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

Oliver Gableske[1]

[1]oliver.gableske@uni-ulm.de
Institute of Theoretical Computer Science
Ulm University
Germany

This is a joint work with Marijn Heule.

Pragmatics of SAT, 18.06.2011

Outline

Motivation and Goal of our Work

- Motivation
  - Design of a fast SAT Solver for random k-SAT
  - SLS approach has proven its worth
  - Combining UP and SLS has been successful on structures instances
  - SLS+UP: Does it also work for random ones?

Motivation and Goal of our Work

- Motivation
  - Design of a fast SAT Solver for random k-SAT
  - SLS approach has proven its worth
  - Combining UP and SLS has been successful on structures instances
  - SLS+UP: Does it also work for random ones?
- Goal
  - Improve the performance of a given SLS solver on random instances using UP

SLS

- Task: For a given 3-SAT formula F
  - with n variables, $\mathcal{V} = \{x_1, \ldots, x_n\}$
  - and m clauses, $\mathcal{C} = \{c_1, \ldots, c_m\}$
- Find an assignment $\alpha : \mathcal{V} \to \{0, 1\}$, such that $F(\alpha) = 1$

SLS

- Task: For a given 3-SAT formula F
  - with n variables, $\mathcal{V} = \{x_1, \ldots, x_n\}$
  - and m clauses, $\mathcal{C} = \{c_1, \ldots, c_m\}$
- Find an assignment $\alpha : \mathcal{V} \rightarrow \{0, 1\}$, such that $F(\alpha) = 1$
- To perform search, SLS solvers use
  - a total assignment α
  - an objective function f (number of unsatisfied clauses in F under α)

SLS

```
SLS(3-CNF F, timeout t)
    Randomly initialize α;
    repeat
        if F(α) = 1 then output satisfying assignment; terminate;
        if ∃α' ∈ Neighborhood(α): f(α') ≤ f(α)
        then //greedy mode
            α := α'; //flip the variable that gives the best improvement
        else //random mode
            flip random variable according to some heuristic;
    until timeout t is reached;
    output unknown;
```

Sparrow

- Random mode requires a heuristic to decides what variable to flip
- Sparrow heuristic [ABAF2010] has shown strong performance on 3-SAT

Sparrow

- Random mode requires a heuristic to decides what variable to flip
- Sparrow heuristic [ABAF2010] has shown strong performance on 3-SAT

- In random mode, at least one unsatisfied clause is available
- Sparrow works as follows:
  - Pick one unsatisfied clause at random, $u_i = (x_{i_1} \vee \ldots \vee x_{i_k})$
  - For all the variables in u_i, compute a probability $p(x_{i_j})$ to flip this variable
  - Randomly pick a variable from u_i according to the probability distribution and flip it

Eagle

- The SLS solver we want to improve is called `Eagle`
  - `Eagle` is a from scratch re-implementation of the `Sparrow` solver [ABAF2010]
  - `Eagle` is a G2WSAT solver using the Sparrow heuristic in random mode [OGMH2010]

Eagle

- The SLS solver we want to improve is called Eagle
  - Eagle is a from scratch re-implementation of the Sparrow solver [ABAF2010]
  - Eagle is a G2WSAT solver using the Sparrow heuristic in random mode [OGMH2010]
- The reason why we used it:
  - Eagle shows strong performance on random 3-SAT
  - Improving algorithms that are good by themselves is usually hard
  - Improving a strong SAT solver is a non-trivial task
  - Succeeding in this task is considered to be a useful result

iUP

- Unit Propagation (UP) is well known from the literature and of fundamental importance to systematic search solvers
- The iterated application of unit propagation until saturation is called iUP

iUP

- Unit Propagation (UP) is well known from the literature and of fundamental importance to systematic search solvers
- The iterated application of unit propagation until saturation is called iUP
- Problem: a plain 3-CNF formula does not contain unit clauses, so what do we propagate?
- Solution: if no unit clause is present, pick some variable and propagate a value for it
- Eventually, we get
  - unit clauses
  - the empty clause

iUP

- Unit Propagation (UP) is well known from the literature and of fundamental importance to systematic search solvers
- The iterated application of unit propagation until saturation is called `iUP`

