Dynamic Scoring Functions with Variable Expressions: New SLS Methods for Solving SAT

Dave A. D. Tompkins and Holger H. Hoos

Department of Computer Science
University of British Columbia
{davet,hoos}@cs.ubc.ca

Abstract. We introduce a new conceptual model for representing and designing Stochastic Local Search (SLS) algorithms for the propositional satisfiability problem (SAT). Our model can be seen as a generalization of existing variable weighting, scoring and selection schemes; it is based upon the concept of Variable Expressions (VEs), which use properties of variables in dynamic scoring functions. Algorithms in our model are constructed from conceptually separated components: variable filters, scoring functions (VEs), variable selection mechanisms and algorithm controllers. To explore the potential of our model we introduce the Design Architecture for Variable Expressions (DAVE), a software framework that allows users to specify arbitrarily complex algorithms at runtime. Using DAVE, we can easily specify rich design spaces of SLS algorithms and subsequently explore these using an automated algorithm configuration tool. We demonstrate that by following this approach, we can achieve significant improvements over previous state-of-the-art SLS-based SAT solvers on software verification benchmark instances from the literature.

1 Introduction

The propositional satisfiability problem (SAT) is an important subject of study in many areas of computer science and is a prototypical $\mathcal{NP}$-complete problem. Among the best known methods currently available for solving certain types of SAT instances are Stochastic Local Search (SLS) procedures; these are typically incomplete, *i.e.*, they cannot determine with certainty that a given propositional formula is unsatisfiable, but they often find models of satisfiable formulae surprisingly effectively [9]. SLS algorithms for SAT typically start by randomly assigning to every variable appearing in a given formula a value of either true or false; then, in each subsequent *search step* a variable is *selected* to have its truth assignment *flipped* from true to false or vice versa. The method of selecting the variable to be flipped in each step is usually guided by a *scoring function* that minimizes the number of currently unsatisfied clauses.

In this work, we propose a new conceptual model for specifying SLS algorithms for SAT, and provide a software framework to aid in the development of new algorithms. Our model was developed to provide a clean conceptual separation between the scoring function(s) and the *Variable-Selection Mechanism (VSM)* of an algorithm. We introduce the concept of *Variable Expressions (VEs)* to generalize scoring functions; while VEs are ultimately used for variable selection, they can transcend the traditional notion of score. VEs are mathematical expressions that compute numerical values from one or more *properties* of a variable in combination with constants, operators and functions. The variable properties that can appear in VEs include well-known concepts from the literature, such as GSAT's score property [17] and the age property used by NOVELTY and WALKSAT/TABU [14]. A VE can be a simple property (*e.g.*, $\langle \mathsf{age} \rangle$) or any mathematical expression with one or more properties, such as $\langle \mathsf{score} + 3 \cdot \log(\mathsf{age}) \rangle$. Most existing SLS algorithms for SAT select variables based on scoring functions that correspond to a single, rather simplistic VE; in this paper we present evidence that potentially complex VEs can be very effective.

To explore the potential of our model, we introduce the *Design Architecture for Variable Expressions (DAVE)*, a software extension of our versatile UBCSAT architecture [18]. No programming is required to develop new algorithms in DAVE; the complete algorithm specification (including arbitrarily complex VEs) can be provided at run-time. We provided this flexibility in DAVE from the outset, with the goal of leveraging existing automated algorithm configuration tools (henceforth, *configurators*) such as PARAMILS [11]. With the combination of DAVE and a configurator, designers have an unprecedented amount of flexibility and power to help automate the design of new high-performance SLS algorithms and algorithm hybrids.

The remainder of the paper is structured as follows. In Section 2, we describe our experimental methodologies. In Section 3, we introduce more advanced VEs and demonstrate their efficacy. In Section 4, we present our general conceptual model and briefly discuss its implementation (DAVE). In Section 5, we introduce a new, highly parametric algorithm named VE-SAMPLER to demonstrate how DAVE facilitates the automated design of SLS algorithms. In Section 6, we discuss related work from the literature, and in Section 7, we summarize the contributions made in this work and outline directions for future research.

2 Experimental Details & Methodology

In the experiments presented throughout this study, we used the PARAMILS automated algorithm configurator by Hutter *et al.* [11] to optimize the parameter settings of various SLS-based SAT algorithms for performance on a particular instance set. To ensure that our results generalize to instances other than those used during the optimization process, we randomly split each set into two halves, a *training* set and a *test* set, where an optimal configuration is found by conducting experiments on the training set. Instances in the test set were only used for the final performance measurements presented in this paper.

In our experiments, we mostly focused on the CBMC software verification instance set generated, and used as a benchmark, by KhudaBukhsh *et al.* [12]. The instances were generated by a Bounded Model Checking (BMC) tool [4] and were pre-processed with SATELITE [5]. This set is interesting to us primarily because it has some of the structural properties of larger and more complicated software verification problems (that are still somewhat intractable for SLS solvers). For example, many of the complete solvers from the 2009 SAT Competition (such as PICOSAT [3]) can solve the hardest CBMC instance in less than one second, whereas well-known state-of-the-art SLS solvers from the competition such as ADAPTG^{2}WSAT and GNOVELTY$^+$ require over an hour to solve the same instance. At the same time, a significant number of the instances can be solved by SLS algorithms within a low enough time to allow for extensive experiments.

