Stochastic Local Search for Satisfiability Modulo Theories

Andreas Fröhlich and **Armin Biere**
Johannes Kepler University

Christoph M. Wintersteiger and **Youssef Hamadi**
Microsoft Research

Abstract

Satisfiability Modulo Theories (SMT) is essential for many practical applications, e.g., in hard- and software verification, and increasingly also in other scientific areas like computational biology. A large number of applications in these areas benefit from bit-precise reasoning over finite-domain variables. Current approaches in this area translate a formula over bit-vectors to an equisatisfiable propositional formula, which is then given to a SAT solver. In this paper, we present a novel stochastic local search (SLS) algorithm to solve SMT problems, especially those in the theory of bit-vectors, directly on the theory level. We explain how several successful techniques used in modern SLS solvers for SAT can be lifted to the SMT level. Experimental results show that our approach can compete with state-of-the-art bit-vector solvers on many practical instances and, sometimes, outperform existing solvers. This offers interesting possibilities in combining our approach with existing techniques, and, moreover, new insights into the importance of exploiting problem structure in SLS solvers for SAT. Our approach is modular and, therefore, extensible to support other theories, potentially allowing SLS to become part of the more general SMT framework.

1 Introduction

Satisfiability Modulo Theories (SMT) represents the decision problem for logical formulas with respect to certain background theories. It combines the problem of Boolean satisfiability (SAT) with other areas, e.g., the theories of integers, real numbers, lists, arrays, and bit-vectors, and has many different applications, predominantly in hard- and software verification (De Moura and Bjørner 2009; Barrett et al. 2009; Yuan, Pixley, and Aziz 2006; Godefroid, Levin, and Molnar 2008; Naveh et al. 2007). While most of the methods presented in this paper are generally applicable, we focus on the theory of bit-vectors (quantifier-free and fixed-size),which enjoys decidability, but pays the high price of being NEXPTIME-complete, as (Kovásznai, Fröhlich, and Biere 2012) have shown. Examples of state-of-the-art SMT solvers with support for bit-precise reasoning are Boolector (Brummayer and Biere 2009), MathSAT (Bruttomesso et al. 2008), and Z3 (De Moura and Bjørner 2008).

Most approaches for solving this kind of formulas rely on translating the input formula into SAT (dubbed 'bit-blasting') and then handing it to a SAT solver, most often of the conflict driven clause learning (CDCL) kind. In this paper, we present a novel stochastic local search (SLS) algorithm to solve bit-vector formulas directly on the theory level. SLS is a heuristic method which has always played an important role in AI and is successfully applied to many different problems in various areas, e.g., see (Hoos and Stützle 2005). Today, SLS is not a usual ingredient in SMT solvers. We intend to close this gap by providing an SLS algorithm (specialized for the theory of bit-vectors), which is, for the most part, easy to adapt for other theories. Besides avoiding the blowup in size that often comes with bit-blasting, applying SLS techniques on the bit-vector level has several advantages. For example, structural information, i.e., word-level information, is used to guide the search directly. In contrast, a CDCL SAT solver operating on the propositional representation is not aware of this information. Nevertheless, it is possible to profit from techniques used in SAT solving also in SLS on the SMT representation. We show how several techniques that are common in SLS SAT solvers are successfully lifted to the SMT level. In many cases of practical applications (Yuan, Pixley, and Aziz 2006; Godefroid, Levin, and Molnar 2008; Naveh et al. 2007), input formulas are actually expected to be satisfiable, making them well-suited for SLS algorithms. The idea of integrating SLS solvers with other solvers has been explored before, either by employing an SLS solver on the Boolean skeleton of a formula (Griggio et al. 2011), or by explicit incorporation of high-level constraints, either learned automatically, or provided by the user (Naveh 2004). Apart from this, model-driven techniques (though, not local search based) exist for arithmetic theories (de Moura and Jovanovic 2013).

Our experimental results show that the SLS approach we present is competitive with state-of-the-art bit-vector solvers on many practical instances and that it frequently outperforms existing SMT solvers based on bit-blasting. While we found that bit-blasting solvers are still faster overall, this offers an interesting line of research combining SLS with existing techniques, with the goal of improving the state-of-the-art for SMT solvers. Although having undergone years of developement, existing SLS solvers for SAT turn out to perform worse than our approach, sometimes by orders of magnitude. From a theoretical point of view, the importance of exploiting problem structure in SAT solvers

has often been conjectured and discussed (Thiffault, Bacchus, and Walsh 2004; Belov, Järvisalo, and Stachniak 2011; Drechsler, Junttila, and Niemelä 2009; Järvisalo, Junttila, and Niemelä 2005; Pham, Thornton, and Sattar 2007). Nevertheless, previous attempts have not yielded in efficient techniques that play a role in state-of-the-art solvers so far. However, the performance of our algorithm clearly demonstrates that SLS solvers can indeed benefit from structural information during search.

