A Circuit SAT Solver With Signal Correlation Guided Learning

Feng Lu, Li-C. Wang, and Kwang-Ting Cheng
Department of Electrical and Computer Engineering
University of California, Santa Barbara 93106-9560, USA

Ric C-Y Huang
Verplex Systems, Inc.
Milpitas, California 93035, USA

Abstract— Boolean Satistifiability has attracted tremendous research effort in recent years, resulting in the developments of various efficient SAT solver packages. Based upon their design architectures, researchers have tried to develop better heuristics to further improve its efficiency, by either speeding up the Boolean Constraint Propagation (BCP) procedure or finding a better decision ordering (or both). In this paper, we propose an entirely different SAT solver design concept that is circuit-based. Our solver is able to utilize circuit topological information and signal correlations to enforce a decision ordering that is more efficient for solving circuit-based SAT problem instances. In particular, for unsatisfiable circuit examples, our solver is able to achieve from 2x up to more than 75x speedup over a state-of-the-art SAT solver.

I. Introduction

The Boolean Satistifiability (SAT) Problem has been extensively studied in recent years. Popular SAT solvers [1, 2, 3, 4] are often designed based upon the Conjunctive Normal Form (CNF) representation. For many applications in CAD, applying SAT to solve a circuit-oriented problem often requires transformation of the circuit gate-level netlist into its corresponding CNF format [5]. In the circuit-to-CNF transformation, the topological ordering among the internal signals is no longer there. All signals become (input) *variables* in the CNF format.

For solving circuit-oriented CAD problems, circuit structural information can be very useful. For example, researchers have developed various SAT solvers able to utilize circuit-related information to speed up the state-of-the-art SAT solver performance [6, 7, 8].

In this paper, we present a circuit-based SAT solver that utilizes *signal topological ordering* and *signal correlations* to identify an effective decision ordering in the SAT search process. Our solver design follows three key ideas:

- If a group of signals can be identified in advance that they are highly correlated, then in the solver's decision variable selection, they should be grouped together.
- When solving a SAT problem originated from a circuit, sometimes we can solve the problem more easily by following the topological structure. This means that more efficiency gain can be obtained by pre-selecting a set of "proper" signals in the circuit, and explicitly enforcing the SAT solving process to follow their topological ordering to solve the problem.
- When making the decisions of value assignments to signals, it is more effective for the solver to select those values that are more likely to cause conflicts. Therefore, if we know how circuit signals are correlated beforehand, that information can be used to guide the value assignments in the solver.

By combining the three ideas above, we implemented a circuit-based SAT solver whose decision ordering is guided by what we call the *signal correlation learning*. Our solver includes two types of the signal correlation learning: *implicit learning* and *explicit learning* which differ in their ways to affect the decision ordering. In implicit learning, correlated signals are grouped together in the decision variable selection, and correlation information is used to guide the solver to assign values that are more likely to cause conflicts. In explicit learning, a set of K signals $\{s_1, \ldots, s_K\}$ are pre-selected based upon signal correlations. The decision ordering on these signals are enforced by their topological order. Values assignments $\{s_1 = v_1, \ldots, s_k = v_K\}$ are pre-determined in such a way that solving each sub-problem "$s_i = v_i$" ($1 \leq i \leq K$) is likely to cause conflicts. Moreover, the solver starts by *explicitly* following the topological order to solve each sub-problem in $\{s_1 = v_1, \ldots, s_k = v_K\}$, then followed by solving the original SAT problem. We call this approach the *incremental learn-from-conflict* strategy.

For unsatisfiable circuit examples, with implicit learning, our solver was able to achieve on average 3-7x speedup over the state-of-the-art SAT solver ZChaff [1, 2]. With explicit learning, this performance gain could further be enhanced from 13x up to more than 75x. For satisfiable cases that contain CNF format in their problem inputs, our circuit-based solver was not able to take the full advantage of the circuit structural information. As a result, with implicit learning, roughly 2x speedup was obtained, and with explicit learning, only comparable results were achieved. We discuss our circuit-based solver implementation and the experimental results in the rest of the paper.

The rest of the paper is organized as the following. In Section II, we explain the key ideas in more detail. Section III describes our simulation-based method to identify signal correlations. Section IV presents the method for implicit learning. Detail of our solver implementation is also discussed there. In Section V, we present the explicit learning method and discuss results on both satisfiable and unsatisfiable cases. Section VI concludes the paper.