In Section 5 we also provide for the first time experimental data for SLS algorithms on the software verification benchmark set SWV generated by the CALYSTO static checker [2] and used as a benchmark for complete solvers by Hutter *et al.* [10].

A more detailed description of our experimental methodologies, PARAMILS settings, specifications of our run-time environment, further details of our instance sets and algorithm configurations in DAVE can be found in a supplementary online appendix, available at the UBCSAT website [19].

3 Advanced Variable Expressions

Various variable properties and VEs play a prominent role in SLS-based solvers known from the literature. Perhaps the most popular VE currently used by SLS algorithms is $\langle\mathsf{score}\rangle$, which is equivalent to the VE $\langle\mathsf{make} - \mathsf{break}\rangle$ where the properties make and break measure the number of clauses that would become satisfied and unsatisfied, respectively, if the variable were to be flipped. The WALKSAT/SKC algorithm [16] was the first algorithm to use the even simpler VE $\langle\mathsf{break}\rangle$ for scoring variables and also introduced a Boolean $\mathsf{freebie}$ property that is true if, and only if, break equals zero. Algorithms with dynamic clause penalties, such as SAPS, use a *(penalized)* property $\mathsf{penScore}$ that reflects the dynamic clause penalty values (weights). The $\mathrm{G^2WSAT}$ algorithm uses a Boolean $\mathsf{promising}$ property that indicates a positive score property value, but only under certain circumstances (see [13] for details).

Another variable property that is prominently used in existing SLS algorithms for SAT is age; it is defined as the number of search steps that have occurred since the given variable was last flipped. The age property is closely related to the flips property (*a.k.a. flipcount*) used by the HSAT algorithm [7] as a *tie-breaking* mechanism; the flips property measures how many times a variable has been flipped. An interesting and effective combination of the $\mathsf{freebie}$, break, age and flips properties is used in the VW2 algorithm [15].

3.1 Deconstructing VW2

In many ways, Prestwich's VW2 algorithm [15] provided the starting point for our work on VEs, and we describe VW2 in the following.[1] Each variable is assigned a *weight* (which we call the $\mathsf{vw2w}$ property) initialized to zero. At each search step the flip candidates are those variables that appear in a randomly selected unsatisfied clause. If there are any candidates with a $\mathsf{freebie}$ value of one, one of those is selected; otherwise, with probability p, a candidate is selected uniformly at random, and in the remaining cases (*i.e.*, with probability $(1-p)$), the candidate is selected with the smallest value of the VE:

$$\mathsf{break} + c \cdot (\mathsf{vw2w} - \overline{\mathsf{vw2w}}) \,, \tag{1}$$

where the constant c is a parameter and $\overline{\mathsf{vw2w}}$ denotes the average of the $\mathsf{vw2w}$ property across all variables. When a variable is flipped, its $\mathsf{vw2w}$ property is updated according to:

$$\mathsf{vw2w} := (1-s) \cdot (\mathsf{vw2w} + 1) + s \cdot \mathbf{step} \,, \tag{2}$$

[1]For consistency with other parts of our study, we chose to use our notations instead of Prestwich's when describing VW2.

where s is another constant parameter, and $\mathbf{step}$ is the current step iteration value.

A variant of VW2 that we call VW2-SAT05 received the bronze medal in the satisfiable random category of the 2005 SAT competition. This variant eliminates the three VW2 parameters (s, c, p) by setting p to zero and introducing a randomized mechanism to change the behaviour of c and s during the search; it has been included recently in the SATENSTEIN-LS [12] and HYBRID [20] algorithms. However, in our experiments, we found that the original VW2 procedure with parameter settings optimized for a given set of benchmark instances will often outperform VW2-SAT05. In particular, we observed this performance difference on the CBMC software verification instances described in Section 2. In experiments not presented here (see [19] for details), we found that VW2 with parameters $(s, c, p) = (0, 0.01, 0.2)$ is the best-performing SLS-based SAT algorithm currently known for CBMC, which motivated us to study it in more depth.

Upon closer examination of the VW2 VE shown in Equation 1 above, we noticed that the $\overline{\mathsf{vw2w}}$ term can be removed without changing the behaviour of VW2, since this term is constant over all variables and therefore does not affect the variable selection. In the vw2w property update procedure, the s parameter is a *smoothing* parameter. if s is set to one, the VE becomes equivalent to $\langle \mathsf{break} - c \cdot \mathsf{age} \rangle$. If s is set to zero, as in the optimal setting for CBMC, the variable property vw2w becomes equivalent to $\langle \mathsf{break} + c \cdot \mathsf{flips} \rangle$.

For very small values of c, it may appear as though the vw2w property acted as a tie-breaking mechanism, and Prestwich observed that when s is zero, VW2 behaves like HSAT [7]. While it may be easy to dismiss the mechanics of VW2 as a simple tie-breaking scheme, this simplification does not seem justified when considering the parameter settings obtained for VW2 and the length of typical runs required for solving CBMC instances. In our analysis of VW2 on the hardest CBMC instance, we observed that for over half of the search steps the break and flips properties were interacting in a complex way, and VW2 was making trade-offs between satisfying additional clauses (intensification) and changing the values of rarely flipped variables (diversification).