The remaining part of the paper is structured as follows: In Sect. 2, we define the logic and the input format that we use. A brief overview about stochastic local search and the architecture of our algorithm is presented in Sect. 3. In Sect. 4, we give details and concrete implementations of all components used in our algorithm. Furthermore, we describe how several techniques used in SLS solvers for SAT are adopted for SMT. We present an experimental evaluation in Sect. 5 and further discuss insights we gained, as well as possible future work in Sect. 6. Finally, we compare our approach to related work in Sect. 7 and conclude in Sect. 8.

2 Preliminaries

The theory of fixed-width bit-vector logics (i.e., logics where each bit-vector has a given, fixed bit-width) is discussed in many different settings (e.g., (Barrett, Dill, and Levitt 1998; Bjørner and Pichora 1998; Bruttomesso and Sharygina 2009; Cyrluk, Möller, and Rue 1997; Franzén 2010)). Several different formats for bit-vector logics exist, perhaps currently the most common being the SMT-LIB format (Barrett, Stump, and Tinelli 2010). In this paper, we use a restricted definition of a bit-vector logic, which is the input that our SLS algorithm accepts. This form of (simplified) formulas is easily obtained through the means of any SMT solver that has facilities for converting to Negation Normal Form (NNF; we use Z3 as our SMT solver). A bit-vector formula F in NNF is defined by the following grammar:

$$\begin{aligned} F &= < oexpr > \wedge \cdots \wedge < oexpr > \\ < oexpr > &= < aexpr > \vee \cdots \vee < aexpr > \\ < oexpr > &= Atom \mid \neg Atom \\ < aexpr > &= < oexpr > \wedge \cdots \wedge < oexpr > \\ < aexpr > &= Atom \mid \neg Atom \end{aligned}$$

Atoms are either Boolean variables or relations ($=, \leq$) between two bit-vector expressions. We refer to the top-level expressions of F as *assertions*. It is easy to check that every bit-vector formula can be translated to an equivalent one in this grammar with only polynomial growth. To see this, consider that $\leq$ can be replaced by a combination of $<$ and $=$, $\geq$ by negation, and $<$, and if-then-else constructs by using implications on the Boolean level. In some cases, to achieve conversion in polynomial time (and space), it is helpful to introduce Tseitin-variables.

The concrete definition of a bit-vector term is left open on purpose; the exact syntax and semantics of the terms are not relevant in the context of our approach. All common operators, e.g. those from SMT-LIB (Barrett, Stump, and Tinelli 2010), can be used to build arbitrary syntactically valid expressions. The only condition we require is that there is a function to evaluate expressions if fixed input values are assigned to all variables they contain.

Example 1 *As a running example, consider the assertion*

$$x + 1 = y - 1 \ ,$$

where x and y are bit-vectors of size n (sometimes a large number), and the $+$ and $-$ operations are as usual, i.e., with overflow semantics. If we initialize the search at $x = 0$ and $y = 0$, or in vector notation, at

$$x = [0, \ldots, 0] \ , \ y = [0, \ldots, 0] \ ,$$

then the assertion evaluates to

$$[0, \ldots, 0, 1] = [1, \ldots, 1, 1] \ .$$

Assuming that the cost functions for $=$ is the relative number of bits that are assigned equal, an SLS SAT solver that implements only single bit-flip moves will require $n - 1$ moves, each of which slightly improving the cost.

3 Architecture

Given an optimization problem, a generic local search algorithm starts from an initial state and then iteratively moves to a neighbouring state. For the problem of propositional satisfiability, a state corresponds to a truth assignment to all Boolean variables of a given formula. The neighbourhood of a given assignment α is usually defined to be the set of all assignments that have a Hamming distance of 1 from α. Therefore, a neighbouring assignment is obtained by flipping the value of a single Boolean variable and a search consists of repeatedly flipping the values of Boolean variables until a satisfying assignment is found. Most SAT solvers consider input formulas in conjunctive normal form (CNF), i.e. formulas which are sets of clauses. In that case, a scoring function for evaluating the quality of an assignment and optimizing is naturally given by the number of unsatisfied clauses. Actual implementations mainly differ in the heuristics used during the search (Hoos and Stützle 2005).

We use a similar architecture to obtain an SLS solver for SMT problems by generalizing the notion of states to assignments to theory variables; our focus being on fixed-size bit-vector variables. A natural neighbourhood relation is then given by the set of assignments that are reached by flipping a single bit of a bit-vector variable, or the value of a Boolean variable. In the following, whenever we use the term 'variable' without giving an explicit specification of its type, the variable is either a bit-vector variable or Boolean. When we have a set that contains both types of variables and we only give a certain definition for the bit-vector variables, we implicitely treat Boolean variables as bit-vector variables of bit-width 1. Extensions to this neighbourhood relation are discussed in Sect. 4. Fig. 1 describes the high-level concept of our SLS algorithm for SMT.