II. Observations

Most SAT problems encountered in CAD applications are originated from circuits. In a traditional approach, a circuit is transformed into its CNF-equivalent form [5] and then a SAT solver is applied on the circuit in CNF. The major disadvantage with this transformation is the loss of the circuit structural information, specifically the topological ordering among all the signals. From the circuit point of view, a CNF formula is a 2-level OR-AND netlist with inverters possibly associated with the circuit inputs. Hence, with the CNF transformation, all internal signals in the original circuit become primary inputs in the corresponding 2-level OR-AND CNF netlist. When solving a circuit-originated problem, the topological information can be very useful, and with a CNF netlist, it is not easy to fully take advantage of that information.

II-A. The Incremental Learn-From-Conflict Idea

Consider the circuit in Figure 1. Suppose we want to solve a circuit SAT problem with the output objective "$c = 1$." If we apply SAT to

Proceedings of the Design,Automation and Test in Europe Conference and Exhibition (DATE'03)
1530-1591/03 $17.00 © 2003 IEEE

prove $c = 1$, then potentially, the search space is the entire circuit. Now suppose we can identify, in advance, two internal signals a and b such that "$a = 1$" and "$b = 0$" *individually* are very *unlikely* to happen. Then, we can divide the original problem into three sub-problems: 1) solving "$a = 1$," 2) solving "$b = 0$," and then 3) solving "$c = 1$." What is the advantage of doing so?

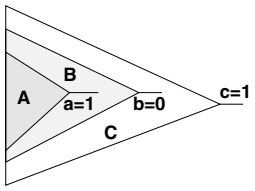

Figure 1: A SAT Process Following Topological Order

Since "$a = 1$" is unlikely to happen, conflicts are more likely to occur during the solving process. As a result, information can be learned more effectively. In a SAT solver, this information is stored as the *learned clauses* each representing a functional sub-space that contains no solution. From another point of view, each learned clause specifies a constraint on a set of circuit signals, which has to be true based upon the circuit structure.

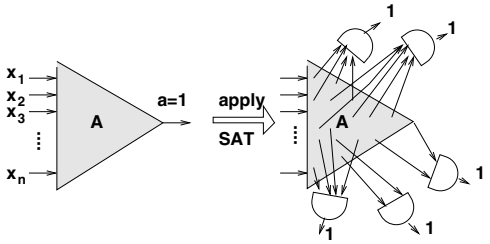

Figure 2: Learned Gates Accumulated by Solving "$a = 1$"

Most importantly, if we assume that solving "$a = 1$" is done only based upon the *cone of logic* headed by the signal "a" (as the shaded area A) then *the learned clauses will be based upon the signals contained in the area A only*. Figure 2 illustrates the results of applying SAT for solving "$a = 1$." Regardless of whether the problem is satisfiable or not, a set of learned clauses can be collected. In the figure, they are represented as the *learned AND gates* whose outputs are 1.

As the solver finishes solving "$a = 1$" and starts solving "$b = 0$," all the learned information regarding the circuit area A can be used to help solving "$b = 0$." In addition, if "$a = 1$" is indeed unsatisfiable, then signal "a" can be assigned with 0 when solving "$b = 0$." Similarly, learned information from solving "$a = 1$" and "$b = 0$" can be used to help solving "$c = 1$."

Intuitively, solving the three sub-problems would be much faster than solving the original problem. This is because when solving "$b = 0$," hopefully much fewer (or no) decisions are required to go into the area A. Hence, the search space is more restricted within the area B. Similarly, solving "$c = 1$" requires focusing decision making only on area C. Moreover, the learned clauses accumulated by solving "$a = 1$" would be shorter because they are based upon the signals on area A only. Similar, the learned clauses accumulated by solving "$b = 0$" would be shorter because fewer decisions are made on area A. Therefore, conceptually, this strategy allows us to solve a complex problem *incrementally*.

We make two key observations: 1) the incremental process suggests that we should guide the SAT solver to solve a sequence of preselected sub-problems following their topological ordering, and 2) the selection of the sub-problems such as "$a = 1$" and "$b = 0$" should be those most likely to be unsatisfiable. Intuitively, the search space for a likely unsatisfiable problem instance is more constrained and hence, conflict analysis can be more effective and more information can be accumulated in the learned clauses.

If solving "$a = 1$" and "$b = 0$" does not cause many conflicts, then there is not much information to be learned from the solving processes and hence, when solving "$c = 1$," the problem size cannot be reduced much (many decisions will still go into areas A and B). We note that in this case, although the the incremental strategy may not be effective, it should not be too harmful either because solving "$a = 1$" and "$b = 0$" are supposed to be fast (without many conflicts).

