ELSE //random mode
        IF flips > lastIUPCall +  $c$ 
          THEN //do iUP
            randomly pick  $a \in [0, 1)$  and set  $c := \min\{z | C(Z) \geq a\}$ ;
            lastIUPCall := flips;
             $\alpha :=$  iUP( $F$ , VAR( $\theta_{RW_5}$ ), VAL( $\alpha$ ), true); //partially override  $\alpha$  with  $\beta$ 
          ELSE //do Sparrow
            pick random unsatisfied clause  $u$ ;
            compute sparrow probabilities for all  $x \in u$ ;
            randomly pick a variable according to the computed probabilities;
            flip the picked variable; flips++; adapt clause weights;
        ENDWHILE
    IF  $\alpha$  satisfies  $F$  THEN output  $\alpha$ ; ELSE output UNKNOWN;

```

Fig. 5. The functioning of EagleUP. See Fig. 1 for the functioning of iUP as well as Fig. 3 and Fig. 4 for the functioning of VAR and VAL.

VAL is implemented following the phase-saving heuristic that uses the complete assignment of the SLS solver as a reference.

After the explanation of iUP we have given an introduction to the Sparrow heuristic and our own Sparrow-like SLS solver Eagle.

We then explained how iUP and Eagle are combined with the help of the Cauchy distribution to yield EagleUP, our new unit propagation enhanced SLS solver.

A more detailed introduction to EagleUP is available in the form of a paper².

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 http://www.uni-ulm.de/fileadmin/website_uni_ulm/iui.inst.190/Mitarbeiter/gableske/SAT11.pdf

2. Balint, A., Fröhlich, A.: Improving Stochastic Local Search for SAT with a New Probability Distribution. Proc. of the 13th international Conference of the Theory and Application of Satisfiability Testing, Springer LNCS 6175, p. 10-16, 2010.
3. Hoos, H.H.: An adaptive noise mechanism for WalkSAT. In AAAI'02, p. 635-660, AAAI Press, 2002.
4. Li, C.M., Huang, W.Q.: Diversification and Determinism in Local Search for Satisfiability. In SAT'05, LNCS 3569, p. 158-172, Springer, 2005.
5. Lupton, R.: Statistics in Theory and Practice. Section 3.7, Cauchy Distribution, p. 21-22. Princeton University Press, ISBN 0-691-07429-1, 1993.
6. Mijnders, S., De Wilde, B., Heule, M. J. H.: Symbiosis of search and heuristics for random 3-SAT. Proc. of the 3rd intl. Workshop on Logic and Search, 2010.
7. Pham, D.N., and Gretton, C.: gNovelty+. Solver description for the SAT 2007 competition. <http://www.satcompetition.org/2007/gNovelty+.pdf>, 2007.
8. Thornton, J., Pham, D.N., Bain, S., Ferreira, V.: Additive versus multiplicative clause weighting for SAT. In AAAI'04, p. 191-196, AAAI Press, 2004.