- Problem: a plain 3-CNF formula does not contain unit clauses, so what do we propagate?
- Solution: if no unit clause is present, pick some variable and propagate a value for it
- Eventually, we get
  - unit clauses
  - the empty clause

- `iUP` in general needs three things
  - A variable selection heuristic (`VAR`)
  - A value selection heuristic (`VAL`)
  - The information whether to stop once the empty clause is found (`conflictStopFlag`)

iUP

iUP(3-CNF F, var. sel. heur. VAR, val. sel. heur. VAL, conflictStopFlag)
  Initialize β:=$\{\}$;
  repeat
    if $F(\beta)$ contains a unit clause
    then assign the corresponding variable in β such that it satisfies the clause;
    else use VAR to select a variable unassigned in β; use VAL to assign it in β;
  until β assigns all variables **or** (conflictStopFlag **and** empty clause found)
  return β;

iUP

iUP(3-CNF F, var. sel. heur. VAR, val. sel. heur. VAL, conflictStopFlag)
  Initialize β:={};
  repeat
    if $F(\beta)$ contains a unit clause
    then assign the corresponding variable in β such that it satisfies the clause;
    else use VAR to select a variable unassigned in β; use VAL to assign it in β;
  until β assigns all variables **or** (conflictStopFlag **and** empty clause found)
  return β;

- In the following:
  - assignment of the SLS solver is called α
  - (partial) assignment of iUP is called β

General idea for combining SLS and iUP

Enhancing a given SLS solver with iUP requires answers to the following questions:

1. **When** to perform iUP during the SLS solvers search?
2. **How** is the result β of iUP used?
3. **What** variable selection heuristic VAR should iUP use?
4. **What** value selection heuristic VAL should iUP use?
5. **What** happens if iUP detects the empty clause?

General idea for combining SLS and iUP

1. **When** to perform iUP during the SLS solvers search?

- iUP is supposed to assist the SLS solver in its search
- A comparatively obvious situation in which the SLS solver could use assistance is when it cannot make any greedy flips
- The most simple answer to the "When" question would be to call for iUP instead of switching into random mode

Idea: Replace the random mode heuristic with a call to iUP.

General idea for combining SLS and iUP

2. **How** is the result β of iUP used?

- The goal of the SLS solver in random mode would be to escape the current "dead end" assignment α
- This is usually done by using a variable selection heuristic like the Sparrow heuristic
- The resulting assignment β from the call to iUP must now be used to fulfill this task

Idea: Compare α and β on all variables assigned in β. "Multi-flip" all variables in α that have a different assignment in β, i.e. all variables that iUP does not agree on.

General idea for combining SLS and iUP

3. **What** variable selection heuristic `VAR` should `iUP` use?

- Research on (double)-look-ahead solvers suggests the use of a recursive weighting heuristic
- An example would be the RW heuristic [SMBWMH2010,DAMF2010]
- RW provides you with an ordering θ_{RW} of the variables
- Intuitively, $\theta_{\text{RW}}(x_i) < \theta_{\text{RW}}(x_j)$ means variable x_i has a stronger impact on the formula than x_j when it is assigned
- A stronger variable impact results in more reduction in the formula
- More reduction yields more unit clauses sooner

Idea: `VAR` picks the first variable according to θ_{RW} that is not yet assigned in β.

General idea for combining SLS and iUP

4. **What** value selection heuristic VAL should iUP use?

- Once iUP decided for a variable to assign next, it must decide what value it wants to assign it to
- The use of β is to help the SLS escape from a dead end α
- β must somehow be related to the dead end assignment α
- A straight forward idea is to have iUP try to reconstruct the SLS solvers assignment α

Idea: VAL performs $\beta(x_i) = \alpha(x_i)$.

General idea for combining SLS and iUP

4. **What** value selection heuristic `VAL` should `iUP` use?

