

# Stabilization of parameter based look-ahead Sat solvers

Hans van Maaren and Tom van der Velden

*Faculty of Electrical Engineering, Mathematics and Computer Science, Delft  
University of Technology, P.O. Box 5031, 2600 GA Delft, The Netherlands.  
email: h.vanmaaren@ewi.tudelft.nl*

February 5, 2004

**Abstract.** A so called 3-weight based look-ahead technique is proposed to stabilize the performance of look-ahead Sat solving methods. The latter methods consist of a *pre-select* procedure which selects candidate branching variables according to a certain criterion. After performing iterative unit propagation using these variables the resulting CNF formulas are evaluated (with respect to another measure, a so called *dif-measure*) in order to determine the actual branching variable from the selected ones. The interplay between the pre-select heuristic and dif evaluation largely determines the performance of the Sat solving algorithms designed that way. Many pre-select heuristic procedures in turn use weight-parameters to distinguish between the impact of shorter clauses and longer ones. Sometimes these weights are explicitly used, stated and optimised to increase performance on a specific problem class, but also, sometimes they are implicitly invoked in a manner hard to analyse. The 3-weight approach now selects a weight-parameter and suggests to repeat the pre-select heuristic with respect to three different values and to use the dif evaluation measure not only to select the actual branching variable, but also to discriminate between the heuristic strength of these three values and to indicate suitable updates to be used in child nodes of the emerging search tree. We present experimental evidence for the stabilizing effect on performance while concluding simultaneously that the resulting overhead in the unit propagation phase is marginal.

**Keywords:** Satisfiability, branching, look-ahead, clause weighting

## 1. Introduction

An attempt to split nowadays state of the art Sat solving techniques [2] into two main stream architectures leads to the distinction between conflict-based solvers and those based on a look-ahead-procedure. Some examples of the first class are Chaff [15], Berkmin [8], Jerusat [1] and Satzoo, generally performing well on industrial problems and some of the hand made benchmarks. Examples of the latter are OKsolver [11], Satz [12], kCNFs [5] [6] and March, which dominate categories consisting of random problems (as well as some of the hand made). In this paper we focus on look-ahead solving techniques and particularly deal with those which rely on the use of weights in their pre-select procedure. Most of the time these parameters are used to distinguish between the impact of shorter clauses and longer ones in building the heuristic for this pre-select phase, the goal of the latter being to dis-

criminate between variables regarding their possible performance as branching variable. Look-ahead based solvers then actually perform an iterative unit propagation procedure using all selected variables as candidate branching variables separately. After this phase, all resulting CNF formulas at the candidate child nodes are evaluated according to a so called dif criterion in order to measure the actual progress in enrolling the formula at hand. The latter criterion then makes the final choice which of the pre-selected candidates actually will be chosen as branching variable. In many cases, the role of the parameters used is not very transparent and their values seem to have been optimised to perform well on a specific problem class, which sometimes results in disappointing performance on other classes. In general, the interplay between the criteria pre-select and dif is crucial for the performance of the resulting algorithm. For instance, possible dramatic effects of a poorly performing pre-select procedure might be compensated by an effective dif measure, while unfavourable effects of a poorly performing dif measure can only be compensated by an effective pre-select heuristic.

In this paper we want to propose a method which may stabilize the performance of look-ahead solvers in cases where the pre-select procedure indeed uses a parameter, specifically in case when dealing with a clause weighting devise (as March, Satz, kCNFs). Thus our concern is setting weights such to achieve reasonable performance over a variety of problems, rather than to optimise performance on a certain class.

The method consists of actually repeating the pre-select phase for three initial values of a selected weight parameter (resulting in a pre-selected set of branching candidates three times larger in worst case) and then using the dif measure not only to determine the actual branching variable but also to indicate updates of the three values used. The evidence for stabilizing effects we present is of experimental nature. We investigate the March pre-select procedure because it transparently uses only one such weight, which makes the analysis of the results easier and independent of the presence of other parameters. March uses a 3-Sat translator in the pre-processing phase which has as a result that only the relevance of 2-literal clauses against 3-literal clause is used in the pre-select heuristic. This allows us to investigate performance with respect to the influence of one parameter only. We show that over a variety of problems, and over a variety of dif-criteria, this 3-weight approach indeed has a stabilizing effect not only with respect to node counts but that the expected overhead coming from the look-ahead phase is only marginal.

The authors are aware of the fact that the presented material in fact only demonstrates that a 3-weight approach has benefits when incorporating it into specific solver configurations. However, they feel that the method proposed expresses a generic idea which might increase overall performance of parameter-based pre-select criteria used in other configurations as well. (For instance, a similar construction is possible in case one wants to invoke different pre-select heuristics by converting them into one, by using weights to combine them.)

Below we first recapitulate some preliminary terminology and present the core architecture of look-ahead solving in more detail. We will give some examples of pre-select and dif criteria commonly used. Then we briefly recall the MAR (Maximal Approximated Radius) pre-selection heuristic used in the early versions of the March solvers. Next, we add a discussion on establishing plausible initial values for the weight parameter involved. We proceed with explaining the 3-weight approach and present a collection of benchmark classes which we use as test cases and motivate why this particular selection is considered representative for our purposes. From the pre-select and dif-measure combinations we construct four protoSolver solvers for which we show that a stabilizing effect of the 3-weight approach is clearly observable in all cases.

## 2. Preliminaries

A CNF (Conjunctive Normal Form) formula is a conjunction of clauses, each clause being a disjunction of literals. A literal is a propositional variable or its negation.

Clauses are assumed not to have multiple occurrences of variables. The length of a clause is the number of literals in it. If a clause has length  $k$  it is called a  $k$ -literal clause or  $k$ -clause for short. A CNF consisting of clauses of length 3 or less is called a 3-CNF. The SAT discipline deals with methods which provide a solution (that is a 0,1 assignment to the variables involved which satisfies all clauses) to a given CNF or to show that such a solution does not exist. The SAT problem was shown to be NP-complete [3]. As such, it plays a central role in complexity theory, but also has numerous of applications in disciplines as planning, scheduling, model checking and protocol- and circuit verification and design.