To drive the search and to evaluate the quality of an assignment, we require a scoring function. We define the score s of a nested expression with respect to an assignment α recursively as a floating value:

$$\begin{aligned} s(e_1 \vee \cdots \vee e_n, \alpha) &= max\{s(e_1, \alpha), \ldots, s(e_n, \alpha)\} \\ s(e_1 \wedge \cdots \wedge e_n, \alpha) &= \tfrac{1}{n} \cdot (s(e_1, \alpha) + \cdots + s(e_n, \alpha)) \end{aligned}$$

for $i = 1$ **to** ∞
  $\alpha = initialize(F)$
  for $j = 1$ **to** $maxSteps(i)$
    $V = selectCandidates(F, \alpha)$
    $move = findBestMove(f, \alpha, V)$
    if $move \neq none$ **then** $\alpha = update(\alpha, move)$
    else $\alpha = randomize(\alpha, V)$

Figure 1: Pseudo-Code of our SLS architecture for SMT.

Further, the score of an atom is defined by

$$s(x^{[1]}, \alpha) = x|_\alpha$$

if the atom is a Boolean variable and, for $0 \leq c_1 \leq 1$, by

$$s(t_1^{[n]} = t_2^{[n]}, \alpha) = \begin{cases} 1 & \text{if } t_1|_\alpha = t_2|_\alpha \\ c_1 \cdot (1 - \frac{h(t_1|_\alpha, t_2|_\alpha)}{n}) & \text{otherwise} \end{cases}$$

$$s(t_1^{[n]} \leq t_2^{[n]}, \alpha) = \begin{cases} 1 & \text{if } t_1|_\alpha \leq t_2|_\alpha \\ c_1 \cdot (1 - \frac{t_2|_\alpha - t_1|_\alpha}{2^n}) & \text{otherwise} \end{cases},$$

with h being the Hamming distance, if the atom is a bit-vector expression. Negated atoms are evaluated analogously. The constant c_1 allows to focus on satisfying expressions by scaling all unsatisfied atoms. It is easy to check that, given a formula F and an assignment α, $s(F, \alpha)$ evaluates to 1 if and only if α is a satisfying assignment for F.

4 Implementation

Note that the algorithm in Fig. 1 contains two loops. The inner loop describes a single round of search, while the outer loop is used to implement restarts after a certain number of search steps. Facilities for restarts are not strictly required, but they increase performance in practice.

Initialization. An initial assignment is generated by setting all variables to some specific value. While SLS solvers for SAT usually use random values to initialize Boolean variables, setting all bit-vectors to 0 can sometimes be beneficial in the context of verification domains. Note that setting all bit-vectors to 0 does not correspond to setting all Boolean variables to 0 in the CNF representation. Without explicit tracking, this information is usually lost during bit-blasting.

Candidate Selection. The time spent in each search step is directly proportional to the number of possible moves that are considered. Checking the full neighbourhood of an assignment is often expensive. To avoid this, we look at the restricted neighbourhood with respect to certain *candidate variables*. Since we are looking for a satisfying assignment and our input formula is a conjunction of top-level assertions, it is reasonable to consider those variables as candidates that occur in at least one unsatisfied assertion. Changing any other variable cannot increase the score by definition. This is a well-known concept in SLS for SAT; similar to the one applied in selection heuristics of the so-called class of *GSAT* algorithms. The set of candidate variables is then further shrunk by considering only variables from *one* unsatisfied assertion, which is selected according to some heuristic beforehand. This is inspired by the so-called class of *WalkSAT* algorithms for SAT. While this comes at the cost of potentially missing the best move with respect to the score function, the overall performance of the algorithm often improves because it performs more moves per second and, at the same time, it is guaranteed that each clause has at least one variable with a wrong assignment. Further, not picking the best move with respect to the overall score is even beneficial sometimes, because it offers some diversification and makes the algorithm more robust with respect to local minima of the search space. For those reasons, WalkSAT architectures are often preferred for SAT. More details and a discussion of the GSAT and WalkSAT architectures are found in (Hoos and Stützle 2005).

For SAT, clause selection in most WalkSAT algorithms is usually done randomly. However, recent work shows that clause selection has a strong impact on the performance of WalkSAT algorithms and random selection is sometimes suboptimal. For example, breadth-first selection heuristics sometimes achieve better results (Balint et al. 2014). Still, clause selection has to be rather simple because SLS solvers for SAT often perform several million moves per second. In contrast, fast assertion selection is less important for our architecture, due to the fact that a single move is much more complicated compared to SAT. This allows us to use more sophisticated heuristics for assertion selection without the risk of it becoming the bottleneck of our algorithm.