- Once `iUP` decided for a variable to assign next, it must decide what value it wants to assign it to
- The use of β is to help the SLS escape from a dead end α
- β must somehow be related to the dead end assignment α
- A straight forward idea is to have `iUP` try to reconstruct the SLS solvers assignment α

Idea: `VAL` performs $\beta(x_i) = \alpha(x_i)$.

- The only way that α and β do not agree on a variable is because of unit propagation.

General idea for combining SLS and iUP

5. **What** happens if `iUP` detects the empty clause?

   - As soon as the empty clause emerges, all further propagations/assignments are meaningless

Idea: `iUP` stops as soon as the empty clause emerges (`conflictStopFlag` := true).

Putting it all together

```
SLSUP(k-CNF F, timeout t)
    Randomly initialize α;
    Compute θ_RW;
    repeat
        if (F(α) = 1) then output satisfying assignment; terminate;
        if ∃α′ ∈ Neighborhood(α): f(α′) ≤ f(α)
        then //greedy mode
            α := α′; //flip the variable that gives the best improvement
        else //random mode
            α :=iUP(F,θ_RW,α,true); //partially override α with β
    until timeout t is reached;
    output unknown;
```

Putting it all together

Does this work?

Putting it all together

Does this work?

- **No!**

Why?

Putting it all together

Does this work?

- **No!**

Why?

- SLS encounters a dead end in about every third flip (3-SAT, determined empirically)
- The amount of variables `iUP` propagates is about 42% before it discovers the empty clause
- We use a static variable ordering and two almost identical α
- The chance to get two different results from consecutive `iUP` calls is practically non-existent
- Calling `iUP` that often is a waste of computational time

Putting it all together

Does this work?

- **No!**

Why?

- SLS encounters a dead end in about every third flip (3-SAT, determined empirically)
- The amount of variables `iUP` propagates is about 42% before it discovers the empty clause
- We use a static variable ordering and two almost identical α
- The chance to get two different results from consecutive `iUP` calls is practically non-existent
- Calling `iUP` that often is a waste of computational time
- Solution: Call `iUP` less often.

Cool-down periods

- Straight forward approach for calling `iUP` less often:
  - manually increase the amount of flips that have to pass between to consecutive calls of `iUP`
  - these intervals of flips in between two `iUP` calls are called cool-down periods $\mathfrak{c}$

Cool-down periods

- Straight forward approach for calling iUP less often:
  - manually increase the amount of flips that have to pass between to consecutive calls of iUP
  - these intervals of flips in between two iUP calls are called cool-down periods $\mathfrak{c}$
- How long should these cool-down periods be?

Cool-down periods

- Straight forward approach for calling `iUP` less often:
  - manually increase the amount of flips that have to pass between to consecutive calls of `iUP`
  - these intervals of flips in between two `iUP` calls are called cool-down periods $\mathfrak{c}$
- How long should these cool-down periods be?
  - Fixed values will not work
  - Pick cool-down periods randomly from a given interval

Cool-down periods

- Straight forward approach for calling `iUP` less often:
  - manually increase the amount of flips that have to pass between to consecutive calls of `iUP`
  - these intervals of flips in between two `iUP` calls are called cool-down periods $\mathfrak{c}$
- How long should these cool-down periods be?
  - Fixed values will not work
  - Pick cool-down periods randomly from a given interval
    - What does the interval look like? $[\mathfrak{c}_{\min}, \mathfrak{c}_{\max}]$
    - What distribution is used for picking values from that interval?

Cauchy distribution

Empirical tests indicate, that the cool-down periods should be picked from the interval $[0, 2.7n]$. But what about the distribution?

Cauchy distribution

Empirical tests indicate, that the cool-down periods should be picked from the interval $[0, 2.7n]$. But what about the distribution?
The Cauchy distribution is defined by its 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

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

with $\omega := 2n$ and $\gamma = 1500$.

