CirCUs: A Hybrid Satisfiability Solver[⋆]

HoonSang Jin and Fabio Somenzi

University of Colorado at Boulder
{Jinh,Fabio}@Colorado.EDU

Abstract. CirCUs is a satisfiability solver that works on a combination of an And-Inverter-Graph (AIG), Conjunctive Normal Form (CNF) clauses, and Binary Decision Diagrams (BDDs). We show how BDDs are used by CirCUs to help in the solution of SAT instances given in CNF. Specifically, the clauses are sorted by solving a hypergraph linear arrangement problem. Then they are clustered by an algorithm that strives to avoid explosion in the resulting BDD sizes. If clustering results in a single diagram, the SAT instance is solved directly. Otherwise, search for a satisfying assignment is conducted on the original clauses, enhanced with information extracted from the BDDs. We also describe a new decision variable selection heuristic that is based on recognizing that the variables involved in a conflict clause are often best treated as a related group. We present experimental results that demonstrate CirCUs's efficiency especially for medium-size SAT instances that are hard to solve by traditional solvers based on DPLL.

1 Introduction

Different representations of Boolean functions have peculiar strengths in regard to satisfiability (SAT) problems. Conjunctive Normal Form (CNF) is often used because it can be manipulated efficiently and because constraints of various provenance are easily translated into it. Boolean circuits, especially semi-canonical ones like the And-Inverter Graph (AIG) [24], allow a variety of simplification techniques that may significantly speed up subsequent analyses. For other representations, like the Disjunctive Normal Form (DNF) and Binary Decision Diagrams (BDDs) [6], the hurdle lies in converting the problem specification into the required form; if this can be accomplished, satisfiability is then trivial. In particular, with BDDs, determining whether a function is satisfiable requires constant time, while a satisfying assignment, if it exists, can be found in $O(n)$ time, where n is the number of variables. Since converting a Boolean circuit into a BDD may incur an exponential blow-up, naive application of BDDs to SAT lacks robustness. On the other hand, there exist numerous cases in which a proper mix of canonical (e.g., BDDs) and non-canonical representations (e.g., CNF or AIG) is very beneficial [25, 8]. This is true, in particular, of SAT solvers based on search, and applied to instances for which compact search trees do not exist or are hard to find.

CirCUs is a SAT solver that accepts as input a combination of an AIG, CNF clauses, and BDDs. Rather than converting all into one form as a preprocessing step, CirCUs operates on all three representations, transforming, when appropriate, parts of the input from one of them to another. For instance, in Bounded Model Checking (BMC) [4] applications, CirCUs reads the input as an AIG with additional constraints given as clauses, and transforms part of the AIG into BDDs, so that it may apply powerful implication and conflict analysis algorithms [23, 21]. The conflict clauses, on the other hand, are recorded in CNF form as suggested in [13]. Because of this ability to operate on multiple representations, we call CirCUs a *hybrid* SAT solver.

In this paper we discuss how CirCUs handles SAT instances given in CNF. After a review of related work in Sect. 2, in Sect. 4, we show how the clauses may be "conditioned" with the help of BDDs so as to allow the solution of some hard, though not very large, problems. The conditioning consists of building BDDs from the clauses in such a way that resource limits are not exceeded. This implies that more than one BDD may be built. If that is the case, CNF clauses are extracted from the BDDs to replace the original ones.

Section 5 presents a new decision variable selection heuristic, which is based on the observation that variables appearing in one conflict clause should be treated as a related group. In Sect. 6 we present empirical evidence that for mid-size hard instances, CNF conditioning is very effective, and that our decision variable heuristic consistently improves over the VSIDS rule of [32]. Finally, we draw conclusions in Sect.7.

[⋆] This work was supported in part by SRC contract 2003-TJ-920.

2 Related Work

Considerable work has been done in which constraints are represented by a collection of BDDs. In symbolic model checking, the transition relation is often represented in such an implicitly conjoined form [38, 7, 19, 30, 17, 22]. The partitioned representation was also applied to the problem of minimum-cost satisfiability in [20]. In our work we leverage several techniques from this body of literature, especially from [22].

More recently, there has been considerable interest in BDD-based techniques for the SAT problem. Gupta el al. [15] proposed BDD-based learning while solving Bounded Model Checking (BMC [4]) instances with a circuit SAT solver. The BDDs are used to supplement conflict-learned clauses. They are created from portions of the circuit that defines the BMC instance. Their approach is similar to our approach in the sense that they use BDD to extract helpful CNF from it. On the other hand, we do no assume the existence of a circuit, and our algorithms are different.