Running some preliminary experiments showed that it is frequently better to select assertions that already have a high score (i.e., are almost satisfied). Given the results from (Balint et al. 2014), it is likely that some diversification is beneficial as well. Therefore, we use a heuristic inspired by the field of bandit theory used in the UCB (Upper Confidence Bounds) algorithm (Agrawal 1995). Let a_i be the assertions of a given formula and c_2 be some constant. We select the unsatisfied assertion that maximizes the term

$$s(a_i, \alpha) + c_2 \cdot \sqrt{\frac{\log selected(a_i)}{moves}},$$

where $selected(a_i)$ is the number of times, the specific assertion a_i has already been selected and $moves$ is the total number of search steps that have been performed so far.

Move Selection. Given a candidate set, we inspect the neighbourhood of the current assignment with respect to all candidate variables. In particular, this implies evaluating the score of each possible assignment obtained by flipping any bit of any candidate variable. Given a set of candidate variables $\{x_1^{[n_1]}, \ldots, x_k^{[n_k]}\}$, the neighbourhood is of size $N = \Sigma_i n_i$. Compared to SAT, the neighbourhood used in our architecture is way larger. Further, evaluating the score function implies updating the whole formula. This is again in contrast to SAT, where the score change is either cached or easy to compute on the fly (Balint et al. 2014). Evaluating possible moves therefore is the bottleneck of our current implementation. As pointed out in Sect. 3, we try to maximize the score function. One important feature to use during score computation is early pruning. In our solver, a formula is saved in a directed acyclic graph (DAG) structure. When

evaluating a new assignment, we have to proceed bottom-up. We start evaluating the atoms and then iteratively continue evaluating parent expressions. We do this in a breadth first way and save all current expressions in a queue structure. Due to the definition of the score function and the fact that our formula does not contain any negations other than those at the atom level, we can immediately stop evaluating a specific assignment if, at one point, the queue only contains expressions with lower scores than those for the same expressions with respect to the original assignment.

In the end, the move with the largest improvement in score is selected. If no improvement in score is possible, no move is returned. This is similar to the behaviour of many SLS algorithms for SAT. For example, the state-of-the-art solver *Sparrow* (Balint and Fröhlich 2010) also applies a similar deterministic highest-reward strategy in one of its components. In order to prevent getting stuck in local minima, we optionally allow *random walks* (Selman, Kautz, and Cohen 1994). With a certain walk probability wp, a random move is selected (even if it is non-improving). From a theoretical point of view, a random walk additionally causes our algorithm to be *probabilistically approximately complete (PAC)* (Hoos and Stützle 2005).

Example 2 *Consider the assertion in Example 1 and the state of the search being such that both variables are assigned* 0 *and of size* n*:*

$$[0, \ldots, 0, 1] = [1, \ldots, 1, 1] \ .$$

While any single-bit flip would only increase the score by $1/n$*, negating either* x *or* y *improves the score by* $(n-2)/n$*, assuming* $c_1 = 1$*, resulting in either*

$$[1, \ldots, 1, 0] = [1, \ldots, 1, 1], \textit{ or}$$
$$[0, \ldots, 0, 1] = [0, \ldots, 0, 0].$$

Update. After an improving move was found, the assignment is updated and propagated through the DAG structure of the formula. As mentioned before, scores are also stored for each subexpression when updating, in order to allow early pruning in the next search steps.

Randomization. Whenever no improving move was found, we simply set one of the candidate variables to a random value within its range and update all nodes in the formula DAG. This strong kind of randomization enables the algorithm to efficiently escape many local minima and allows to traverse new parts of the search space. In contrast to random walks during move selection, this part of the algorithm is essential for solving practical instances. Nevertheless, using only this kind of randomization does not guarantee the PAC-property from the theoretical point of view.

Assertion Weights. In SLS solvers for SAT, clause weighting schemes were an important novelty and are part of many efficient algorithms (Hoos and Stützle 2005). PAWS was the first solver to apply an additive weighting scheme and the same approach can still be found in many modern solvers (Thornton et al. 2004). We adopted this approach to SMT and used it to dynamically assign weights to the top-level assertions of an input formula during search. Each assertion a_i of F gets assigned a weight w_i. Initially, all weights are set to 1. Updates occur whenever no increasing move is possible, i.e., when we randomize, in the following way: With probability $(1 - sp)$, increase the weight w_i of all unsatisfied assertions by c_3. With probability sp, decrease the weight w_i of all satisfied assertions by c_3 to a minimum of 1. Whenever the score of the formula F with respect to an assignment α is evaluated in order to select the best move, we do so according to

$$s(F, \alpha) = w_1 \cdot s(a_1, \alpha) + \cdots + w_n \cdot s(a_n, \alpha) \ .$$

Although this new score function is not normalized anymore, this does not affect the correctness of the algorithm.

Restarts. While restarts are one of the most important features of CDCL solvers, they are usually not beneficial in SLS solvers for SAT. For our bit-vector approach restarts turn out to be beneficial. We implemented an exponential restart scheme, similar to those that are used in CDCL solvers, specifically, the Luby scheme (Luby, Sinclair, and Zuckerman 1993). We define the maximum number of steps in the i-th round as

$$\textit{maxSteps(i)} := \begin{cases} c_4, & \text{if } i \text{ is odd} \\ c_4 \cdot 2^{\frac{i}{2}}, & \text{if } i \text{ is even} \end{cases}$$

This is different to existing schemes in the sense that it has more very short runs but at the same time it grows faster.