Any CNF is satisfiability equivalent to a 3-CNF, meaning that either both CNFs are satisfiable or both are unsatisfiable. The 3-SAT translator we use in this paper is explained by the following example. Consider the 7-clause:

$$x_1 \vee \neg x_2 \vee x_3 \vee x_4 \vee x_5 \vee \neg x_6 \vee x_7$$

This clause is satisfiability equivalent to following set of clauses:

$$\begin{aligned}
 & x_1 \vee \neg x_2 \vee d_1 \\
 & x_3 \vee x_4 \vee d_2 \\
 & x_5 \vee \neg x_6 \vee d_3 \\
 & x_7 \vee d_4 \\
 & \neg d_1 \vee \neg d_2 \vee d_5 \\
 & \neg d_3 \vee \neg d_4 \vee d_6 \\
 & \neg d_5 \vee \neg d_6
 \end{aligned}$$

Notice that the translator above introduces variables not present in the original CNF. Branching on these dummy variables is not favourable in general and therefore we will incorporate in any of the pre-select procedures below the rule that these variables cannot be pre-selected.

An *iterative unit propagation* (IUP) caused by setting a certain variable on a certain Boolean value is the procedure which sets emerging 1-literal clauses to their obvious value as long as such clauses remain.

For a CNF formula  $F$  and a variable  $x$  the formula  $F(x = b)$  is the resulting CNF after applying an iterative unit propagation with  $x = b$  ( $b$  Boolean).

Given a literal  $l$  we denote the number of occurrences of  $l$  in  $k$ -clauses by  $\#occ_k(l)$  whenever the CNF formula involved gives no reason for confusion.

### 3. Core architecture of Look-ahead solvers

The architecture of a look-ahead solver (originated in [7]) consists of the following branch and backtrack procedure:

- input a CNF  $F$
- apply a procedure *pre-select*( $F$ ) to the variables in  $F$  (this procedure thus selects a subset of variables)
- apply an iterative unit propagation procedure with respect to all selected variables (and their negations) *separately*
- apply a *dif* criterion to all resulted CNFs (a *dif* being a device assigning a number to the pair  $(F, F(x = b))$ )
- combine for any selected variable  $x$  the two numbers  $dif(F, F(x = 1))$  and  $dif(F, F(x = 0))$  into one single number *mix-dif*( $F, x$ )
- from the pre-selected variables choose one with largest *mix-dif* as actual branching variable

- if  $x$  is selected apply a device  $get\text{-}direction(F(x = 1), F(x = 0))$  (thus  $get\text{-}direction$  selects one from the two resulting CNFs)
- go into depth-first recursion, choosing the  $get\text{-}direction$  branch first

In the above it is assumed that stopping criteria and backtrack decisions are incorporated in the usual fashion.

Before going into the details concerning the devices  $pre\text{-}select$ ,  $dif$ ,  $mix\text{-}dif$  and  $get\text{-}direction$  we first emphasize that state of the art look-ahead solvers have additional features incorporated (as various pre-processing options, local learning, binary clause reasoning, autarky reasoning). In our experiments none of these additional features is used in solving the formulas in order to study the influence of the weight parameter in a manner independently from other influences.

### 3.1. PRE-SELECT

In the above, the  $pre\text{-}select$  criterion aims to present a set of favourable branching variables ([16] gives an overview of branching heuristics used). Pre-selection is usually based on counting number of occurrences of variables in clauses, eventually regarding clause lengths as well. Variables ranked high according to such a counting device are pre-selected (for instance a certain percentage of highest ranked variables). An example is the MAR (Maximal Approximated Radius) selection (the one used in the early versions of March, also used in this paper and explained later). A solver might eventually pre-select all variables (considered to be expensive for large size formulas) and rely on the quality of the  $dif$  evaluation (OKsolver). The other extreme is to select just one (highest ranked), as was done in the early SAT solvers. In the latter case, obviously there is no need to invoke a  $dif$  criterion. Thus, generally,  $pre\text{-}select$  tries to predict a certain speed in enrolling the formula by choosing a specific variable for the IUP procedure, thereby using relatively cheap counting criteria. Some other examples are:

1. Select each variable which occurs at least  $k$ -times in 2-clauses (but such that both positive and negative occurrences are present). This criterion is used in kCNFs and Satz, although with a different use of  $k$ .
2. For any literal  $x$  let

$$\text{pres}(x) = w \cdot \#occ_2(\neg x) + \#occ_3(\neg x) + \sum_{\neg x \vee y \in F} \#occ_3(\neg y) \quad (1)$$

and set

$$\text{Pres}(x) = \text{pres}(x) \cdot \text{pres}(\neg x). \quad (2)$$

Then select a fixed percentage with highest Pres value. This criterion is used in the later March versions.

### 3.2. DIF

A *dif* criterion actually investigates the effect of setting a pre-selected variable on a specific Boolean value by truly performing the IUP procedure involved. Although one might think of various criteria (how many variables are eliminated, how many clauses are satisfied, how many clauses are shortened, etc.) to the authors knowledge only three devices are commonly used:

1. Count the number of newly created 2-clauses (eventually incorporating also those which are temporarily present during the IUP procedure), used in Satz and the older versions of March.

2. Define

$$dif(x) = a \cdot \#occ_2(\neg x) + \#occ_3(\neg x) \quad (3)$$

and

$$dif = \sum_{x \vee y} (dif(x) \cdot dif(y))^b \quad (4)$$

(where the summation is over the newly created 2-clauses  $x \vee y$ ), used in kCNFS and later versions of March, although with different settings for  $a$  and  $b$ . The values of parameters  $a$  and  $b$  are obtained by optimising the solvers performance on random CNF problems.