Damiano and Kukula [9] replace clauses with BDDs in a classical DPLL solver, while in [12], the authors propose the method that uses BDDs to precompute complete lookahead information to drive the search. This is done by converting each BDD into a finite state machine that reads assignments to the BDD inputs and outputs implied values. During a preprocessing phase, Franco et al. use *strengthening* to infer additional literals and equivalences, since their BDD is highly localized because of BDD blow-up. The search is then conducted on the modified BDDs. By contrast, the technique we discuss in this paper either solves the SAT instance without search, or eventually operates on CNF that has been possibly enhanced using the extracted BDDs.

3 Preliminaries

We consider three ways of representing a Boolean function. The first is a Boolean circuit, that is, a directed acyclic graph whose nodes correspond to input variables and Boolean gates. Specifically, we use a form of Boolean circuit called And-Inverter Graph (AIG) in which each node's function is one of $x \wedge y$, $x \wedge \neg y$, $\neg x \wedge y$, and $\neg x \wedge \neg y$. An AIG contains no isomorphic subgraphs; for this reason, it is called *semicanonical*.

The second representation is Conjunctive Normal Form (CNF). A CNF formula is a set of *clauses*; each clause is a set of *literals*; each literal is either a variable or its complement. The function of a clause is the disjunction of its literals, and the function of a CNF formula is the conjunction of its clauses.

The last representation of Boolean functions is Binary Decision Diagrams (BDDs). A BDD is a Boolean circuit such that each node is labeled by either a Boolean constant (terminal node) or a variable (internal node). Each internal node has two children, T and E. The function of an internal node labeled by v is defined recursively by $(v \wedge f(T)) \vee (\neg v \wedge f(E))$, where $f(T)$ and $f(E)$ are the functions of T and E. A reduced BDD is one in which there are no isomorphic subgraphs, and no node has identical children. (Such nodes are redundant.) A BDD is ordered if the variables encountered along all paths from root to leaves respect a fixed order. Reduced, ordered BDDs are canonical: for a given variable order, two functions are the same if and only if they have the same BDD [6]. We shall refer to reduced, ordered BDDs simply as BDDs. Another form of diagrams that are useful in manipulating Boolean functions are Zero-suppressed BDDs (ZDDs). The difference between BDDs and ZDDs is that in the former, nodes with identical children are removed, while in the latter nodes whose T child is the constant 0 are removed. ZDDs are usually more concise than BDDs when representing sets of clauses (each clause corresponding to a path in the diagram). BDDs, on the other hand, are usually better when representing the functions themselves.

CirCUs is a SAT solver based on the DPLL procedure [11, 10] and conflict clause recording [35, 41, 32, 14]. It is built on top of VIS [5, 40], and uses the CUDD package [36] for BDD and ZDD manipulations. Figure 1 describes the core of the decision procedure, whose input is an AIG, a set of CNF clauses, and a set of BDDs.

The pseudo-code of DPLL procedure is presented in Fig. 1. Procedure CHOOSENEXTASSIGNMENT checks the implication queue. If the queue is empty, the procedure makes a *decision*: it chooses one unassigned variable and a value for it, and adds the assignment to the implication queue. If none can be found, it returns `false`. This causes DPLL to return an affirmative answer, because the assignment to the variables is complete and no conflict is detected. If a new assignment has been chosen, its implications are added by DEDUCE to the queue. If the implications yield a conflict, this is analyzed to produce two important results. The first is a clause implied by the given circuit and objectives. This *conflict clause* is added to the clauses of the circuit. Termination relies on conflict clauses, because they prevent the same variable assignment

```
DPLL() {
    while (CHOOSENEXTASSIGNMENT() == FOUND)
        while (DEDUCE() == CONFLICT) {
            blevel = ANALYZECONFLICT();
            if (blevel $\leq$ 0) return UNSATISFIABLE;
            else BACKTRACK(blevel);
        }
    return SATISFIABLE;
}
```

Fig. 1. DPLL algorithm

from being tried more than once. The second result of conflict analysis is the *backtracking level*: Each assignment to a variable has a *level* that starts from 0 and increases with each new decision. When a conflict is detected, the algorithm determines the lowest level at which a decision was made that eventually caused the conflict. The search for a satisfying assignment resumes from this level by deleting all assignments made at higher levels. This *non-chronological backtracking* allows the decision procedure to ignore inconsequential decisions that have provably no part in the conflict being analyzed.

The pseudo-code of Fig. 1 is essentially the same used to describe CNF SAT solvers like GRASP and Zchaff. However, in CirCUs all operations are carried out on the three Boolean function representations at once. CNF clauses and BDDs are connected to the AIG so that propagation of implications and conflict analysis proceed seamlessly on all of them. The algorithm uses a common assignment stack and implication queue. The decision variable selection is also common. In particular, the DVH heuristic of Sect. 5 is used by CirCUs regardless of the mix of function representations. The specific implication and conflict analysis algorithms for AIG, clauses, and BDDs are described in [24, 32, 23].