3.2 VW2+VE: Modifying the VE in VW2

Considering this type of complementarity in the role of the break and flips properties and the strong performance of VW2, it seemed promising to explore different ways of constructing a VE based on those two properties. Because the difference in scale between the two properties becomes increasingly larger as the search progresses, we decided to *normalize* the values of these properties to the interval $[0, 1]$. We achieved this using the formula $\frac{\mathsf{p}}{\max(\mathsf{p})}$, where $\max(\mathsf{p})$ refers to the maximum value of the property p for all flip candidates, which for VW2 would be those variables in the currently selected clause.

In addition to normalizing the property values, we also allowed for *non-linear* interaction between the two properties. Our motivation was that the *relative* difference in magnitude between two different property values could have an important impact on the behaviour of the algorithm. Since the values have already been normalized, we used a simple polynomial transformation on the normalized values of the flips property, to

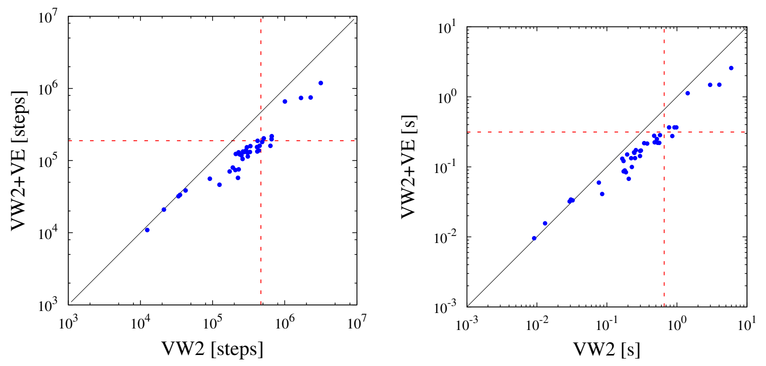

Fig. 1. VW2+VE *vs* VW2 on CBMC . Each point corresponds to the median run-length (left) and run-time (right) from 25 runs on an instance in the CBMC test set. The mean values of those medians are indicated by the dashed lines. The ratio of the means (which we denote as the speedup factor *s.f.*) is 2.47 (left) and 2.10 (right).

obtain the generalized VE:

$$\frac{\mathsf{break}}{\max(\mathsf{break})} + c \cdot \left(\frac{\mathsf{flips}}{\max(\mathsf{flips})}\right)^{a} \;, \tag{3}$$

which we used to replace the scoring function of VW2. We refer to the resulting variant of VW2, in which we also disabled smoothing, as VW2+VE. Automated configuration of this algorithm for our CBMC training set using PARAMILS yielded the parameter configuration $(c, a, p) = (0.95, 8, 0.05)$ (see [19] for details).

As can be seen from Figure 1, the use of this generalized VE leads to improved performance in terms of local search steps required for solving the CBMC instances (as always, we show results for the test set, which is disjoint from the training set used for parameter optimization). However, the VE is more complex, and evaluating it requires an additional initial iteration to determine the maximum values. This leads to a less pronounced improvement in terms of time performance, which is illustrated in Figure 1 (right). Still, VW2+VE performs better than VW2 on the CBMC benchmark, which – based on our earlier findings – makes it the best SLS-based SAT algorithm for that benchmark currently available.

3.3 Normalization in VEs

In VW2+VE, we normalized the break and flips properties so they would fall within the interval $[0, 1]$. We will now generalize this further, using from here on the notation $\|x\|$ in VEs to indicate that the value x has been normalized using one of several different methods. The method used in VW2+VE, $\|x\| = \frac{x}{\max(x)}$, preserves ratios between the values being normalized. Alternatively, a flat normalization $\|x\| = \frac{x-\min(x)}{\max(x)-\min(x)}$ forces the maximum and minimum to be one and zero, respectively, and a summation normalization $\|x\| = \frac{x}{\mathrm{sum}(x)}$ forces the sum of the values to be one. Of course, numerous other normalizations are possible, including non-linear normalizations and normalizations more suitable for both positive and negative values.

In the literature, some scoring functions are designed to select variables with the *minimum* value (such as VW2's), whereas others select the variable with the *maximum* value (such as the traditional $\langle \mathsf{make} - \mathsf{break} \rangle$). Both cases are common, and which one should be used is usually obvious from the context; however, this may not always be the case as we consider more complicated VEs. To address this issue, we first note that the question of favouring minimum or maximum values already arises for variable properties: for example, a small value of flips is considered favourable, while the opposite is true for age. To facilitate the construction of more complex VEs, we will require that all properties be transformed to favour maximum values. To this end, we revise our notation for normalization so that $\|\mathsf{p}\|$ will indicate that p has been normalized and transformed (if necessary). A simple transformation and normalization would be $(1 - \|\mathsf{p}\|)$, and we found that $\|\max(\mathsf{p}) + \min(\mathsf{p}) - \mathsf{p}\|$ worked quite effectively in practice.

When normalizing the make and break properties, we observed that they can also be normalized *w.r.t.* the number of clauses in which the variable appears. We will introduce the variable properties $\mathsf{relMake}$ and $\mathsf{relBreak}$ to correspond to the *relative* number (fraction) of clauses that become satisfied or unsatisfied, respectively, as a result of flipping a given variable. For example, if the positive literal x occurs in $\mathsf{numPosOcc}$ clauses and the negative literal $\neg x$ occurs in $\mathsf{numNegOcc}$ clauses, then the value of $\mathsf{relMake}$ is equivalent to $\langle \mathsf{make}/\mathsf{numPosOcc} \rangle$ when x is false and $\langle \mathsf{make}/\mathsf{numNegOcc} \rangle$ when x is true. While for randomly generated instances with uniform structure, normalizing the score in this manner would have no material effect, for structured formulae, such as the CBMC instances, there is often large variability in the number of clauses each variable appears in, and consequently, this normalization can make a substantial difference. Ansótegui *et al.* explored the *scale-free* structure of industrial instances and the impact of this structure on complete solvers [1], and we believe that there is potential for SLS algorithms to exploit this structure as well.