Note that restarts in our implementation only refer to a reset of the current assignment. In contrast, they do not imply a reset of information gathered during search, e.g., how often an assertion has already been selected or the weight of an assertion. Preliminary experiments showed that it is beneficial to keep those values. Intuitively, this allows learning from previous runs to make better decisions in later ones.

Extended Neighbourhoods. As pointed out in Sect. 3, a simple neighbourhood relation is given by flipping single bits of bit-vector variables which is very similar to the neighbourhood considered in SLS solvers for SAT. It is easy to see that this neighbourhood relation already allows traversing the full search space. Nevertheless, extended neighbourhoods tailored towards bit-vectors often have advantages. We therefore included three additional moves for bit-vector variables in our algorithm: Incrementing by 1, decrementing by 1, and bitwise negation. Given a set of candidate variables $\{x_1^{[n_1]}, \ldots, x_k^{[n_k]}\}$, the neighbourhood then is of size $N' = \Sigma_i(n_i + 3)$. Considering the fact that n_i often is 16 or 32 in bit-vector applications, the overhead is relatively small and usually outweighted by the benefit of permitting those natural bit-vector moves. We also tried implementing other moves, such as shifts by 1, multiplication by 3, or unary minus, but could not further improve performance by doing so, in general. For future work, it will be interesting to combine our approach with techniques used in the context of *Variable Neighbourhood Search (VNS)* (Mladenović 1995; Hansen, Mladenovi, and Moreno Prez 2008). Preliminary experiments showed promising results.

Example 3 *We left Example 2 at one of the two states:*

$$(a)\ [1, \ldots, 1, 0] = [1, \ldots, 1, 1], \textit{ or}$$
$$(b)\ [0, \ldots, 0, 1] = [0, \ldots, 0, 0],$$

both of which are very close to solutions if our neighbourhood considers increment or decrement moves, e.g., $x + 1$ or $y - 1$ in (a), and $x - 1$ or $y + 1$ in (b). Thus we finally arrive at a solution for this example in only two moves. Note that the number of moves does not depend on the size of the vectors; n separate bit-flip moves are replaced by two more complex moves.

5 Experimental Results

To evaluate the performance of our algorithm, we ran experiments on two different sets of benchmarks. The first benchmark family is the QF_BV benchmark set, which can be found in the SMT-LIB and is also part of the SMT Competition. The QF_BV benchmark set is a huge and broad collection of benchmarks, consisting of many smaller families and is the standard reference for measuring the performance of bit-vector solvers. We ran Z3 (De Moura and Bjørner 2008) on the full benchmark set of 33068 instances and removed all those which Z3 proved to be unsatisfiable within 1200 seconds. From the remaining 11715 instances, we further filtered out those 4543 formulas, that were shown to be satisfiable only by using pre-processing techniques. This left us with a total of 7498 instances in the QF_BV set for the following experiments. A second benchmark family is given by the SAGE2 benchmark set. Those problems were generated as part of the SAGE project at Microsoft (Godefroid, Levin, and Molnar 2008), describing some testcases for automated whitebox fuzz testing. Older benchmarks from the SAGE project can also be found as part of the QF_BV benchmark set. The SAGE2 set consists of 8017 instances (filtered out of original 17920 instances, 9903 were shown to be unsatisfiables within 1200 seconds, none were solved by preprocessing only) which are known to be hard for state-of-the-art SMT solvers. All experiments were run on a Windows HPC cluster of dual Quad-Xeon (E54xx) machines, 16 GB RAM, and used a time limit of 1200 seconds.

We compared our new solver BV-SLS to the most recent version of the state-of-the-art SMT solver Z3, which is based on bit-blasting and then running a CDCL SAT solver on the propositional encoding. For all benchmarks, we used the default configuration of BV-SLS: All variables are initialized to 0, candidate selection occurs using the UCB scheme, constants are set to $c_1 = 0.5$, $c_2 = 20$, $c_3 = 0.025$, $c_4 = 100$, $wp \approx 0.1$, $sp \approx 0.05$, and it uses the extended neighbourhood relation that additionally allows increment by 1, decrement by 1, and bitwise negation.