When the input is in the form of an AIG, replacing parts of it by BDDs allows CirCUs to reduce the number of decisions and conflicts without slowing down implication too much. In this paper, we consider the case in which the input is a set of clauses. The strategy of [23], which replaces *fanout-free* subcircuits of the AIG with BDDs, is not applicable. Instead, we try to improve the given CNF as described in Sect. 4.

4 CNF Conditioning

For hard CNF SAT instances with moderate numbers of variables and clauses, it is often advantageous to *condition* the given set of clauses. In the following, we describe the approach implemented in CirCUs.

A *hypergraph* $G = (V, H)$ consists of a set of vertices V and a multiset of hyperedges H. Each hyperedge is a subset of V. A *linear arrangement* of G is a bijection $\alpha : V \rightarrow \{1, \ldots, |V|\}$.

A set of Boolean functions can be regarded as a hypergraph by associating variables to vertices and functions to hyperedges. A hyperedge connects all the variables appearing in the function to which it is associated. Linear arrangement has been used in [1, 2] to derive variables orders for both BDD construction and SAT. Our use is closer in spirit to the one of [22], in which the objective is to derive a good order for the conjunction of the functions.

We compute a linear arrangement by *force-directed* (or *quadratic*) placement [33], as done in [2]. Given a linear arrangement α_i, the algorithm computes the *center of mass* of hyperedge $h \in H$ thus:

$$COM(h) = \frac{\sum_{v \in h} \alpha_i(v)}{|h|} \ . \qquad (1)$$

The center of mass of a vertex is computed as the average of the centers of mass of all hyperedges incident on the vertex. Finally, α_{i+1} is obtained by sorting vertices according to their centers of mass. The process is iterated starting from an initial given arrangement α_0 until the cost function stops decreasing, or until the alloted computational resources are exhausted. The cost function is the sum of the hyperedge spans, where the span of hyperedge h under arrangement α is

$$span(h) = \max_{v \in h}\{\alpha(v)\} - \min_{v \in h}\{\alpha(v)\} \ . \qquad (2)$$

Once the final vertex arrangement is determined, the order of the hyperedges is given by their centers of mass.

Once the clauses of the given CNF are sorted, if the numbers of variables and clauses do not exceed specified thresholds, the clustering algorithm of [22] is invoked to try to conjoin all clauses into one BDD. The algorithm works on a list of Boolean functions initialized to the sorted list of clauses. It selects a set of adjacent functions to be conjoined, and tries to construct a BDD for them. If the BDD can be built without exceeding a threshold on the number of nodes, it replaces the functions that were conjoined in the list. The candidates are chosen so as to favor the confinement of as many variables as possible to one cluster only. A detailed description of the algorithm can be found in [22]. The thresholds on the numbers of variables and clauses are chosen so that it is likely that all clauses will be conjoined into one BDD. When this happens, the SAT instance is solved directly.

For this purpose, the clustering algorithm iterates until no new clusters are created in one pass. At each pass, it creates a list of candidates. Each candidate is a pair of clusters. The list is ordered in decreasing order of the number of isolated variable to favor candidates that allow many variables to be quantified. (This is beneficial when trying to build one BDD from all clauses.) As a tie-breaker, the upper bound on the number of variables in the resulting cluster is used. This policy favors the creation of small clusters that may be merged in subsequent passes. If a given instance is unsatisfiable, it will result in the constant zero BDD; otherwise it will result in the constant one BDD because all variables are quantified while clustering.

To get a satisfying assignment without saving all the BDDs produced during clustering, we save the last two BDDs, so that a partial assignments can be extracted from them. We then use this partial assignment as a constraint for the CNF SAT solver. This results in a quick solution of the CNF instance because the clustering process is such that the last two BDDs tend to contain the global variables of the function.

If, on the other hand, the initial CNF is too large, or the conjunction of all clauses cannot be carried out without exceeding the resource limits, several BDDs are built, each to be used in conditioning a subset of the CNF formula. The clauses are divided into *short* (one or two literals) and *long* (more than two literals). The long clauses are conjoined in the order determined by the linear arrangement until the BDD for the resulting cluster exceeds a given size, at which point a new cluster is started. Let f be the function for such a cluster. The next step consists of conjoining all the short clauses that share at least one variable with f into a function g. Since g is implied by the original set of clauses, any function f_g such that $f_g \wedge g = f$ can replace f. Therefore, we are interested in a simple CNF representing a function from the interval $[f, f \vee \neg g]$. This is computed by the Morreale-Minato algorithm for prime and irredundant covering of a Boolean function [31, 27]. The algorithm is called on the interval $[\neg f \wedge g, \neg f]$, and DeMorgan's Laws are applied to the resulting DNF.