II-B. Potential Issues

We note that intuitively, the above incremental strategy would not be effective for solving circuit satisfiability if its CNF form is used. This is because with a 2-level OR-AND CNF structure, the topological ordering among the signals is lost. With a 2-level structure, the incremental strategy has very little room to proceed. Moreover, consider the above example again. Since all internal signals become primary inputs to the 2-level OR-AND CNF circuit, both a and b are primary inputs. Hence, the order of solving the sub-problems may become solving "$b = 0$" followed by solving "$a = 1$." In a later section, we will demonstrate that by reversing the topological ordering, the efficiency of the incremental strategy can be dramatically reduced.

Another key issue is how to identify in advance the sub-problems that are most likely to be unsatisfiable. In this paper, we propose to utilize the *pair-wise signal correlations* for identifying likely unsatisfiable cases. In the following, we describe our definition of the signal correlation.

III. Define and Identify Signal Correlations

In this section, we define the the concept of signal correlations. Following that, we will describe two approaches to utilize the signal correlations in the solver: *implicit learning* and *explicit learning*. In implicit learning, signal correlations are used to affect decision ordering and value assignments. In explicit learning, signal correlations are used to implement the incremental learn-from-conflict strategy.

Let $\{s_0, s_1, \ldots, s_n\}$ be "$s_0 = 0$" (0 is a constant) plus n signals on a given circuit. We say that s_i is correlated with s_j for all $i \neq j$, if either "$s_i = s_j$" or "$s_i \neq s_j$" is true with a very high probability when an arbitrary input assignment is supplied to the circuit.

Suppose we have identified a signal correlation between s_i and s_j as "$s_i = s_j$." Then, intuitively assigning $s_i = 1$ and $s_j = 0$ (or vice verse) will most likely casue the SAT process to produce many conflicts because the search space is highly constrained. As a result, much information can be learned and stored in the learned clauses.

Algorithm III.1: RANDOM SIMULATION(*Circuit*)

```
comment: C is the initial equivalence class.
i ← 0; C ← set of all signals;
S = {C}; S' ← S;
while (i < 4) and (S ≠ S')
do { S' ← S;
     Produce 32 random input assignments;
     Perform parallel logic simulation [10];
     Based upon the simulation results and
      the current equivalence classes,
       compute the new equivalence classes;
     S ← {the new equivalence classes};
     i ← i + 1;
return (S)
```

How to identify signal correlations? One possible way is to utilize *random simulation*. In Algorithm III.1, we demonstrate a simple procedure to compute the set of equivalence correlations (i.e.

"$s_i = s_j$") using random simulation. To identify the correlations "$s_i = 0$," "$s_i = 1$," and "$s_i \neq s_j$" are not the same but similar.

In the algorithm, an *equivalence class* is a subset of signals mutually having the equivalence relationship. In the algorithm, two signals will be put into the same class if their simulation results from all the simulated assignments are identical.

We note that deciding the equivalence classes among all signals can be achieved quite efficiently with a hash table. Hence, its run time is actually close to linear instead of quadratic on the number of signals under consideration. Also note that each time, only the signals currently belonging to the same equivalence class are needed to be considered together. If two signals have already been sorted into different classes, then no need to further decide if they have the equivalence relationship or not.

Our random simulation is simple. Each time 32 random input assignments are simulated together using a word (32 bits) in parallel logic simulation [10]. Based upon the simulation results, two signals are decided whether they still belong to the same equivalence class or not. If repeating the simulation step four times does not lead to identifying any new equivalence class(es), then the simulation stops. At the end, the set of the equivalence classes is returned. In our current implementation, we use "four" as the number to stop the simulation. This number is selected arbitrarily. What we need is a small constant so that as soon as the efficiency of the random simulation for identifying new signal correlations drops, the simulation stops. In this way, we can avoid spending too much time in simulation.

In the returned set S, an equivalence class containing the subset of signals $\{s_1, \ldots, s_m\}$ indicates that they are mutually correlated ($s_i = s_j, \forall i, j$). we note that if constant 0 is not included and the size of the subset is greater than 3 (i.e. $m > 3$), then we would remove those signals from being considered as correlated. The reasoning behind this step is that a large m might be just an indication that the random simulation is ineffective to differentiate those signals. It does not necessarily imply that those signals are actually having the equivalence relationships with a high probability.

We emphasize that our definition of the signal correlation includes for a particular signal s_i that s_i is correlated to the constant 0. Hence, both "$s_i = 0$" and "$s_i = 1$" cases are included in our definition of the signal correlation. For convenience, we sometimes still call them "pair-wise" correlations with the understanding that the pairs are defined over a signal and the constant 0.