To evaluate the direct benefit of using bit-vector information for SLS, we ran several state-of-the-art SLS solvers for SAT on propositional encodings of our benchmarks as CNF. To obtain those encodings, we used the bit-blasting component of Z3. This conversion was done together with preprocessing in advance to the experiments (also using a time limit of 1200 seconds). We did not add this to the actual runtime of the solvers, assuming that the input is directly given as a pre-processed CNF. In theory, this gives an advantage to the SAT solvers. Further, CNF conversion did not succeed for all instances. This was either because Z3 ran out of memory (M/o) or because it did simply not terminate in the given time limit (T/o). In total, CNF conversion produced 21 M/o and 75 T/o results for QF_BV, and 29 T/o results for SAGE2. We considered those instances as not being solved by the SAT solvers, since it was not feasible to obtain a CNF representation in the first place. Z3, using bit-blasting, could not solve any of those either. BV-SLS was also not able to solve any of the corresponding QF_BV instances, but found a solution in 13 cases for the SAGE2 formulas.

	QF_BV	SAGE2
CCAnr	**5409**	64
CCASat	4461	8
probSAT	3816	10
Sparrow	3806	12
VW2	2954	4
PAWS	3331	**143**
YalSAT	3756	142
Z3 (Default)	7173	5821
BV-SLS	**6172**	**3719**

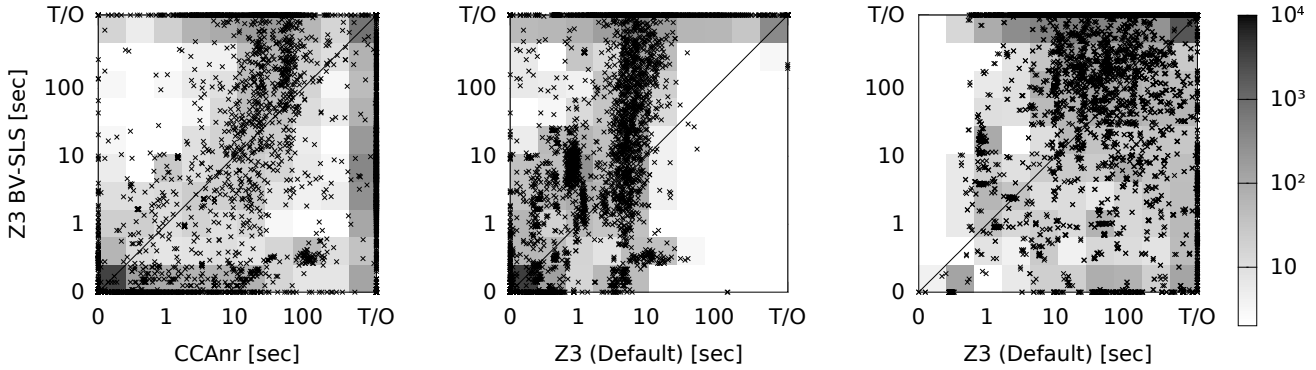
Table 1: Number of solved instances.

As SLS SAT solvers, we used several versions of CCA (Cai and Su 2012), probSAT (Balint and Schöning 2012), Sparrow (Balint and Fröhlich 2010), YalSAT (Biere 2014), and the implementations of PAWS (Thornton et al. 2004) and VW2 (Prestwich 2005) in UBCSAT (Tompkins and Hoos 2005). CCASat, probSAT, and Sparrow have consistently achieved good results over the last SAT Competitions. CCAnr, PAWS and VW2 are known for performing particularly well on some application benchmarks. YalSAT was among the few 'pure' SLS solvers (i.e. not using a CDCL component) that produced good results in the application track of the latest SAT Competition. We also tried to include Sattime2014r (Li and Li 2012), but encountered difficulties porting it to Windows. However, Sattime2014r had very similar performance to YalSAT in the application track of the SAT Competition 2014.

The number of solved instances is given in Table 1. The scatter plots and heat maps in Figs. 2a, 2b, and 2c provide details about the runtime behavior of our implementation. All solvers were run once (i.e. with one seed) per instance.

6 Discussion

The results from Sect. 5 provide several insights. First of all, comparing our SLS algorithm on the bit-vector respresentation with SLS solvers for SAT showed that we can actually profit from using additional word-level information, especially on the SAGE2 benchmark set (Tab. 1). This is particularly interesting in the context of SAT solvers. While SLS solvers for SAT are known to perform well on randomly generated formulas and, sometimes, on hard combinatorial benchmarks, CDCL solvers usually perform much better on so-called *structured* formulas, often coming from practical problems in industry which have been reencoded into propositional logic. This is often attributed to the fact that CDCL solvers are able to learn during search and make inferences, extracting this kind of original structure from a propositional formula. In contrast, most SLS solvers only use very local information. Exploiting structure for SLS SAT solvers has



(a) QF_BV (b) QF_BV (c) SAGE2

Figure 2: Scatter plots and heat maps comparing BV-SLS to CCAnr on QF_BV and to Z3 on both benchmark sets.

been looked at before, but did not yield in efficient solvers so far. Our results clearly show that SLS solvers can actually profit from using structural information during search.