The result of the Minato-Morreale algorithm is computed as a Zero-Suppressed BDD (ZDD) [28]. The clauses are then obtained by enumeration of the paths of the ZDD. Since the computed CNF is not guaranteed to have the minimum number of clauses, it is possible that more clauses be extracted than were used to produce f. If this happens, the process is abandoned, and the original clauses are used instead. Even in such a case, the construction of the BDD may be helpful: If a variable occurring in the clauses conjoined to obtain f does not occur either in f or in the other clauses, then it can be universally quantified from the original clauses.

The final step of conditioning consists of extracting all short clauses from the function in the interval $[f, f \vee \neg g]$ chosen by the Morreale-Minato algorithm. This is accomplished by a single traversal of each BDD, during which the short clauses of a BDD with top node ν are obtained from the short clauses of the children of ν [37]. The procedure extends the one for unit clauses of [20]. Both procedures, as well as the Morreale-Minato algorithm, are implemented in CUDD [36].

5 Decision Variable Selection

The choice of the decision variables has a large impact on the run time of the DPLL procedure. Hence, considerable attention has been devoted to the problem. (See, for instance, [34, 26, 18].) Many rules have been proposed that are based on the frequency of literals in unresolved clauses; for instance, the Dynamic Largest Individual Sum (DLIS) heuristic of GRASP [35]. Chaff's VSIDS rule [32] disregards whether a clause is resolved in the interest of speed. It also introduces the notion that recently generated conflict clauses should be given more weight in the choice of the next variable. The VSIDS rule increases the score of a literal whenever a clause containing that literal is added to the database. Scores are periodically halved to give more weight to recently added conflict clauses. The literal with the highest score is chosen whenever a decision must be made.

Though non-chronological backtracking helps the DPLL procedure to recover from poorly chosen decision variables, it is only effective once a conflict has been detected. Suppose a conflict clause γ involves variables at decision levels $d_0 \ldots, d_k$. Ideally, one would have $d_{i+1} = d_i + 1$ for $0 < i < k$. Otherwise, the work done in propagating the effects of the irrelevant intervening decisions is wasted. Increasing the scores of the variables in γ as done in VSIDS helps because the variables at the higher decision levels will be chosen earlier in the sequel of the search. However, the variables in the conflict clause at the lower decision levels will also be chosen earlier. More importantly, it may take several conflicts for a group of related variables to have similar scores if their initial scores are sufficiently different. In BerkMin [14] this problem is addressed by choosing the decision literal from the unassigned variables in the most recent conflict clause that is unsatisfied. The limitation of this approach is that a conflict clause's ability to cause its literals to be treated like a related group is lost as soon as it is no longer the most recent unsatisfied clause.

By contrast, the approach followed in CirCUs is the following. Suppose a new conflict clause $\gamma = \{l_0, \ldots, l_k\}$ is generated. Suppose that d_i is the decision level of l_i, and, w.l.o.g., that $d_i < d_{i+1}$ for $0 < i < k$. The scores of all literals in the clause are incremented by one with the exception of the literal l_k at the current decision level, whose score is set equal to one less than the score of l_{k-1}. Boosting the score of the most recent decision variable causes the relation between l_{k-1} and l_k to be recorded in the scores, producing a longer lasting effect than in the BerkMin case. We call the new heuristic Deepest Variable Hiking (DVH).

Figure 2 shows two series of decisions to illustrate the advantages of the DVH heuristic. Each circle represents a decision made by a score-based heuristic and the dark circles represent decisions whose implications are involved in the conflict-learned clause. We assume that the conflict occurs in both cases at decision level d_{i+4}.

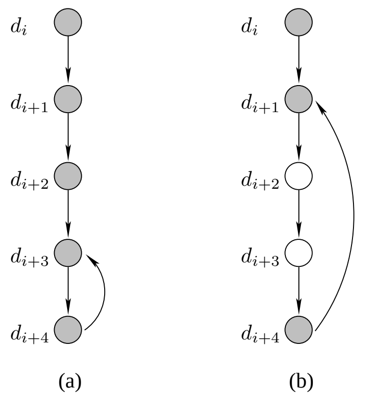

Fig. 2. Two examples of decision

If all the decisions are relevant to the current conflict, then the conflict-learned clause will contain literals implied by all previous decisions as shown in Fig. 2 (a). In this case we backtrack to the decision level d_{i+3}. If, however, irrelevant intervening decisions were made, such as those at levels d_{i+2} and d_{i+3} in Fig. 2 (b), then backtracking will be to a lower decision level like d_{i+2} in the example. Since the decisions made at level d_{i+2} and d_{i+3} are not related to the conflict-learned clause, the cost of BCP for those decisions is wasted. Even though the current scores of the decision variables at levels d_{i+2} and d_{i+3} are higher than the one at level d_{i+4}, the variable of d_{i+4} is a better choice. Thanks to the DVH decision heuristic, we can avoid the waste of effort even when exploring subspaces in which the clause derived from the current conflict is satisfied.