3. Count a  $k$ -weighted sum over newly created  $k$ -clauses (OKsolver).

More educated *dif* criteria which investigate more properties of the formula-enrolling process seem not popular, probably because of the fact that the *dif* criterion must be applied many times in a look-ahead environment, so it is considered important to keep it simple and straightforwardly computable.

### 3.3. MIX-DIF

This device tries to balance between the distinctly measured progressions  $L = dif(F, F(x = 1))$  and  $R = dif(F, F(x = 0))$ . If one of both is extremely high but the other is low it is questionable whether such an  $x$  should be considered as favourable. Therefore, just averaging those two numbers seems to be never used as a *mix-dif*. Although many averaging functions exist (as Cobb-Douglas functions and CES functions used in econometric models) which try to balance with respect to various

features, the arithmetic mean (hence just the product of both numbers) seems to be a default choice in look-ahead based solvers:

$$\text{Mix-dif}(L, R) = c \cdot LR + L + R, \quad (5)$$

where  $c$  is taken large such that the part  $L + R$  acts merely as a tie-breaking device. The OKsolver uses a different tie-breaking.

### 3.4. GET-DIRECTION

From both branches  $F(x = 1)$  and  $F(x = 0)$  one has to be selected for first investigation. If the input CNF is satisfiable an effective *get-direction* heuristic is of course extremely important. We give some examples of *get-direction* decisions used in various settings:

1. If variable  $x$  is selected investigate  $w(\#\text{occ}_2(x) - \#\text{occ}_2(\neg x)) + (\#\text{occ}_3(x) - \#\text{occ}_3(\neg x))$ . If the latter is positive first enter branch  $F(x = 1)$ , else enter branch  $F(x = 0)$ . This device is used in kCNFS (with  $w$  large) and in the early versions of March ( $w = 10$ ).
2. If variable  $x$  is selected first enter that branch which has the *smaller* dif value (used in late versions of March).
3. If variable  $x$  is selected investigate

$$-\sum_k (\#k\text{-clauses in } F(x = 1)) \cdot \ln(1 - 2^{-k}) \quad (6)$$

and the similar quantity for  $F(x = 0)$ . First enter the branch with smallest of the two numbers. This device is used in the OKsolver. In fact this device enters that branch first which has a larger probability of being satisfied by a random assignment.

From the above we selected the MAR *pre-select*, the *dif* measures (1) and (2) with parameters  $a = 1$  and  $b = 1$ , and the two *get-direction* devices (1) and (2) to construct four prototype solvers to be used in our experiments. The weight  $w$  in *get-direction* (1) will in fact be the same parameter as used by the MAR rule.

### 3.5. MAR RULE

The Maximal Approximated Radius rule originated in [16]. This rule is based on geometric observations: to a CNF an ellipsoid region in Euclidean Space is associated which is shown to contain all satisfying assignments. For each variable  $x$  it can be computed what the range of this region is in the direction of the  $x$ -axis. If this range is small,

the variable involved is considered to have large impact on the formula. Actual computation of this range is expensive however, since it involves matrix inversions. Therefore, an approximation of this range is presented in [16] which is relatively effectively computable, but still is considered expensive with respect to other existing rules. The MAR-value of a variable present in a 3-CNF reduces under this approximation to

$$Mar(x_i) = Q_{ii} - \sum_{i \neq j} \frac{Q_{ij}^2}{Q_{jj}} \quad (7)$$

where

$$Q_{ij} = w \cdot Q_{ij,2} + Q_{ij,3} \quad (8)$$

$$Q_{ij,2}^+ = \text{number of 2-clauses where } x_i \text{ and } x_j \text{ appear either} \\ \text{both positive or both negated} \quad (9)$$

$$Q_{ij,2}^- = \text{number of 2-clauses containing either } x_i \text{ and } \neg x_j \\ \text{or } \neg x_i \text{ and } x_j \quad (10)$$

$$Q_{ij,2} = Q_{ij,2}^+ - Q_{ij,2}^- \quad (11)$$

$$\text{similarly } Q_{ij,3} \text{ is defined} \quad (12)$$

In the above, the parameter  $w$  distinguishes between clauses of length 2 and length 3. Clauses of length 2 are considered to have a larger impact on the formula at hand.

The value of the parameter also influences the shape of the ellipsoid region mentioned above. In [14] it is shown that, restricted to random 3-CNF problems, optimal value of  $w$  (with respect to MAR using solver performance) coincides with that value minimizing the volume of the region (in fact  $w_{opt} \approx 11$ ). This observation will be used by us to establish appropriate initial weights in our 3-weight based approach.

#### 4. Initial weight problem

Although our 3-weight approach will be illustrated by its effects using the MAR *pre-select* criterion the underlying idea might be considered generic for those situations where the effect of setting a parameter to a certain value can be evaluated through an independent *dif* measure. The initial weight problem however is typically associated to those cases where this parameter is used to weight clauses of various lengths. We start with the observation in [14] that an appropriate initial value of the weight  $w$  might be obtained by minimizing the volume of the ellipsoid as a function of  $w$ . Again, this minimization process is rather expensive and we have chosen to deal with a rather simplified form of it. Instead

of minimizing the exact expression for the volume of an ellipsoid region (as a function of one variable  $w$ ) we simplified this process by roughly estimating eigenvalues involved, finally leading to the following function of  $w$ :

$$Vol(w) \approx (n + 1) \log(3s_3 + (n + 1)\|q\|_2) + n \log\left(\frac{w2s_2 + 3s_3}{n + 1} + \|q\|_2\right) \quad (13)$$

where

$$n = \# \text{ variables in } F \quad (14)$$

$$q_i = w(\#\text{occ}_2(x_i) - \#\text{occ}_2(\neg x_i)) + (\#\text{occ}_3(x_i) - \#\text{occ}_3(\neg x_i)) \quad (15)$$

$\|\cdot\|_2$  is the Euclidean norm,  $s_2$  stands for the number of 2-clauses and  $s_3$  for the number of 3-clauses.