Cauchy distribution

Empirical tests indicate, that the cool-down periods should be picked from the interval $[0, 2.7n]$. But what about the distribution?
The Cauchy distribution is defined by its 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

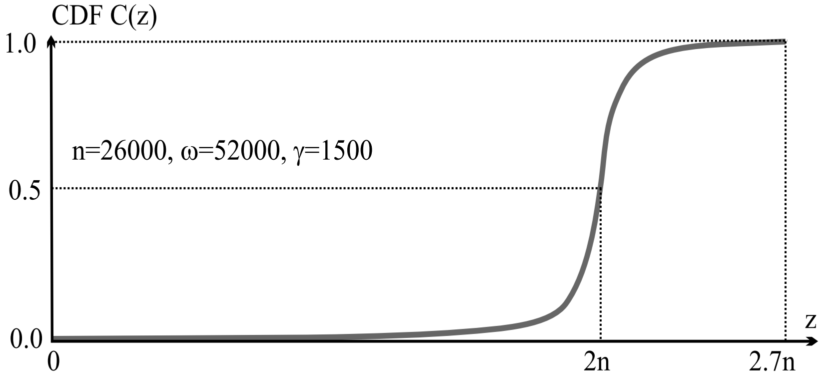
$$C : \mathbb{R} \mapsto \mathbb{R},\ C(z) = P(Z < z) = \frac{1}{2} + \frac{1}{\pi} \cdot \arctan\left(\frac{z - \omega}{\gamma}\right).$$

with $\omega := 2n$ and $\gamma = 1500$.
The general idea is: after every call to iUP

- pick $a \in [0, 1)$ uniformly at random
- compute $\mathfrak{c} = \lfloor \min\{z | C(z) \geq a\} \rfloor$

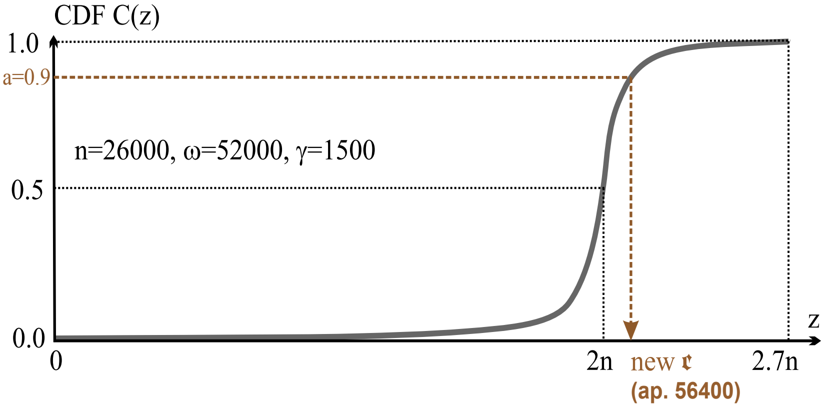
Cauchy distribution

Given a formula F with 26000 variables.



Cauchy distribution

Given a formula F with 26000 variables.



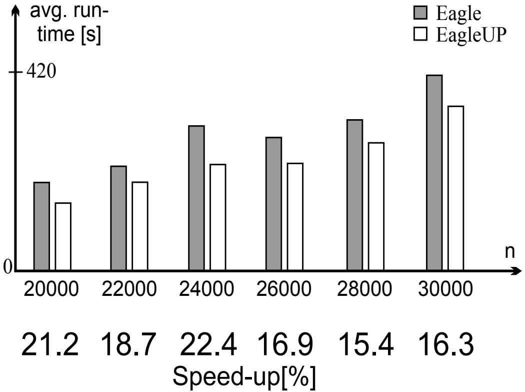
Again, putting it all together

```
EagleUP(k-CNF F, timeout t)
    Randomly initialize α;
    Compute θ_RW;
    Compute Cauchy CDF C(z), z ∈ [0, 2.7n], ω := 2n, γ := 1500, 𝔠 = ω;
    flips:=0; lastIUPcall:=0;
    repeat
        if (F(α) = 1) output satisfying assignment; terminate;
        if ∃α′ ∈ Neighborhood(α): f(α′) ≤ f(α)
        then //greedy mode
             α := α′; flips++;
        else //random mode
             if flips > lastIUPcall + 𝔠
             then //do iUP
                  α :=iUP(F,θ_RW,α,true); //partially override α with β
                  lastIUPcall=flips;
                  randomly pick a ∈ [0, 1) and set 𝔠 := min{z|C(Z) ≥ a};
             else //do Sparrow
                  use Sparrow heuristic to flip a variable; flips++;
    until timeout t is reached;
    output unknown;
```