Another observation we made is that existing algorithms combine make and break symmetrically, but there may be an advantage to constructing VEs in which they are weighted differently. We therefore consider the generalized VE $\langle c_1 \cdot \mathsf{make} - c_2 \cdot \mathsf{break} \rangle$, which uses simple scaling to weight the two variable properties differently. We note that WALKSAT/SKC [16] can be seen as using a special case of this VE where $c_1 = 0$. While it is possible that in many cases choosing $c_1 = 1$ may lead to the best performance, there is no reason to assume that this would always be the case.

Finally, we observed that the summation normalization $(\frac{x}{\mathrm{sum}(x)})$ behaved quite differently than the one we used in VW2+VE $(\frac{x}{\max(x)})$, even though at first glance it would appear that they should only differ by a constant factor. However, that constant factor is the *clause length*, which is constant for any particular search step, but can differ between search steps. In other words, we discovered that normalization *w.r.t.* the clause length can be beneficial, and we believe that such normalizations merit further study.

3.4 WALKSAT+VE: Modifying the VE in WALKSAT

To investigate the potential latent in the generalizations introduced up to this point, we constructed a new SLS algorithm we call WALKSAT+VE. This algorithm is obtained

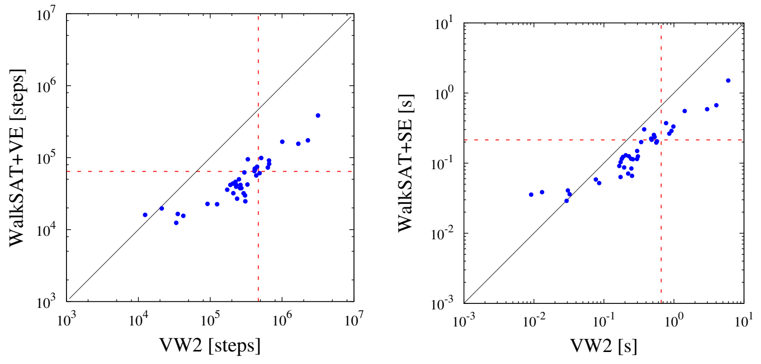

Fig. 2. WALKSAT+VE *vs* VW2 on CBMC . The *s.f.* is 7.25 (left) and 3.07 (right). (The data is presented analogously to that in Figure 1.)

from the original WALKSAT/SKC algorithm by replacing the VE ⟨break⟩ with the following VE that makes use of scaling, normalizations and non-linear transformations:

$$c_1 \cdot \|\text{make}\|^{a_1} + c_2 \cdot \|\text{relMake}\|^{a_2} + c_3 \cdot \|\text{break}\|^{a_3} + c_4 \cdot \|\text{relBreak}\|^{a_4} \quad . \qquad (4)$$

Whereas VW2+VE benefited from the flips property providing diversification, this VE uses only greedy components (make and break) and a standard random walk mechanism. To test the effectiveness of our new algorithm, we ran PARAMILS to optimize the values of the constants and the normalization parameters (hidden in the $\|p\|$ notation) on the CBMC training set (see [19] for details).

The performance of the configuration thus obtained on the CBMC test set is illustrated in Figure 2. Our new WALKSAT variant significantly outperforms the previously best known SLS algorithm for this benchmark (VW2) and solves it more than 1 000 times faster than WALKSAT/SKC. These results are especially impressive when examining step performance, but because of the complexity involved with this advanced VE, the results *w.r.t.* time performance are somewhat less impressive, but still significant. We were genuinely surprised that with this relatively modest modification to the venerable, but rather dated WALKSAT/SKC algorithm, we were able to outperform all known SLS algorithms. This experiment clearly demonstrate the potential of complex VEs as a basis for the development of new, high-performance SLS algorithms.

4 Modeling and Designing SLS Algorithms with VEs

Now that we have motivated our interest in VEs, we will present our VE-based model. Our model, as illustrated in Figure 3, includes an *algorithm controller* and three core stages: a variable filter stage, a VE evaluation stage and a variable selection stage. There is a final stage that simply flips the selected variable and updates the state information resulting from the flip (*e.g.*, property values) and any algorithm state information (such as the noise value in algorithms with adaptive noise). We will first describe the three core stages and then describe the algorithm controller.

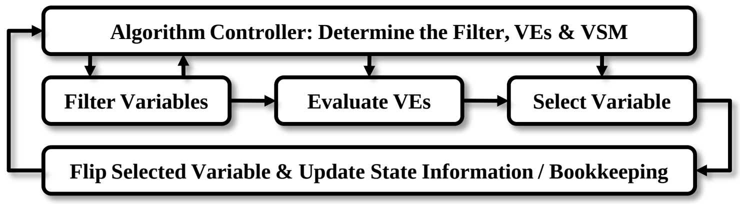
Algorithm Controller: Determine the Filter, VEs & VSM
Filter Variables
Evaluate VEs
Select Variable
Flip Selected Variable & Update State Information / Bookkeeping

Fig. 3. Our conceptual SLS algorithm model.