We use a simple Newton-Raphson iteration of the above to obtain our appropriate initial weight, only at the top node of the emerging search tree (or the first level where 2-clauses appear to be present). The outcome of our experiments clearly indicates that, without assuming any knowledge of the input CNF, the initial weight obtained such is a suitable candidate.

## 5. The 3-weight approach

The method proposed in this section is applicable in cases where the pre-select procedure has a built in parameter  $w$ . We use the notation  $pre-select(w)$ . Examples are already provided in the above discussion on widely used pre-select criteria. But there are other possibilities to construct such procedures. Imagine three different *pre-select* criteria, denoted by  $pre-select_1$ ,  $pre-select_2$  and  $pre-select_3$  then the following criteria are pre-select heuristics depending on a parameter:

- $pre-select(w) = pre-select_1 + w pre-select_2$
- $pre-select(w) = 1/w \cdot pre-select_1 + pre-select_2 + w \cdot pre-select_3$

In fact, one might use other averaging devices to obtain similarly defined procedures.

The pre-select procedures are used in look-ahead solvers in the following way: for a fraction of the best performing variables under the pre-select used both IUP procedures are entered. This fraction can be taken fixed (for instance 10%) throughout the emerging search tree, but it might also vary, depending on the depth. The latter distinction is not relevant for our proposed method however. In order to make the

discussion below more transparent we assume that a fixed percentage (let us say 10%) is considered.

Let us start with three different parameter values with small, medium and large values:  $w_{s0} < w_{m0} < w_{l0}$ .

We actually investigate the ranking on the variables under the three different pre-select criteria  $\text{pre-select}(w_{s0})$ ,  $\text{pre-select}(w_{m0})$  and  $\text{pre-select}(w_{l0})$ . We proceed with selecting the best performing 10% for all three criteria and enter the IUP procedures for all selected variables in total, where we take care for the fact that a variable enters IUP only once (it might very well be that a specific variable is selected more than once under the three different criteria!).

Hence, in the above, in worst case 30% of the variables enters IUP. The other extreme would be that all three criteria select the same 10%. (*Our experiments indicate that with this 10% fraction, for the configurations and benchmarks studied typically some 12% to 15% actually enters IUP.*)

The next step consists of applying the mix-dif procedure involved. Since our mix-dif measure assigns a number to all selected variables we may use this information to let mix-dif assign a number to each of  $w_{s0}$ ,  $w_{m0}$ ,  $w_{l0}$ , using the following aggregation: the performance of  $w_i$  is the average over the mix-dif numbers obtained by applying IUP to the best 10% under criterion  $\text{pre-select}(w_i)$ . Now we are facing two possible strategies:

*Strategy 1* Suppose mix-dif finally selected a variable for branching. This variable comes from at least one of the three different rankings. We declare the corresponding weights to be the best performing of the three weights considered. If  $w_{m0}$  is among them, we enter our next node with the same three weights (conservative tie breaking). If not, we choose the largest weight from the remaining possibilities (tie breaking in favour of the larger one). If this is  $w_{s0}$  we proceed with choosing a  $w$  with  $w < w_{s0}$  and update the parameters  $w_{s1} := w$ ,  $w_{m1} := w_{s0}$  and  $w_{l1} := w_{m0}$  for the next node in the emerging search tree. In case  $w_{l0}$  turned out to won the battle we choose  $w > w_{l0}$  and update accordingly  $w_{s1} := w_{m0}$ ,  $w_{m1} := w_{l0}$  and  $w_{l1} := w$ .

*Strategy 2* The weight with the highest average mix-dif performance is declared to be the winner. Tie breaking (although less frequently necessary) is done as in strategy 1, as well as updating the three values for the weight to be used in the next node.

It is clear that the performance of the resulting look-ahead algorithm will be rather sensitive with respect to the dif measure used, as well as to the selected strategy and the weight updating possibilities.

What we want to advocate is that regardless of what dif and strategy is chosen it is better (in the sense of stabilizing overall performance) to update weights dynamically than to stick with one specific value. Moreover, we feel that in updating weights in a manner as above it is more important to spread possible values over orders of magnitude rather than trying to obtain precise values for parameter  $w$ .

Therefore we have restricted ourselves in our experiments to the following context: An initial weight  $w_0$  is chosen and we start with  $w_{s0} = w_0/10$ ,  $w_{m0} = w_0$  and  $w_{l0} = w_0 \cdot 10$ . Updating will be done by using this updating factor 10 consistently over the search tree (recall that our experiments invoke the MAR selection procedure; if the parameter  $w$  models other features than clause weighting this factor 10 might be rather inappropriate).

Thus choosing for instance  $w_0 = 1$  as initiating parameter a 3-weight pattern as in Table I might occur in the emerging search tree:

Table I. Possible weight pattern

	$w_s$	$w_m$	$w_l$	$w_{winner}$
Node 0	0.1	1	10	10
Node 1	1	10	100	10
Node 2	1	10	100	100
Node 3	10	100	1000	10
Node 4	1	10	100	1
Node 5	0.1	1	10	0.1
Node 6	0.01	0.1	1	...

In fact, we observed throughout our experiments wild patterns as well as steady ones, depending on solver configuration and benchmark. Because our experiments clearly show stabilizing effects overall, this indicates that, regardless of what configuration chosen, our three weight approach enables the involved algorithm to learn and to benefit from this learning.

As a last remark we emphasize that one might think of various other and more educated possibilities of updating weights using the pre-select and dif criteria at hand. The above one however is a very simple one, only causing significant overhead calculations due to extra ordering variables calls and more IUP subroutines to perform.

The last mentioned overhead however is not really there, as we will clarify later.