Although there is still a gap between the average performance of our solver compared to state-of-the-art SMT solvers based on bit-blasting and CDCL (Tab. 1), our approach can actually outperform Z3 on several instances, especially among those contained in the SAGE2 benchmark set (Fig. 2c). This is interesting from two different points of view. First, our algorithm is a completely new approach, which has not yet had several years of research, development and tuning that CDCL algorithms have seen. It is very likely that data structures, implementation and heuristics can still be improved easily for our approach. For example, improvements might be found by adopting further techniques which have already been applied successfully in SAT solving or by using more complex heuristics that are not possible to realize in the propositional case. Since SLS solvers are known to be very sensitive with regard to their parameters, automated configuration, as applied for SAT (Hutter et al. 2009), could be beneficial for our approach too. Second, combinations of our algorithm with existing SMT solvers seem promising, because both kind of solvers often perform well on distinct kind of problems (Fig. 2c). Simply running an SLS component for a very short time before the actual SMT solver could already help finding solutions for many additional problems, potentially improving the state-of-the-art in SMT by SLS. In a further step, solvers could also start to exchange information between each other, as it has already been tried for SLS and CDCL solvers for SAT (Kroc et al. 2009), e.g. by initializing the VSIDS values of the CDCL solver according to information gained by a previous SLS run.

Another possibility for future work is the extension of our algorithm to allow other theories apart from bit-vectors. As described in Sect. 3, the underlying architecture of our algorithm is very general. The only time we actually use bit-vector information is in the definition of the score function for bit-vector expressions and the neighbourhood relation. All other components, as described in Sect. 4, as well as their improvements by techniques known from SAT solving, only take into account the Boolean part of a given formula. To allow dealing with arbitrary other theories, it is sufficient to provide a score function for the theory expressions as well as a neighbourhood relation on the theory variables.

7 Related Work

(Griggio et al. 2011) define the *WalkSMT* algorithm, which uses an SLS solver for the Boolean abstraction of a given problem, but they do not exploit SLS on the theory level. An additional theory solver is used to check satisfying Boolean assignments for theory consistency and, if they are inconsistent, to refine the abstraction in a lazy way.

(Naveh 2004) present a stochastic CSP solver that applies a bit-string encoding and then uses a stochastic search algorithm. However, their description is rather high-level and no concrete implementation is given. The most significant difference to our work can be found in the fact that we explicitely look at the problem from an SMT perspective. By doing so, we are able to successfully lift many sophisticated techniques from SAT solving to our approach. The experimental evaluation in (Naveh 2004) is only performed on a limited set of crafted benchmarks. By using the full QF_BV benchmark for our evaluation, we provide a more detailed picture of the overall performance of our solver.

8 Conclusion

In this paper, we proposed an approach towards bridging the gap between SMT and SLS. We presented a novel SLS algorithm to solve bit-vector formulas directly on the theory level. Furthermore, we explained how several techniques used in SLS solvers for SAT can be lifted to the SMT level and gave experimental results, confirming the benefit of applying local search directly on the bit-vector representation instead of using a propositional encoding. This gave new insights into the importance of exploiting problem structure also in SLS solvers for SAT. While there is still a gap in performance compared to state-of-the-art bit-vector solvers, our approach outperforms Z3 on many instances of practical relevance. This offers interesting possibilities in combining our solver with existing approaches, potentially improving the performance of both.

References

Agrawal, R. 1995. Sample mean based index policies with o(log n) regret for the multi-armed bandit problem. *Advances in Applied Probability* 27(4):1054–1078.

Balint, A., and Fröhlich, A. 2010. Improving stochastic local search for SAT with a new probability distribution. In *Proc. SAT'10*.

Balint, A., and Schöning, U. 2012. Choosing probability distributions for stochastic local search and the role of make versus break. In *Proc. SAT'12*.

Balint, A.; Biere, A.; Fröhlich, A.; and Schöning, U. 2014. Improving implementation of SLS solvers for SAT and new heuristics for k-SAT with long clauses. In *Proc. SAT'14*.

Barrett, C.; Sebastiani, R.; Seshia, S. A.; and Tinelli, C. 2009. *Satisfiability Modulo Theories*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press. chapter 26, 825–885.

Barrett, C. W.; Dill, D. L.; and Levitt, J. R. 1998. A decision procedure for bit-vector arithmetic. In *Proc. DAC'98*.

Barrett, C.; Stump, A.; and Tinelli, C. 2010. The SMT-LIB standard: Version 2.0. In *SMT'10*.

Belov, A.; Järvisalo, M.; and Stachniak, Z. 2011. Depth-driven circuit-level stochastic local search for SAT. In *Proc. IJCAI'11*, 504–509.

Biere, A. 2014. Yet another local search solver and Lingeling and friends entering the SAT Competition 2014. In *Proc. SAT Competition 2014*, 39–40.

Bjørner, N., and Pichora, M. C. 1998. Deciding fixed and non-fixed size bit-vectors. In *Proc. TACAS'98*, 376–392.

Brummayer, R., and Biere, A. 2009. Boolector: An efficient smt solver for bit-vectors and arrays. In *Proc. TACAS'09*.