The **Variable Filter Stage** outputs a list of variables that are candidates to be flipped in this search step. For example, the clause-based filter used in WALKSAT/SKC [16] and VW2 [15] selects an unsatisfied clause uniformly at random, and then only the variables that appear in that clause are flip candidates. Other examples include the GSAT algorithm [17], which considers all variables, the SAPS algorithm, which includes all variables that appear in unsatisfied clauses, and the G^{2}WSAT algorithm [13], which includes a filter that only allows variables with a promising property value of one.

The **VE Evaluation Stage** is very straightforward. The input is the list of n flip candidates from the filter stage and k VEs from the controller, and the output is an array of $n \times k$ values where each of the VEs are evaluated for each candidate.

The **Variable Selection Stage** makes the final decision as to which of the candidates will be flipped, based on the array of values from the VE evaluation stage. For simplicity, we will assume that a single candidate is selected and flipped in each step, but in practice, the VSM could select zero or many candidates. For most existing SLS algorithms, the variable selection mechanism (VSM) is a simple $\max$ (or $\min$) operation, where the candidate with the maximum value of the first VE is selected; additional VEs can be used for tie-breaking, and any remaining ties will be broken randomly. The NOVELTY algorithm [14] is an example of an algorithm with a VSM that incorporates multiple VEs (score and age).

The **Algorithm Controller** controls the behaviour at each step by determining the components of each of the three stages: the filter, the set of VEs and the VSM. The controller may use the same components for every step, make independent random decisions for each step or it may use a more sophisticated decision mechanism. The GSAT algorithm [17] represented in our model uses a simple controller, where the components are the same at every step: no filter (consider all variables), use a simple VE of $\langle$score$\rangle$ and a $\max$ VSM. The GWSAT algorithm added a random walk to GSAT, and is represented in our model by a randomized controller that with some probability selects an alternate filter (only variables that appear in unsatisfied clauses) and a VSM that selects candidates randomly. In Figure 3, we indicate control flow from the filter back to the controller to allow for controllers that may wish to re-filter the variables or defer the determination of the VEs or VSM until after the filter results are known. For example, as a form of clause normalization (see Section 3.3), a controller could use a random-clause-based filter and choose VEs based on the length of the selected clause.

In our model, complex controllers can be constructed that do not directly decide the components for the three stages, but instead utilize a number sub-controllers. Since each sub-controller can correspond to a unique algorithm (or the same algorithm with different parameter settings), this allows the construction of *hybrid* algorithms. A hybrid algorithm can switch between different algorithms randomly, periodically, when some criteria is met (*e.g.*, search stagnation is detected) or according to some other customized mechanism. G^2WSAT is one such hybrid algorithm, where if any variables have a $\mathsf{promising}$ property of one, a GSAT-based step occurs, otherwise, a WALKSAT-based step occurs [13].

Now that we have presented our highly flexible model, we will briefly outline our Design Architecture for Variable Expressions (DAVE), based on our versatile UBCSAT architecture [18]. (For a complete and up-to-date description of DAVE, consult the UBCSAT website [19].) One of the design goals of DAVE was to reduce (and potentially eliminate) the programming component of algorithm design by allowing the entire algorithm behaviour to be specified at run-time. The user can specify the algorithm controller (and sub-controllers), the filter(s), the VE(s) and the VSM(s). The only programming required is to introduce *new* variable properties, controllers, filters or VSMs. Because the *configuration space* of DAVE is actually an *algorithm specification space*, when we use DAVE in combination with an automated configurator, we can find optimized algorithm specifications automatically. To further facilitate the use of a configurator, DAVE supports a sophisticated macro-based syntax that allows controllers, filters, VEs, and VSMs to be highly paramaterized.

In DAVE, most variable properties depend on the current value of the variable. We use the notation p' to correspond to the property value for the negation of a given variable. For example, the flips property in DAVE is actually half of the total flip count $(\mathsf{flips} + \mathsf{flips}')$; similarly, age' ignores the most recent flip and measures the number of search steps that have occurred since the flip prior to the most recent flip.

The only other implementation detail of DAVE that we will address here, as it is specifically relevant to the presentation and understanding of the performance results we report later, is the interpreted nature of the algorithms specified in DAVE. Since DAVE receives the algorithm specification and VEs at run-time, the code is not natively compiled, but instead, each operation is individually interpreted and executed. This means that an algorithm in DAVE will not achieve the same performance as the equivalent algorithm in compiled source code *w.r.t.* CPU time, which is why we encourage measuring DAVE algorithms by their step performance where there is no such penalty. In preliminary experiments, we have seen algorithms in DAVE run 1.5-3 times slower than their native implementations, where the speed of DAVE is often more a function of the number of operators used in the VE, as opposed to the true complexity of the algorithm. This is one reason why we present DAVE as a design architecture that facilitates the exploration of new algorithmic ideas; it is our intent that new and robust algorithms that are developed in DAVE will subsequently be incorporated directly in UBCSAT as stand-alone optimized algorithms. We are currently in the preliminary stages of developing a software tool that can automatically generate fast, native source code that will implement an algorithm specified in DAVE.