## 6. Benchmark selection

In order to get an impression of the performance of our 3-weight approach we have chosen five well known (medium size) benchmark classes of rather different structure: random 3-CNF, random 3-colouring, Parity, Inductive Inferences and Quasi Groups. We motivate this selection below.

### 6.1. RANDOM 3-CNF

Look-ahead solvers generally perform well on this class and various existing parameters have been optimised to increase performance here ([5] and [14]). Therefore we believe it is instructive to show that even in this case improvement can be obtained. We generated (using OK generator [10]) 500 formulas with 250 variables at approximate threshold density  $d = 4.25$ . Initiating the weights accordingly the formula of section 4 is done in the second node of the search tree (because no 2-clauses are present in the first node).

### 6.2. RANDOM 3-COLOURING

A similar observation can be made that look-ahead solvers show efficient performance on this class. Again, 500 problems with 300 nodes and 690 edges (close to approximate node-edge threshold density 2.3) are generated. In CNF terms these formulae contain 900 variables, 300 3-clauses and 2970 2-clauses. We include the so called unicity clauses, which state that a node can be assigned at most one of the colours available (generally omitted in CNF translations of the graph colouring problem).

The latter inclusion makes the CNFs involved even more structural in appearance, while still dealing with the same combinatorial problem. DPLL techniques usually find the latter CNFs harder to deal with! We have chosen this benchmark class as an intermediate between random 3-CNF and structural problem classes. Also, for this class the use of a 3-SAT translator is not necessary yet. Since these formulas contain many 2-clauses at the beginning this benchmark class is rather suitable to test our initializing weight suggestion of section 4.

### 6.3. PARITY

The well known Parity benchmark problems [4] are known to be rather insensitive under branch selection heuristics (or better: show unpredictable performance). Here, some sort of equivalence reasoning is needed

to deal with them efficiently ([17] and [13]). Improvement of the performance of our prototype look-ahead algorithms (not including such reasoning) on this class under the 3-weight approach therefore we consider to be of convincing additional value. We selected the medium size Parity 16 formulas (10 formulas, all satisfiable). Also here, no 3-SAT translator is involved yet. The 3-clauses of the Parity formulas in fact are grouped by 4-tuples, each 4-tuple denoting a so called 3-equivalence. Because of this feature, existing clause weighting strategies generally seem to be less appropriate.

#### 6.4. INDUCTIVE INFERENCE

This benchmark set comes from [9] and contains 42 samples. The formulas are rather structured and contain long clauses. Thus, this class is suitable to investigate to what extent a 3-SAT translator influences the performance of our 3-weight approach. Moreover, local search methods generally perform well on this class, while lookahead solvers show some sort of an extreme behaviour: either a solution is obtained (almost) without branching (the set contains only satisfiable samples) or they get stuck at large depth. We have chosen this class to see whether the 3-weight approach shows some improvement in this respect.

#### 6.5. QUASI GROUPS

The quasi group benchmark set [18] contains satisfiable as well as unsatisfiable samples. The formulas stem from algebraic structures and hence the CNF translations are structured, contain longer clauses of various size and moreover, the set contains easy and hard samples. This benchmark class we consider to be extremely suitable to test the influence of the initialized weight strategy of section 4.

Although the authors are aware that the above selection still is arbitrarily, they believe that a certain aspect of a-selectness is present because of the considerations made in the above.

## 7. Computational results

As mentioned in section 3 four different prototype solvers are constructed from the various *Dif* and *get-direction* options:

- Solver 1: Combines Dif 1 and get-direction 1.
- Solver 2: Combines Dif 1 and get-direction 2.
- Solver 3: Combines Dif 2 and get-direction 1.

Solver 4: Combines Dif 2 and get-direction 2.

We emphasize again that it is not our intention to investigate which combinations are the best performing on specific benchmark types, but merely that all possible combinations will benefit from the 3-weight approach.

As an introduction to the manner we will expose our experimental results the reader is asked to consult table II which contains the results of solver 3 on the unsatisfiable random 3-CNF class.

Table II. Results for Solver 3 on unsatisfiable random 3-SAT instances.

Solver 3	Static weight		3-weight approach 1		3-weight approach 2		Static weight	
	$\mu$	%el	$\mu$	%el	$\mu$	%el	$\mu$	%el
$w = 0.01$	57719	10	16693	14	6856	16	19331	17
$w = 0.1$	47707	10	6009	17	5837	17	17254	17
$w = 1$	13305	10	5756	17	5780	17	8312	17
$w = 10$	6277	10	5753	17	5780	17	5570	17
$w = 100$	6761	10	5787	17	5778	17	5766	17
$w = 1000$	6936	10	6438	12	5820	16	5848	17
$w = \text{NR}$	6378	10	5792	16	5820	16	5605	17
Average	20726	10	7461	16	5953	17	9670	17
Deviation	22179		4078		399		5998	

Each row corresponds to a run with the initial weight stated, where the row NR presents the results of the run using the initial weight from section 4. The first column gives the average node count  $\mu$  for each initial weight, where 10% of the best ranked variables under the pre-select Mar (depending on the initial weight!) enters the IUP procedure. The initial weight indicated is maintained during the search, hence the heading "Static weight". Not surprisingly, the run with initial weight 10 shows optimal, in accordance with the results from [14]. The row Average then gives the average node counts over all runs considered. The row Deviation presents the corresponding deviation for this last mentioned averaging process.

The second column deals with our 3-weight approach using strategy 1 (see section 5) to update the weight parameter. Thus, in this case the initial weight will be subject to corrections as described earlier. Under subcolumn "%el" the average fraction of variables is presented which enters the IUP procedure: thus, for instance "17" means that from the 3 times 10% best selected variables under the 3 different weights there

is a considerable overlap, resulting in an average of 17% of the variables entering IUP.

Since a part of the improvement with respect to column 1 clearly is caused by a higher percentage of variables entering IUP, we repeated the "Static weight" experiment but now with about the same percentage entering the IUP phase as showed up under the 3-weight approach. The corresponding results are exposed in column 4.