IV. Implicit Learning By Signal Grouping

The incremental learn-from-conflict strategy involves *explicitly* selecting a correlated pair of signals, assigning their values in the way that it will most likely to cause conflicts, and then utilize the SAT solver to accumulate learned clauses (learned gates). In contrast, in implicit learning, signal correlations are used to influence the decision variable selection and variable value assignment. This means that instead of creating a sequence of likely unsatisfiable sub-problems for the SAT solver to solve them explicitly one by one, correlation information is used only *within the decision variable selection procedure*.

The latest Chaff package ZChaff provided the baseline for our development. In the initial version of the circuit-based SAT solver, we implemented all the ideas in ZChaff, including the VSIDS decision variable selection, clause removal, watched literal [1], and UIP based conflict analysis [2]. In addition, our solver can also add the justification frontier (J-node) [10] into the consideration of decision variable selection.

In our correlation-guided implicitly learning, we use signal correlations to "group" variables in the decision variable selection. For example, suppose a signal s_i is correlated with signal s_j as "$s_i \neq s_j$." During the solving process, whenever s_i gets to be assigned a value, we want to "immediately" make the decision to assign the same value to s_j. In this way, we "group" the two signals together in the solver's value-assignment process, and values are assigned in such a way that they are most likely to cause conflicts.

If two signals are highly correlated in terms of "$s_i \neq s_j$," then intuitively, the search sub-space imposed by "$s_i = s_j$" will be highly constrained. As a result, SAT can learn more quickly in this sub-space than searching the complementary sub-space imposed by "$s_i \neq s_j$." Algorithm IV.1 depicts the detail of our signal correlation guided implicit learning.

Algorithm IV.1: SELECT DECISION VARIABLE(*Jnodes*)

Suppose s just being assigned a value v by implication (BCP)
if ($\exists s'$, s' is correlated with s) and (s' has not yet assigned a value)
then
- select s' as the next decision signal
- **if** (it is equivalence correlation)
  - **then** $s' \leftarrow \overline{v}$
  - **else** $s' \leftarrow v$

else
- use VSIDS among J-nodes to select a signal s'
- **if** (s' is correlated with 0)
  - **then**
    - $s' \leftarrow 1$ if it is equivalence correlation, or
    - $s' \leftarrow 0$ otherwise

return (s')

IV-A. Our Circuit Solver Implementation Detail

We first describe the implementation detail of our circuit-based solver, and present the baseline comparison results between ZChaff and our solver without the proposed learning. Improved results by implicit learning then follow.

Although our initial circuit-based SAT solver borrows almost all the ideas from ZChaff, our implementation is different from ZChaff in several aspects:

- The input to the solver is assumed to be in a circuit format (such as the ".bench" format). After the circuit is read in, it is transformed into a netlist based upon only the 2-input AND primitive. In the netlist, we allow inverters to be associated with the AND gate inputs as attributes. Lookup tables are used for fast implications on the AND primitive [8].

  If an input is in its CNF form, we first convert it into a 2-level OR-AND circuit. Then, the circuit will be given to our circuit solver. We note that this could add some overhead to the representation of the problem.

- Since our ultimate goal is to develop a circuit SAT solver applicable to sequential circuits directly, internal circuit representation and data structures were designed for later extension to the sequential domain. For instance, "FRAME" objects were used to contain dynamic information that is valid within a time frame during sequential time frame expansion [10]. This adds additional overhead to our code, as comparing to ZChaff.

- For each learned clause, pointers to the two watched literals are explicitly stored. In this way, when one literal gets assigned a value, it can immediately check to see if the other has been assigned a value or not, as opposed to an additional search on the learned clause in the current ZChaff implementation. This change is minor.

- We implement *restart* based upon the average back-jump levels calculated over 4096 backtrack occurrences. In our current implementation, if the average is less than 1.2, we re-start the solving process.

- In ATPG terminology, a justification frontier (J-node) is a gate whose output has received a value, and some of its inputs need further decision(s) to justify the value. For example, if an AND gate output is set to 0, then it becomes a J-node if none of its inputs are currently assigned with 0 and there are more than one unassigned inputs.

  In our implementation, only inputs to J-nodes are considered in the calculation of VSIDS for current decision making. However, we note that here our definition of the J-node include all the learned gates. Therefore, initially the restriction on J-nodes for decision making would make our solver behave differently from the ZChaff. However, after many learned gates are accumulated, effectively the two would follow the same VSIDS algorithm. Nevertheless, we note that since our solver tends to make different decision at the beginning, the entire decision ordering can be very different from ZChaff.