Since we increase the score of the variable of level d_{i+4} to one less than the score of the variable at level d_{i+2}, the score-based decision heuristic treats them as a related group. If they are not relevant variables for the rest of the search, then the periodic decay will reduce their scores thereby decreasing their importance automatically.

Suppose the data inputs of a multiplexer are driven by two subcircuits having disjoint supports and that the sel signal selects which circuitry is connected to the output of multiplexer. Once sel is decided then the variables in the unselected circuitry can be ignored since they no longer affect the value of the circuit. Silva et al. address this problem in [34] and a related approach is presented in [3]. Gupta et al. [16] use circuit SAT to identify the unobservable gates and disable the corresponding clauses in the CNF database. In [39], the author proposes an efficient translation of CNF from circuits that considers unobservable gates. Even though the DVH heuristic does not explicitly address unobservable gates in a circuit, it does help when such gates are present thanks to its ability to increase the dynamics of decision heuristics. For instance, once the sel signal is assigned and we find a conflict from the circuitry feeding one of the inputs to the multiplexer, the DVH heuristic helps the decision procedure focus on the part where the conflict was found.

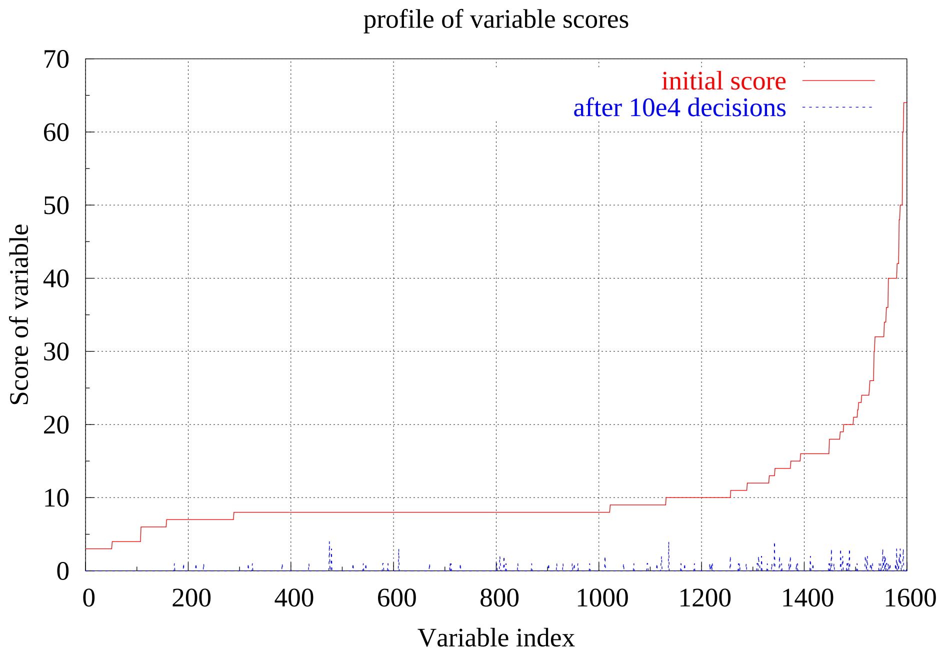

Fig. 3. Scores of variables while solving C880.cnf using VSIDS

The VSIDS rule as implemented in Zchaff halves the literal scores once every so many decisions. If the ratio of decisions to conflicts is large, most scores decay to 0. In Fig. 3 We show the profile of variable scores produced by VSIDS for C880.cnf, which is one of SAT 2003 industrial benchmark. In the figure, one can find two lines. They are the profiles of initial scores and the scores after 10000 decisions are made. The variables are sorted according to their initial scores. One can see from the figure that not only most variables have scores of zero, but also the few non-null scores take only a very limited number of values.

When this is the case, variables are chosen on the basis on insufficient information. The DVH heuristic of CirCUs tries to overcome this problem by reducing the halving frequency if the ratio of decisions to conflicts is too high.