Finally column 3 contains the results of our 3-weight approach using strategy 2 (see section 5) in updating the weights.

Evaluating the above figures demonstrates the feature we want to advocate: It is better to select variables for the IUP phase using more than one pre-select option *if the order of magnitude of an optimal static parameter is not known at forehand and that, under these circumstances, the NR initial weight is a good choice.*

Especially the 3-weight approach 2 shows the stabilizing effects convincingly: even when starting with a particularly bad initial weight (here 0.01) the method seems to correct for this failure rather quickly.

The appendix to this paper contains the tables for the other solvers and benchmarks selected. Since column 4 gives a more reliable comparison than column 1 we have deleted column 1 in these tables. Further, since over all experiments the performance of strategy 1 (column 2) is more or less comparable with the results in table II, that is, somewhere between the performances corresponding to columns 3 and 4, we have chosen to delete this column 2 as well.

Moreover, in the appendix, the tables "random 3-CNF" and "random 3-colouring" contain the results of aggregating the satisfiable and unsatisfiable samples.

When inspecting the tables from the appendix the above cursively stated conjectures are confirmed: either 3-weight approach 2 gives a more favourable average node count, or a considerably smaller deviation or (in many cases) both! The extend of the effect differs over the various benchmarks but the 3-weight approach is never substantially worse than the corresponding static configuration.

Some peculiarities can be observed however:

*-Parity family run  $w=0.01$*

The reader may notice that this particular run spoils the still overall better performance of the 3-weight approach. This is due to the fact that the static run got 16% of the fraction of variables entering IUP, because this fraction turned out to be the average fraction under the 3-weight approach, while the 3-weight approach for this particular initial

weight apparently suffered from almost complete overlap under the three pre-select heuristics, as is observable in the %el column.

The reader may convince himself that without this particular run the figures would alter in a more convincing benefit under the 3-weight approach. Please notice that the NR start is a safe one for this benchmark class.

*-Inductive inference runs*

These runs show no convincing overall improvements under the 3-weight approach but turn out to be very favourable with respect to the NR starts.

To give the reader some impression of the order of magnitude of the NR values we added table III

Table III. NR volume weight statistics for each benchmark set.

Benchmark set	Average	Deviation	Maximum	Minimum
random 3-SAT	30.8	2.9	45.2	21.9
3-Colouring	0.2	0.0	0.2	0.2
Parity 16	1.0	0.1	1.2	0.9
Inductive Inference	1.6	1.2	4.3	0.3
Quasi Groups	7.8	5.1	18.3	1.0

Notice that the NR initial values are non conformistic in that sense that they do not overall assume shorter clauses to be more important to investigate with respect to branching heuristics.

## 8. Conclusions and further research

We have gathered experimental evidence for the fact that in configurations of partial look-ahead based Sat solvers the pre-selection of variables which are allowed to enter the IUP phase might be better based on rankings with respect to different selection criteria, that is, in case one wants to have a solver with stable performance overall, rather than an optimized one for a particular goal. The authors are aware of the fact that they have only verified this conclusion for specific solver settings and specific benchmarks, but they feel that these settings and benchmarks chosen show enough diversity to support this claim.

Also, it must be stated here that these improvements concern node counts and that the overhead created by invoking more criteria (rankings) in some cases may spoil this gain, as we actually did observe

with our prototype solvers. However, in turn, we did not optimize these settings with respect to this feature because we preferred to have a transparent experimental environment. This latter means that we feel not comfortable to make a clear overall statement with respect to actual CPU time benefits of the 3-weight approach with respect to the experiments performed. These experiments however showed clearly that these benefits are there when comparing worst case static runs with the corresponding dynamic ones.

Also, in the particular settings using a MAR selection criterion (or a similar one using a weight parameter to model the difference of importance of clauses of different lengths) in building the pre-selection rankings, we have indicated a method which gives an indication of an appropriate initial weight.

In order to select favourable variables to enter the IUP phase we have chosen a fixed percentage to investigate. That this strategy might be improved is shown by figure 1. This figure shows the effect of ranking variables with respect to the MAR selection for a typical run of a graph colouring sample. As one can notice there is a clear observable jump in these values at nodes deeper in the search tree, while there is no such jump at the root node. Capturing these possible jumps seems an important issue for the pre-selection phase. Clearly, one wants to incorporate the high ranked variables in the IUP phase regardless whether this fraction is less or more than a chosen fixed percentage. The same figure however also shows (the values at the root node) the difficulties in recognizing and evaluating these jumps. Capturing essential jumps is an issue for future research.

## References

1. Alexander, N.: 2002, ‘Backtrack search algorithms for propositional satisfiability: Review and innovations’. Master’s thesis, Hebrew University of Jerusalem.
2. Berre, P. L. and L. Simon: 2003, ‘The Essentials of the SAT’03 competition’. In: *Springer-Verlag, LNCS 2919*, pp. 452–467.
3. Cook, S. A.: 1971, ‘The complexity of theorem-proving procedures’. In: *Proceedings of the Third IEEE Symposium on the Foundations of Computer Science*. pp. 151–158.
4. Crawford, J., M. Kearns, and R. Schapire: 1995, ‘The minimal disagreement parity problem as a hard satisfiability problem, Draft version’.
5. Dequen, G. and O. Dubois: 2002, ‘Renormalization as a function of clause lengths for solving random k-sat formulae’. In: *Proceedings of Fifth Inter-*

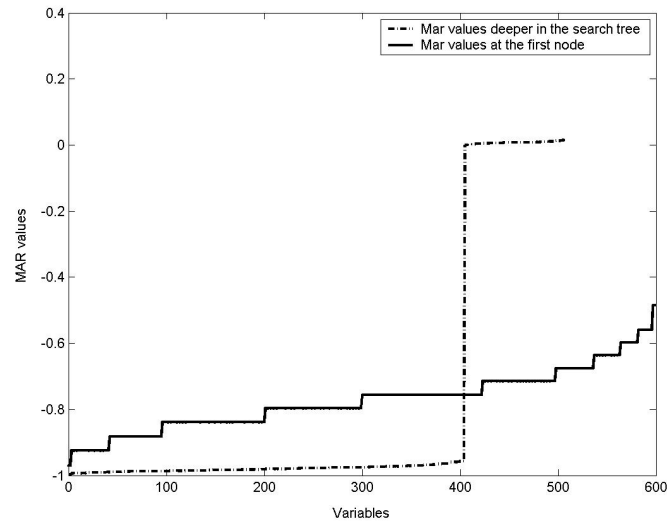


Figure 1. MAR values and the root and deeper in the search tree.