5 VE-SAMPLER: Exploring New SLS Methods using DAVE

In this section we introduce a new algorithm framework we call VE-SAMPLER. VE-SAMPLER uses a randomized controller that selects between six sub-controllers, where each sub-controller is selected with a probability proportional to a configurable weight. Each of the six sub-controllers uses a simple $\max$ VSM, and has a configurable clause-based filter, where the unsatisfied clause selected is either random, the clause unsatisfied the longest, or the clause most frequently unsatisfied. The VE of the first sub-controller is $\langle \mathsf{freebie} \rangle$, similar to the random walk in WALKSAT/SKC [16]; the $\max$ VSM will select all freebie candidates, or all candidates if no freebies exist, and then break ties randomly. The VEs for the other five sub-controllers are all of the form:

$$\|\mathsf{p1}\|^{a_1} + \mathrm{clw}(s,m,l) \cdot \|\mathsf{p2}\|^{a_2} \ , \tag{5}$$

where $\mathsf{p1}$ and $\mathsf{p2}$ are configurable, and correspond to variable properties (or a ratio of properties) selected from lists we describe below. The clw function represents a simple mechanism we created to addresses clause normalization (briefly discussed in Section 3.3) in a practical, yet interesting way; the three configurable parameters of $\mathrm{clw}(s,m,l)$ correspond to scaling coefficients that depend on whether the clause length is small (< 3), medium $(= 3)$, or large (> 3); *i.e.*, if the clause length is two then $\mathrm{clw}(s,m,l) = s$.

The VE described in Equation 5 is similar to the VEs in VW2+VE and WALK-SAT+VE *w.r.t.* the normalization and non-linear transformation used. We chose to use only two properties to avoid the reduction in CPU time performance we saw with four properties in WALKSAT+VE; however, we believe that our approach of using multiple VEs via a controller can provide a similar level of algorithm robustness without significantly degrading per-step time complexity. Of the five sub-controllers, one was configured to have only *greedy* properties similar to WALKSAT+VE, while the remaining four were configured to have one greedy property ($\mathsf{p1}$) and one *diversification* property ($\mathsf{p2}$) similar to VW2+VE. The five greedy properties available were score, make, relMake, break and relBreak.

We wanted diversification properties that were independent of the greedy variable properties and required little or no computational overhead to maintain. For VE-SAMPLER we created the following new properties: flitCount is incremented every search step where the variable (with its current value) has appeared in the list of flip candidates, relFiltCount is similar, but increases by $1/\mathbf{clauselen}$, and goodFlips and badFlips are incremented every time the variable (with its current value) is flipped and the number of satisfied clauses goes up or down, respectively. In total, there were thirteen diversification properties (or ratios of properties) available in VE-SAMPLER:

flips,	age/flips,	relFiltCount,	goodFlips/flips,
age,	age′/age,	relFiltCount/flips,	goodFlips/goodFlips′,
age′,	filtCount,	relFiltCount/relFiltCount′,	goodFlips/badFlips

and rand, which draws a number uniformly at random from the interval $[0, 1]$. While some of these properties are based on prior evidence and intuition, others are simply interesting ideas that we thought might be effective.

Our goal with VE-SAMPLER was to make very few decisions at design time and to configure the resulting, highly paramaterized algorithm automatically for optimized performance [8]. In total, VE-SAMPLER has over 10^{50} possible configurations, which, to the best of our knowledge, is the largest design space searched using PARAMILS [11]

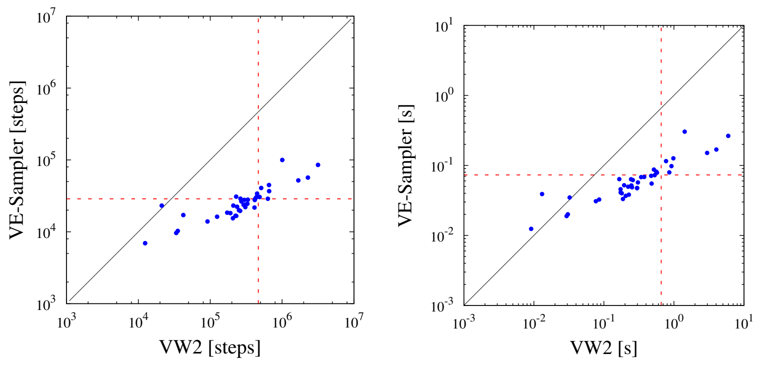

Fig. 4. VE-Sampler *vs* VW2 on CBMC . The *s.f.* is 16.2 (left) and 9.0 (right). (The data is presented analogously to that in Figure 1.)