6 Experimental Results

We performed two sets of experiments to assess the impact of the techniques described in Sections 4 and 5. The first set studies the effects of CNF conditioning on the speed of the SAT solver for 89 examples from the hand-made category of the SAT2003 benchmark set. These examples are not very large—up to 2,000 variables and 60,000 clauses—but some of them are hard for many solvers. The experiments were run on a 2.4 GHz Pentium IV with 500 MB of RAM running Linux. Runs longer than 2,000 s were terminated.

Table 1 shows the examples that were used for the CNF conditioning experiments. The columns comparing CPU time show that CirCUs achieves huge improvements over Zchaff. We also show the numbers of completed instances with in parenthesis.

Table 1. Examples from hand-made category of the SAT2003 benchmark set

Benchmark name	Number of instances	CPU time	
		Zchaff	CirCUs
bevan/marg*	14	4330.36(12)	1.33(14)
bevan/urqh1c*	13	14638.84(8)	6.77(13)
bevan/urqh*	12	20028.83(2)	4.81(12)
markstrom/mm*	8	1047.22(8)	1262.46(8)
purdom/	4	3160.07(3)	1720.21(4)
simon/sat02/x1*	19	38000.00(0)	6.97(19)
simon/sat02/x2*	9	18000.00(0)	3.38(9)
simon/sat02/Urquhart*	10	20000.00(0)	4.59(10)

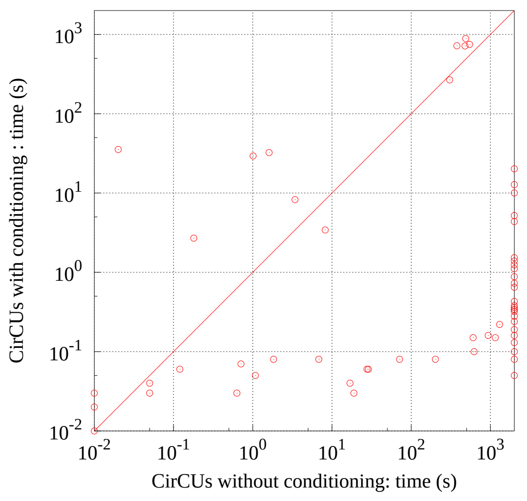

Fig. 4. Effects of CNF conditioning

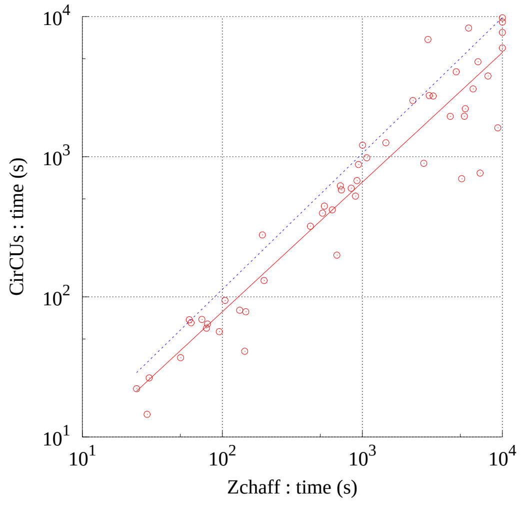

Fig. 5. DVH versus VSIDS

Figure 4 shows a log-log scatterplot that compares CirCUs runtimes with and without CNF conditioning. One can easily identify two groups of instances. Those for which reshaping is effective, including those for which a monolithic BDD can be built, and those near or above the main diagonal, for which conditioning does not appreciably change the CNF. In the latter group, the overhead of constructing the BDDs is not recovered.

It should be pointed out that sorting the clauses by linear arrangement and applying the clustering algorithm of [22] are fundamental for efficiency. Many of the examples that terminate in a few seconds with the algorithm of Sect. 4 cannot be completed otherwise.

The second set of experiments compares the DVH variable selection heuristic of Sect. 5 to the popular VSIDS heuristic used in Chaff. We compared three sets of 50 runs: Zchaff [32], CirCUs with VSIDS, and CirCUs with DVH. CNF conditioning was not used in these experiments that were performed on a 1.7 GHz Pentium IV with 2 GB of RAM running Linux. The timeout was set at 10,000 s. The SAT instances are derived from BMC experiments on models from the VIS Verilog benchmark collection [40].

The results are summarized in Fig. 5. The log-log scatterplot shows the points comparing CirCUs with DVH to Zchaff. The two straight lines are regression curves of the form $y = \kappa \cdot x^{\eta}$, where κ and η are obtained by least-square fitting. The upper line is for the comparison of CirCUs with VSIDS to Zchaff; it is provided for calibration. It shows that the two solvers are quite comparable in performance when using the same decision heuristic. The lower line is for CirCUs with DVH vs. Zchaff. The separation of the two lines indicates that DVH provides a speedup of almost 2 over VSIDS.