- national Symposium on Theory and Applications of Satisfiability Testing*. pp. 130–132.
6. Dubois, O. and G. Dequen: 2001, ‘A backbone-search heuristic for efficient solving of hard 3-sat formulae’. In: *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI’01)*. Seattle, Washington, USA.
  7. Freeman, J. W.: 1995, ‘Improvements to Propositional Satisfiability Search Algorithms’. Ph.D. thesis, Departement of computer and Information science, University of Pennsylvania, Philadelphia.
  8. Goldberg, E. and Y. Novikov: 2002, ‘BerkMin: A Fast and Robust SAT-Solver’. In: *Design, Automation, and Test in Europe (DATE ’02)*. pp. 142–149.
  9. Kamath, A., N. Karmarkar, K. Ramakrishnan, and M. Resende: 1992, ‘A continuous approach to inductive inference’. *Mathematical Programming* **57**, 215–238.
  10. Kullman, O.: 2002, ‘Towards an adaptive density based branching rule for SAT solvers, using a database for mixed random conjunctive normal forms built upon the Advanced Encryption Standard (AES)’. In: *Fifth International Symposium on Theory and Applications of Satisfiability Testing, Cincinnati 2002*, pp. 190-200.
  11. Kullmann, O.: 1998, ‘Heuristics for SAT algorithms: Searching for some foundations’. Submitted to *Annals of Mathematics and Artificial Intelligence*.
  12. Li, C. and Anbulagan: 1997, ‘Look-Ahead Versus Look-Back for Satisfiability Problems’. In: *Springer-Verlag, LNCS 1330*, pp. 342–356.
  13. Li, C.-M.: 2000, ‘Integrating Equivalency reasoning into Davis-Putnam procedure’. In: *Proceedings of the Seventeenth National Conference in Artificial Intelligence (AAAI’00)*. Austin, Texas, USA, pp. 291–296.

14. Maaren, H. V. and J. P. Warners: 2003, ‘Solving satisfiability problems using elliptic approximations - a note on volumes and weights’. *Annals of Mathematics and Artificial Intelligence* **37**, 273–283.
15. Moskewicz, M. W., C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik: 2001, ‘Chaff: Engineering an Efficient SAT Solver’. In: *Proceedings of the 38th Design Automation Conference (DAC’01)*.
16. Warners, J. and H. van Maaren: 2000, ‘Solving satisfiability problems using elliptic approximations: effective branching rules’. *Discrete Applied Mathematics* **107**, 241–259.
17. Warners, J. P. and H. Van-Maaren: 1998, ‘A two phase algorithm for solving a class of hard satisfiability problems’. *Operations Research letters* **23**, 81–88.
18. Zhang, J. and H. Zhang: 1995, ‘SEM: a System for Enumerating Models’. In: *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI’95)*. Montréal, Québec, Canada, pp. 298–303.

## Appendix

Table IV. Results for Solver 1 and 2 on 500 random 3-SAT instances, 250 variables with clause-variable density 4.27.

	Solver 1				Solver 2			
	3-weight		static		3-weight		static	
	$\mu$	%el	$\mu$	%el	$\mu$	%el	$\mu$	%el
$w = 0.01$	5599	17	16535	17	5534	17	16585	17
$w = 0.1$	5032	17	14733	17	4947	17	14794	17
$w = 1$	5038	17	6985	17	4923	17	7030	17
$w = 10$	5044	17	4885	17	4927	17	4768	17
$w = 100$	5048	17	5054	17	4934	17	4942	17
$w = 1000$	5069	17	5133	17	4937	17	5016	17
$w = \text{NR}$	5006	16	4916	17	4954	16	4804	17
Average	5119	17	8320	17	5022	17	8277	17
Deviation	212		5076		226		5151	

Table V. Results for Solver 3 and 4 on 500 random 3-SAT instances, 250 variables with clause-variable density 4.27.

	Solver 3				Solver 4			
	3-weight		static		3-weight		static	
	$\mu$	%el	$\mu$	%el	$\mu$	%el	$\mu$	%el
$w = 0.01$	4661	16	13111	17	4603	16	13328	17
$w = 0.1$	3986	17	11676	17	3958	17	11869	17
$w = 1$	4051	17	5605	17	3904	17	5654	17
$w = 10$	4054	17	3901	17	3904	17	3761	17
$w = 100$	4049	17	4048	17	3904	17	3893	17
$w = 1000$	4082	16	4106	17	3938	16	3948	17
$w = \text{NR}$	4006	16	3933	17	3938	16	3787	17
Average	4127	16	6626	17	4021	17	6606	17
Deviation	238		4006		257		4168	

Table VI. Results for Solver 1 and 2 on 500 random 3-colouring instances, 900 variables with edge-node density 2.3.