Results of the Empirical Study

This part of the empirical study consists of 600 3-SAT formulas

- of sizes 20000 variables to 30000 variables (100 each, 50 runs each)
- with a ratio of 4.2



- Check `http://edacc2.informatik.uni-ulm.de/EDACC3/index`

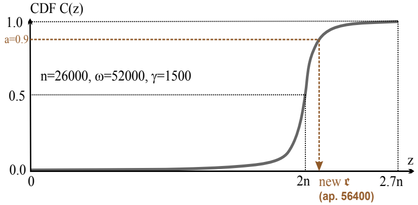
Conclusions and Future Work

Conclusions:

- We provided a scheme to combine SLS and UP to gain speed-ups on random 3-SAT formulas
- The usage of cool-down periods is of vital importance

Future Work:

- Why does the Cauchy distribution work? Is there any other Distribution that gives better results?
- Why is the possibility to have short/long cool-down periods so important?



Thanks

Thank you for your attention!

Questions?

Empirical study

Part A: 600 random 3-CNF formulas, 20000 . . . 30000 var., ratio 4.2

Part B: 1300 random 3-CNF formulas, 26000 var., ratios 4.14 . . . 4.24

Part A Solver	**succ. rate** [%]	**avg. run time** [s]	**avg. std. dev.** [s]	**speed up** [%]	**succ. rate** [%]	**avg. run time** [s]	**avg. std. dev.** [s]	**speed up** [%]	**succ. rate** [%]	**avg. run time** [s]	**avg. std. dev.** [s]	**speed up** [%]
	v20,000, r4.2				v22,000, r4.2				v24,000, r4.2			
TNM	77.90	708.09	389.61		68.34	899.15	434.65		58.00	899.64	401.62	
Eagle	99.70	164.71	138.51	76.7	99.70	209.47	173.56	76.7	98.42	279.40	213.44	68.9
EagleUP	99.72	129.76	97.81	21.2	99.96	170.13	129.28	18.7	99.28	216.64	155.78	22.4
	v26,000, r4.2				v28,000, r4.2				v30,000, r4.2			
TNM	49.90	1017.95	374.88		47.86	1062.19	383.74		30.32	1192.53	314.95	
Eagle	97.64	297.37	229.84	70.7	97.70	318.93	234.06	69.9	95.82	443.45	310.29	62.8
EagleUP	98.18	247.07	185.92	16.9	98.76	269.73	190.01	15.4	97.94	371.05	261.43	16.3

Part B Solver	**avg. run time** [s]	**speed up** [%]	**avg. run time** [s]	**speed up** [%]	**avg. run time** [s]	**speed up** [%]	**avg. run time** [s]	**speed up** [%]	**avg. run time** [s]	**speed up** [%]	**avg. run time** [s]	**speed up** [%]
	r4.14		r4.16		r4.18		r4.20		r4.22		r4.24*	
Eagle	9.36	6.5	29.85	11.1	94.97	16.6	297.37	16.9	763.49	6.7	1107.28	5.7
EagleUP	8.75		26.53		79.24		247.07		712.04		1043.27	

Results for Part A and B suggest superiority of `EagleUP` over `Eagle`.

Bibliography

- ABAF2010 Balint, A., Fröhlich, A.: Improving Stochastic Local Search for SAT with a New Probability Distribution. In SAT'10, LNCS 6175:10-16. Springer 2010.
- OGMH2011 Gableske, O., Heule, M.J.H.: Solving Random 3-SAT using SLS with Unit Propagation. PoS Workshop at SAT'11, 2011.
- SMBWHM2010 Mijnders, S., De Wilde, B., Heule, M.J.H.: Symbiosis of search and heuristics for random 3-SAT. In LaSh'10, 2010.
- DAMF2010 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.