Our implementation of BerkMin's heuristic did not work so well, but lack of access to the source code means that we cannot be sure our interpretation of it is faithful to the original.

7 Conclusions

We have presented CirCUs, a hybrid SAT solver that operates on an And-Inverter Graph, a set of CNF clauses, and a set of BDDs. We have described the approach used to speed up the solver when the input is in CNF form. By converting the clauses into one or more BDDs, we are often able to either solve the problem directly, or extract an improved CNF formula. We have shown the effectiveness of this strategy on small-but-hard examples from the SAT2003 benchmark set.

We have also presented an improved decision variable selection heuristic, and shown its effectiveness by comparing it to the popular VSIDS heuristic of Zchaff.

Our results demonstrate the usefulness of allowing the SAT solver to operate on multiple representations of the input problem. We intend to explore more applications of this principle, and we are busy improving the efficiency of the current implementation. For instance, we plan to compare the current force-directed placement approach to the MLP algorithm [29].

References

1. F. A. Aloul, I. L. Markov, and K. A. Sakallah. Mince: A static global variable-ordering for SAT and BDD. Presented at IWLS01, June 2001.
2. F. A. Aloul, I. L. Markov, and K. A. Sakallah. FORCE: A fast and easy-to-implement variable-ordering heuristic. In *Proceedings of the Great Lakes Symposium on VLSI*, pages 116–119, Washington, DC, Apr. 2003.
3. C. W. Barrett, D. L. Dill, and A. Stump. Checking satisfiability of first-order formulas by incremental translation to SAT. In E. Brinksma and K. G. Larsen, editors, *Fourteenth Conference on Computer Aided Verification (CAV'02)*, pages 236–249. Springer-Verlag, Berlin, July 2002. LNCS 2404.
4. A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Fifth International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'99)*, pages 193–207, Amsterdam, The Netherlands, Mar. 1999. LNCS 1579.
5. R. K. Brayton et al. VIS: A system for verification and synthesis. In T. Henzinger and R. Alur, editors, *Eighth Conference on Computer Aided Verification (CAV'96)*, pages 428–432. Springer-Verlag, Rutgers University, 1996. LNCS 1102.
6. R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, Aug. 1986.
7. J. R. Burch, E. M. Clarke, and D. E. Long. Representing circuits more efficiently in symbolic model checking. In *Proceedings of the Design Automation Conference*, pages 403–407, San Francisco, CA, June 1991.
8. J. R. Burch and V. Singhal. Tight integration of combinational verification methods. In *Proceedings of the International Conference on Computer-Aided Design*, pages 570–576, San Jose, CA, Nov. 1998.
9. R. Damiano and J. Kukula. Checking satisfiability of a conjunction of BDDs. In *Proceedings of the Design Automation Conference*, pages 818–823, June 2003.
10. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5:394–397, 1962.
11. M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the Association for Computing Machinery*, 7(3):201–215, July 1960.
12. J. Franco, M. Kouril, J. Schlipf, J. Ward, S. Weaver, M. Dransfield, and W. M. Vanfleet. SBSAT: A state-based, BDD-based satisfiability solver. In *International Conference on Theory and Applications of Satisfiability Testing (SAT 2003)*, Portofino, Italy, May 2003.
13. M. K. Ganai, P. Ashar, A. Gupta, L. Zhang, and S. Malik. Combining strengths of circuit-based and CNF-based algorithms for a high-performance SAT solver. In *Proceedings of the Design Automation Conference*, pages 747–750, New Orleans, LA, June 2002.
14. E. Goldberg and Y. Novikov. BerkMin: A fast and robust SAT-solver. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 142–149, Paris, France, Mar. 2002.
15. A. Gupta, M. Ganai, C. Wang, Z. Yang, and P. Ashar. Learning from BDDs in SAT-based bounded model checking. In *Proceedings of the Design Automation Conference*, pages 824–829, June 2003.
16. A. Gupta, A. Gupta, Z. Yang, and P. Ashar. Dynamic detection and removal of inactive clauses in SAT with application in image computation. In *Proceedings of the Design Automation Conference*, pages 536–541, Las Vegas, NV, June 2001.
17. A. Gupta, Z. Yang, P. Ashar, and A. Gupta. SAT-based image computation with application in reachability analysis. In W. A. Hunt, Jr. and S. D. Johnson, editors, *Formal Methods in Computer Aided Design*, pages 354–271. Springer-Verlag, Nov. 2000. LNCS 1954.
18. M. Herbstritt and B. Becker. Conflict-based selection of branching rules. In *Sixth International Conference on Theory and Application in Satisfiability Testing (SAT2003)*, pages 441–451, Portofino, Italy, May 2003. Springer. LNCS 2919.