	Solver 1				Solver 2			
	3-weight		static		3-weight		static	
	$\mu$	%el	$\mu$	%el	$\mu$	%el	$\mu$	%el
$w = 0.01$	1115	13	1461	13	1113	13	1458	13
$w = 0.1$	1121	13	1089	13	1117	13	1087	13
$w = 1$	1124	13	961	13	1117	13	959	13
$w = 10$	958	10	954	13	956	10	954	13
$w = 100$	958	10	953	13	956	10	951	13
$w = 1000$	962	10	953	13	959	10	950	13
$w = \text{NR}$	998	16	983	13	995	16	1048	13
Average	1034	12	1051	13	1030	12	1048	13
Deviation	82		188		81		187	

Table VII. Results for Solver 3 and 4 on 500 random 3-colouring instances, 900 variables with edge-node density 2.3.

	Solver 3				Solver 4			
	3-weight		static		3-weight		static	
	$\mu$	%el	$\mu$	%el	$\mu$	%el	$\mu$	%el
$w = 0.01$	740	14	944	13	723	14	925	13
$w = 0.1$	736	14	752	13	722	14	748	13
$w = 1$	733	14	685	13	732	14	682	13
$w = 10$	730	13	663	13	731	13	663	13
$w = 100$	720	11	664	13	721	11	661	13
$w = 1000$	730	10	662	13	730	10	660	13
$w = \text{NR}$	713	15	692	13	698	15	691	13
Average	729	13	723	13	723	13	718	13
Deviation	9		102		12		96	

Table VIII. Results for Solver 1 and 2 on Parity 16 instances.

	Solver 1				Solver 2			
	3-weight		static		3-weight		static	
	$\mu$	%el	$\mu$	%el	$\mu$	%el	$\mu$	%el
$w = 0.01$	2612	10	1543	16	2432	10	1254	16
$w = 0.1$	942	16	1543	16	503	16	1254	16
$w = 1$	942	16	975	16	503	16	538	16
$w = 10$	912	16	902	16	534	16	542	16
$w = 100$	912	16	1343	16	534	16	1326	16
$w = 1000$	860	13	6453	16	942	12	6800	16
$w = \text{NR}$	940	16	975	16	503	15	538	16
Average	1160	15	1962	16	850	14	1750	16
Deviation	641		1999		716		2257	

Table IX. Results for Solver 3 and 4 on Parity 16 instances.

	Solver 3				Solver 4			
	3-weight		static		3-weight		static	
	$\mu$	%el	$\mu$	%el	$\mu$	%el	$\mu$	%el
$w = 0.01$	9950	11	7171	16	4087	11	3221	16
$w = 0.1$	3055	17	7158	16	2024	16	3316	16
$w = 1$	3055	17	3175	16	2024	16	2136	16
$w = 10$	3055	17	3096	16	2024	16	2049	16
$w = 100$	3113	17	2438	16	1966	16	1934	16
$w = 1000$	2473	12	5840	16	1851	12	5642	16
$w = \text{NR}$	3042	17	3151	16	2020	16	2187	16
Average	3963	15	4576	16	2285	15	2926	16
Deviation	2649		2072		797		1325	

Table X. Results for Solver 1 and 2 on Inductive Inference instances.

	Solver 1				Solver 2			
	3-weight		static		3-weight		static	
	$\mu$	%el	$\mu$	%el	$\mu$	%el	$\mu$	%el
$w = 0.01$	380	13	364	13	421	12	346	12
$w = 0.1$	372	13	303	13	421	13	352	12
$w = 1$	381	13	396	13	421	12	420	12
$w = 10$	363	13	398	13	406	12	380	12
$w = 100$	376	13	401	13	410	12	397	12
$w = 1000$	449	12	414	12	417	11	397	12
$w = \text{NR}$	315	13	227	13	381	13	402	12
Average	376	13	358	13	411	12	385	12
Deviation	39		69		14		27	

Table XI. Results for Solver 3 and 4 on Inductive Inference instances.

	Solver 3				Solver 4			
	3-weight		static		3-weight		static	
	$\mu$	%el	$\mu$	%el	$\mu$	%el	$\mu$	%el
$w = 0.01$	254	11	329	11	175	12	302	12
$w = 0.1$	245	11	283	11	177	12	269	12
$w = 1$	253	11	342	11	206	12	175	12
$w = 10$	281	11	388	11	198	12	138	12
$w = 100$	351	11	412	11	206	12	156	12
$w = 1000$	392	11	412	11	208	11	171	12
$w = \text{NR}$	249	11	243	11	175	12	184	12
Average	289	11	344	11	192	12	199	12
Deviation	59		65		16		62	

Table XII. Results for Solver 1 and 2 on Quasi Group instances.

	Solver 1				Solver 2			
	3-weight		static		3-weight		static	
	$\mu$	%el	$\mu$	%el	$\mu$	%el	$\mu$	%el
$w = 0.01$	943	15	1161	15	935	15	1061	15
$w = 0.1$	900	15	1255	15	892	15	1218	15
$w = 1$	865	15	1265	15	864	15	1265	15
$w = 10$	908	15	918	15	908	15	918	15
$w = 100$	848	15	822	15	852	15	825	15
$w = 1000$	836	15	735	15	840	15	805	15
$w = \text{NR}$	959	15	1034	15	958	15	1034	15
Average	894	15	1027	15	893	15	1018	15
Deviation	47		210		44		181	

Table XIII. Results for Solver 3 and 4 on Quasi Group instances.

	Solver 3				Solver 4			
	3-weight		static		3-weight		static	
	$\mu$	%el	$\mu$	%el	$\mu$	%el	$\mu$	%el
$w = 0.01$	1215	13	1251	13	1218	14	1230	13
$w = 0.1$	1162	13	1275	13	1165	13	1274	13
$w = 1$	1204	13	1315	13	1202	13	1315	13
$w = 10$	1127	13	974	13	1188	13	971	13
$w = 100$	1131	13	961	13	1130	13	961	13
$w = 1000$	1113	13	909	13	1112	13	908	13
$w = \text{NR}$	1117	13	1099	13	1177	13	1094	13
Average	1153	13	1112	13	1170	13	1108	13
Deviation	42		169		38		166	