Algorithm	CBMC			SWV (partial)				SWV (full)	
	Steps	Time		Steps	Time		%	PAR	%
	$\times 10^3$	sec.	*s.f.*	$\times 10^3$	sec.	*s.f.*	Compl.		Compl.
VW2-Sat05	3 577	6.22	0.11	10 089	19.20	0.16	100	3 008	50.1
VW2	467	0.66	*ref.*	1 555	3.10	*ref.*	100	3 042	49.3
SATenstein-LS	228	0.80	0.82	1 465	12.50	0.25	100	3 040	49.5
VE-Sampler	**29**	**0.07**	**9.00**	**245**	**0.90**	**3.61**	100	**2 664**	**50.7**

Fig. 5. Experimental Results for VE-Sampler . Values shown are the means of the median run-length and run-time from (left) 25 runs on instances from the CBMC test set and (right) 10 runs on instances from SWV. The *s.f.* is the ratio of the time *w.r.t.* VW2. All algorithms completed 100% of the CBMC instances. The PAR (Penalized Average Run-time) is the average from all runs on all instances, where incomplete runs after 600 seconds are penalized by a factor of 10 (6 000 seconds) (see [12] for details). All algorithms (except the parameterless VW2-Sat05) were optimized by ParamILS.

so far. We present the results of our ParamILS-configured VE-Sampler in Figures 4–6. We compared VE-Sampler against the SLS-based solvers VW2 [15] and SATenstein-LS (see Section 6), both also configured with ParamILS (see [19] for details). The results we present were obtained using a compiled version of VE-Sampler, where the original version, implemented in DAVE, was approximately 1.5 times slower.

VE-Sampler performs substantially better than VW2 and SATenstein-LS on our CBMC test set, especially in terms of search steps. On the much more challenging real-world software verification instances from the SVW set, VE-Sampler also performs significantly better than VW2 and SATenstein-LS. We note that none of the SLS algorithms we are aware of can solve more than about half of the complete set of SWV instances within our 600 second cutoff, but VE-Sampler does solve the other half of the instances more efficiently than any other SLS algorithm. While the results in Figure 5 are impressive and represent the current state-of-the-art in SLS-based SAT

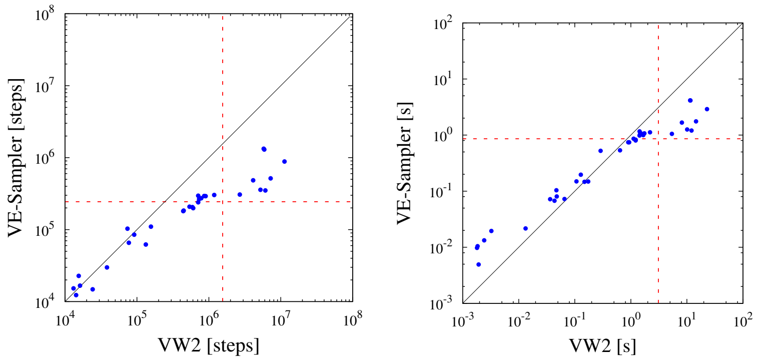

Fig. 6. VE-SAMPLER *vs* VW2 on SWV (partial). Each point corresponds to the median run-length (left) and run-time (right) from 10 runs on an instance in the SWV (partial) test set. The *s.f.* is 6.36 (left) and 3.61 (right).

solvers on these types of instances, the complete solver PICOSAT [3] is twice as fast as VE-SAMPLER on CBMC, seven times as fast on SWV (partial) and can solve any instance from the full SWV set in just a few CPU seconds. Thus, while we have considerably reduced the performance gap between SLS-based and DPLL-based SAT solvers on these software verification instances, there is still much room for improvement.

When studying the VE-SAMPLER configurations found by PARAMILS, we observed that configurations with similarly good performance often had substantially different configurations. This might suggest that VE-SAMPLER is somewhat robust *w.r.t.* its configuration, and that PARAMILS was far from finding the true optimal configuration of VE-SAMPLER (with over 10^{50} possible configurations, this is not surprising). We also observed configurations where two sub-controllers would be configured to use the same variable properties, but to be quite different otherwise. This was the case in the configurations featured in the results above, where the final CBMC configuration heavily weighted two sub-controllers with the properties relMake and age′, and the final SWV configuration heavily weighted two sub-controllers with the properties break and flips (see [19] for details). We believe this suggests that a hybrid algorithm including multiple configurations of the same underlying mechanism can achieve very robust performance.

6 Related Work

The manner in which SLS algorithm hybrids can be implemented in DAVE can be seen as a generalization of the HYBRID algorithm by Wei *et al.* [20]. HYBRID implements a clever heuristic to select between the algorithms VW2-SAT05 and ADAPTG^{2}WSAT at each search step. Their heuristic corresponds to a specific algorithm controller in our model, and once implemented in DAVE, it becomes a universal controller that can be used to select between *any* two algorithms. Furthermore, the selection of the algorithms to be hybridized can be achieved by using an automated configurator.

DAVE is conceptually related to the SATENSTEIN-LS solver by KhudaBukhsh *et al.* [12], which also extends UBCSAT, albeit in a different direction. SATENSTEIN-LS incorporates proven components from over two dozen existing SLS algorithms, including GNOVELTY$^+$, ADAPTG^{2}WSAT$_+$, SAPS and PAWS (see [12] for details) and can be configured to instantiate any of those algorithms, as well as many complex hybrids. SATENSTEIN-LS is very efficient when properly configured and is the best known SLS algorithm on several benchmark sets [12]. Whereas the SATENSTEIN-LS authors liken their generated algorithms to Frankenstein's monster, stitched together from existing algorithm parts, we believe that our model is more akin to a mad scientist experimenting with algorithmic DNA. The significant difference is that SATENSTEIN-LS has a bounded configuration space, whereas DAVE is a design environment that supports arbitrarily complex algorithms in a potentially unbounded space.