Bruttomesso, R., and Sharygina, N. 2009. A scalable decision procedure for fixed-width bit-vectors. In *Proc. ICCAD'09*, 13–20.

Bruttomesso, R.; Cimatti, A.; Franzén, A.; Griggio, A.; and Sebastiani, R. 2008. The MathSAT SMT solver. In *Proc. CAV'08*, 299–303.

Cai, S., and Su, K. 2012. Configuration checking with aspiration in local search for SAT. In *Proc. AAAI'12*.

Cyrluk, D.; Möller, O.; and Rue, H. 1997. An efficient decision procedure for a theory of fixed-sized bitvectors with composition and extraction. In *Proc. CAV97*, 60–71.

De Moura, L., and Bjørner, N. 2008. Z3: an efficient SMT solver. In *Proc. TACAS'08/ETAPS'08*, 337–340.

De Moura, L., and Bjørner, N. 2009. Satisfiability modulo theories: An appetizer. In *Proc. SBMF'09*, 23–36.

de Moura, L. M., and Jovanovic, D. 2013. Model-driven decision procedures for arithmetic. In *Proc. SYNASC'13*.

Drechsler, R.; Junttila, T.; and Niemelä, I. 2009. *Non-Clausal SAT and ATPG*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press. chapter 21, 655–693.

Franzén, A. 2010. *Efficient Solving of the Satisfiability Modulo Bit-Vectors Problem and Some Extensions to SMT*. Ph.D. Dissertation, University of Trento.

Godefroid, P.; Levin, M. Y.; and Molnar, D. A. 2008. Automated whitebox fuzz testing. In *Proc. NDSS'08*.

Griggio, A.; Phan, Q.-S.; Sebastiani, R.; and Tomasi, S. 2011. Stochastic local search for SMT: Combining theory solvers with WalkSAT. In *Proc. FroCoS'11*, 163–178.

Hansen, P.; Mladenovi, N.; and Moreno Prez, J. 2008. Variable neighbourhood search: methods and applications. *4OR* 6(4):319–360.

Hoos, H., and Stützle, T. 2005. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann.

Hutter, F.; Hoos, H. H.; Leyton-Brown, K.; and Stützle, T. 2009. ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research* 36:267–306.

Järvisalo, M.; Junttila, T. A.; and Niemelä, I. 2005. Unrestricted vs restricted cut in a tableau method for boolean circuits. *Ann. Math. Artif. Intell.* 44(4):373–399.

Kovásznai, G.; Fröhlich, A.; and Biere, A. 2012. On the complexity of fixed-size bit-vector logics with binary encoded bit-width. In *Proc. SMT'12*, 44–55.

Kroc, L.; Sabharwal, A.; Gomes, C. P.; and Selman, B. 2009. Integrating systematic and local search paradigms: A new strategy for MaxSAT. In *Proc. IJCAI'09*.

Li, C. M., and Li, Y. 2012. Satisfying versus falsifying in local search for satisfiability. In *Proc. SAT'12*, 477–478.

Luby, M.; Sinclair, A.; and Zuckerman, D. 1993. Optimal speedup of Las Vegas algorithms. *Information Processing Letters* 47:173–180.

Mladenović, N. 1995. A variable neighborhood algorithm – a new metaheuristics for combinatorial optimization. In *Abstracts of Papers Presented at Optimization Days. Montral*.

Naveh, Y.; Rimon, M.; Jaeger, I.; Katz, Y.; Vinov, M.; s Marcu, E.; and Shurek, G. 2007. Constraint-based random stimuli generation for hardware verification. *AI Magazine* 28(3):13–30.

Naveh, Y. 2004. Stochastic solver for constraint satisfaction problems with learning of high-level characteristics of the problem topography. In *Proc. LSCS'04*, 17.

Pham, D. N.; Thornton, J.; and Sattar, A. 2007. Building structure into local search for sat. In *Proc. IJCAI'07*, 2359–2364.

Prestwich, S. 2005. Random walk with continuously smoothed variable weights. In *Proc. SAT'05*, 203–215.

Selman, B.; Kautz, H. A.; and Cohen, B. 1994. Noise strategies for improving local search. In *Proc. AAAI'94*, 337–343.

Thiffault, C.; Bacchus, F.; and Walsh, T. 2004. Solving non-clausal formulas with DPLL search. In *Proc. CP'04*.

Thornton, J.; Pham, D. N.; Bain, S.; and Jr., V. F. 2004. Additive versus multiplicative clause weighting for SAT. In *Proc. AAAI'04*, 191–196.

Tompkins, D. A. D., and Hoos, H. H. 2005. UBCSAT: An implementation and experimentation environment for SLS algorithms for SAT and MAX-SAT. In *Proc. SAT'04*.

Yuan, J.; Pixley, C.; and Aziz, A. 2006. *Constraint-based verification*. Springer.