19. A. J. Hu and D. Dill. Efficient verification with BDDs using implicitly conjoined invariants. In C. Courcoubetis, editor, *Fifth Conference on Computer Aided Verification (CAV '93)*, pages 3–14. Springer-Verlag, Berlin, 1993. LNCS 697.
20. S.-W. Jeong and F. Somenzi. A new algorithm for 0-1 programming based on binary decision diagrams. In T. Sasao, editor, *Logic Synthesis and Optimization*, chapter 7, pages 145–165. Kluwer Academic Publishers, Boston, MA, 1993.
21. H. Jin, M. Awedh, and F. Somenzi. CirCUs: A satisfiability solver geared towards bounded model checking. In R. Alur and D. Peled, editors, *Sixteenth Conference on Computer Aided Verification (CAV'04)*. Springer-Verlag, Berlin, July 2004. To appear.
22. H. Jin, A. Kuehlmann, and F. Somenzi. Fine-grain conjunction scheduling for symbolic reachability analysis. In *International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'02)*, pages 312–326, Grenoble, France, Apr. 2002. LNCS 2280.
23. H. Jin and F. Somenzi. CirCUs: Speeding up circuit SAT with BDD-based implications. Submitted for publication, Nov. 2003.
24. A. Kuehlmann, M. K. Ganai, and V. Paruthi. Circuit-based Boolean reasoning. In *Proceedings of the Design Automation Conference*, pages 232–237, Las Vegas, NV, June 2001.
25. A. Kuehlmann and F. Krohm. Equivalence checking using cuts and heaps. In *Proceedings of the Design Automation Conference*, pages 263–268, Anaheim, CA, June 1997.
26. P. Liberatore. On the complexity of choosing the branching literal in DPLL. *Artificial Intelligence*, 116(1–2):315–326, 2000.
27. S.-I. Minato. Fast generation of irredundant sums-of-products forms from binary decision diagrams. In *SASIMI '92*, pages 64–73, Kyoto, Japan, Apr. 1992.
28. S.-I. Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *Proceedings of the Design Automation Conference*, pages 272–277, Dallas, TX, June 1993.
29. I.-H. Moon, G. D. Hachtel, and F. Somenzi. Border-block triangular form and conjunction schedule in image computation. In W. A. Hunt, Jr. and S. D. Johnson, editors, *Formal Methods in Computer Aided Design*, pages 73–90. Springer-Verlag, Nov. 2000. LNCS 1954.
30. I.-H. Moon, J. H. Kukula, K. Ravi, and F. Somenzi. To split or to conjoin: The question in image computation. In *Proceedings of the Design Automation Conference*, pages 23–28, Los Angeles, CA, June 2000.
31. E. Morreale. Recursive operators for prime implicant and irredundant normal form determination. *IEEE Transactions on Computers*, C-19(6):504–509, June 1970.
32. M. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the Design Automation Conference*, pages 530–535, Las Vegas, NV, June 2001.
33. N. R. Quinn. The placement problem as viewed from the physics of classical mechanics. In *Proceedings of the Design Automation Conference*, pages 173–178, Boston, MA, June 1975.
34. J. P. M. Silva. The impact of branching heuristics in propositional satisfiability algorithms. In *Proceedings of the 9th Portuguese Conference on Artificial Intelligence (EPIA)*, Sept. 1999.
35. J. P. M. Silva and K. A. Sakallah. Grasp—a new search algorithm for satisfiability. In *Proceedings of the International Conference on Computer-Aided Design*, pages 220–227, San Jose, CA, Nov. 1996.
36. F. Somenzi. *CUDD: CU Decision Diagram Package*. University of Colorado at Boulder, ftp://vlsi.colorado.edu/pub/.
37. F. Somenzi and K. Ravi. Extracting simple invariants from BDDs. Unpublished manuscript, May 2002.
38. H. Touati, H. Savoj, B. Lin, R. K. Brayton, and A. Sangiovanni-Vincentelli. Implicit enumeration of finite state machines using BDD's. In *Proceedings of the IEEE International Conference on Computer Aided Design*, pages 130–133, Nov. 1990.
39. M. N. Velev. Exploiting signal unobservability for efficient translation to CNF in formal verification of microprocessor. In *In the Proceedings of the IEEE/ACM Design, Automation and Test in Europe Conference (DATE)*, Feb. 2004.
40. URL: http://vlsi.colorado.edu/$\sim$vis.
41. H. Zhang. SATO: An efficient propositional prover. In *Proceedings of the International Conference on Automated Deduction*, pages 272–275, July 1997. LNAI 1249.