In that latter respect, DAVE is similar in nature to the Composite heuristic Learning Algorithm for SAT Search (CLASS) by Fukunaga [6]. CLASS is a genetic programming system that constructs new variable selection heuristics. Our work with VEs is somewhat orthogonal to the research direction underlying CLASS; our goal has been to decouple the scoring functions (VEs) from the VSMs and focus on the VEs, whereas in CLASS they are tightly coupled. There is potential for combining the strategies of DAVE and CLASS, and we are considering incorporating a CLASS-like syntax for VSMs into a future version of DAVE. Conversely, CLASS could be extended by incorporating our concept of VEs.

7 Conclusions & Future Work

In this work, we have proposed a new conceptual model for SLS algorithms based on variable expressions (VEs), and we demonstrated that algorithms with complex VEs can be very effective in practice. We created a new software framework for designing new SLS algorithms and algorithm hybrids in our model, and we demonstrated that by combining our software with an automated algorithm configuration tool, it was quite easy to construct a new algorithms that is nine times faster than the existing state-of-the-art SLS-based SAT solvers on a set of software verifications known from the literature.

Apart from the previously mentioned work on CLASS-based VSMs (Section 6) and the automated generation of source code from DAVE configurations (Section 4), we see several other promising directions for future work. We expect that there are more variable properties that can be effectively incorporated into VEs, as well as more sophisticated ways of combining variable properties beyond the simple normalization, scaling and non-linear transformations we presented in this work; we especially believe that there are more effective ways to handle clause normalization. Now that we have conceptually separated the components of algorithm controllers, filters, VEs and VSMs, we believe that algorithm designers will be able to focus on those individual components; with the ability to quickly and automatically test their ideas in DAVE, we anticipate rapid development in each of these areas. Overall, we believe that the utilization of rich and flexible design environments such as DAVE in combination with powerful automated configuration tools will make it possible to achieve further, substantial progress in the state of the art in SLS-based SAT solving.

Acknowledgements

We would like to thank the anonymous reviewers for their valuable feedback. This research has been enabled by the use of computing resources provided by WestGrid and Compute/Calcul Canada. Furthermore, HH acknowledges funding received through the NSERC Discovery Grants Program.

References

1. Ansótegui, C., Bonet, M.L., Levy, J.: On the structure of industrial SAT instances. In: Proceedings of CP-09. LNCS, vol. 5732, pp. 127–141 (2009)
2. Babić, D., Hu, A.J.: Calysto: Scalable and precise extended static checking. In: Proceedings of ICSE-08. pp. 211–220 (2008)
3. Biere, A.: PicoSAT essentials. Journal on Satisfiability, Boolean Modeling and Computation 4, 75–97 (2008)
4. Clarke, E., Kroening, D., Lerda, F.: A tool for checking ANSI-C programs. In: Proceedings of TACAS-04. LNCS, vol. 2988, pp. 168–176 (2004)
5. Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. In: Proceedings of SAT-05. LNCS, vol. 3569, pp. 61–75 (2005)
6. Fukunaga, A.S.: Automated discovery of local search heuristics for satisfiability testing. Evolutionary Computation 16(1), 31–61 (2008)
7. Gent, I.P., Walsh, T.: Towards an understanding of hill-climbing procedures for SAT. In: Proceedings of AAAI-93. pp. 28–33 (1993)
8. Hoos, H.H.: Computer-aided design of high-performance algorithms. Tech. Rep. TR-2008-16, Department of Computer Science, University of British Columbia (2008)
9. Hoos, H.H., Stützle, T.: Stochastic Local Search: Foundations and Applications. Morgan Kaufmann (2005)
10. Hutter, F., Babić, D., Hoos, H.H., Hu, A.J.: Boosting verification by automatic tuning of decision procedures. In: Proceedings of FMCAD-07. pp. 27–34 (2007)
11. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: ParamILS: An automatic algorithm configuration framework. Journal of Artificial Intelligence Research 36, 267–306 (2009)
12. KhudaBukhsh, A.R., Xu, L., Hoos, H.H., Leyton-Brown, K.: SATenstein: Automatically building local search SAT solvers from components. In: IJCAI-09. pp. 517–524 (2009)
13. Li, C.M., Huang, W.Q.: Diversification and determinism in local search for satisfiability. In: Proceedings of SAT-05. LNCS, vol. 3569, pp. 158–172 (2005)
14. McAllester, D., Selman, B., Kautz, H.: Evidence for invariants in local search. In: Proceedings of AAAI-97. pp. 321–326 (1997)
15. Prestwich, S.: Random walk with continuously smoothed variable weights. In: Proceedings of SAT-05. LNCS, vol. 3569, pp. 203–215 (2005)
16. Selman, B., Kautz, H.A., Cohen, B.: Noise strategies for improving local search. In: Proceedings of AAAI-94. pp. 337–343 (1994)
17. Selman, B., Levesque, H., Mitchell, D.: A new method for solving hard satisfiability problems. In: Proceedings of AAAI-92. pp. 459–465 (1992)
18. Tompkins, D.A.D., Hoos, H.H.: UBCSAT: An implementation and experimentation environment for SLS algorithms for SAT and MAX-SAT. In: Revised Selected Papers of SAT-04. LNCS, vol. 3542, pp. 306–320 (2005)
19. UBCSAT Website: `http://www.satlib.org/ubcsat`
20. Wei, W., Li, C.M., Zhang, H.: A switching criterion for intensification and diversification in local search for SAT. Journal on Satisfiability, Boolean Modeling and Computation 4, 219–237 (2008)