  One important thing worth mentioning is that if we did not treat the learned gates as J-nodes, then the performance would degrade significantly.

IV-B. Baseline Comparison Results

Tables I presents the comparison results for unsatisfiable cases. "C-SAT" is our implementation of the SAT solver using the original VISDS decision variable selection. "C-SAT-Jnode" is the version including the J-node decision selection. All experiments ran on a Pentium-3 1G machine with 1G RAM under Linux Mandrake 2.4.3.

In the cases denoted as "circuit.equiv" we contructed an equivalence checking circuit model by taking two copies of the same circuit. Each pair of corresponding primary outputs are XORed and all the outputs of the XOR go to an AND gate. The SAT problem is to ask if the output of the AND gate is 1. In each case, it is unsatisfiable. As shown, C-SAT-Jnode is slightly better than C-SAT, and is slightly worse than ZChaff. In the case of C6288, none of the solvers was able to complete the run.

Circuit	ZChaff	C-SAT	C-SAT-Jnode
C1355.equiv	3.7	7.98	3.77
C1908.equiv	4.6	5.22	1.39
C3540.equiv	53	101	102
C5315.equiv	56	49	105
C7552.equiv	215	304	177
C6288.equiv	*	*	*
Total	332.3	467.2	389.16

*Aborted after 7200 seconds.

TABLE I: INITIAL RUN TIME RESULTS (SECS) FOR UNSAT CASES

Circuit	ZChaff	C-SAT	C-SAT-Jnode
9Vliw001	1057	3539	969
9Vliw004	953	1521	133
9Vliw005	3126	1114	462
9Vliw007	140	98	3938
9Vliw008	1450	1722	3184
9Vliw010	867	1291	25
Total	7593	9285	8711

TABLE II: INITIAL RUN TIME RESULTS (SECS) FOR SAT CASES

Similar results can be observed in Table II for satisfiable cases taken from the benchmarks in [9]. The slight performance degradation in our tool can be attributed to the more complicated data structures reserved for future sequential circuit solver enhancement.

IV-C. Improved Results Using Implicit Learning

Table III shows the results for unsatisfiable cases by applying the implicit learning method to our circuit solver. Several new examples are added, denoted as "circuit.opt." These examples are similar to the equivalence checking models described earlier. The difference is that in these examples, we did not take two identical copies of the same circuits. Instead, we optimize a circuit with Design Compiler to produce a functionally equivalent, structurally different circuit. Then, the two circuits are used to create the equivalence checking model.

For the "circuit.equiv" cases, our new solver achieves more than 5x speedup over ZChaff by looking at the sub-total results. For the "circuit.opt" examples, the speedup is more than 10x. If we consider all cases together, the speedup is more than 7x. This demonstrates the effectiveness of our implicit learning techniques for solving the unsatisfiable cases originated from circuits. For all cases, the simulation times are minimal. We emphasize that the new C-SAT-Jnode solver was still unable to complete the run on C6288.

		With Implicit Learning	
Circuit	ZChaff	C-SAT-Jnode	Simulation
C1355.equiv	3.7	0.35	0.01
C1908.equiv	4.6	0.6	0.02
C3540.equiv	53	21	0.03
C5315.equiv	56	17	0.04
C7552.equiv	215	23	0.07
C6288.equiv	*	*	0.05
Sub-Total	**332.3**	**61.95**	0.22
C3540.opt	41	39	0.03
C5315.opt	127	4.08	0.04
C7552.opt	433	14.5	0.08
Sub-Total	**601**	**57.58**	0.15
Total	**933.3**	**119.53**†	0.37

*Aborted after 7200 seconds.

†About 7.8x speedup without counting simulation.

TABLE III: IMPROVED RESULTS FOR UNSAT CASES WITH IMPLICIT LEARNING

Table IV presents the results on the satisfiable examples. The speedup is not as dramatic as the unsatisfiable cases. However, if we look at the total run time, more than 2x speedup is obtained. However, the simulation times are noticeable in these examples. The only case that the new solver is slower is the first example. But we note that this is also the easiest example among all.

One thing worth mentioning is that for these satisfiable benchmarks, each problem seems to be specified in such a way that part of the problem is described as a multi-level circuit, and part of it is described in CNF form (instead of constraint gates on the internal signals). Since our reasoning behind the effectiveness of using signal correlations is based entirely on the circuit structure, it is unclear how much the same reasoning would hold based upon those satisfiable benchmarks. Hence, performance degradation was expected.

		With Implicit Learning	
Circuit	ZChaff	C-SAT-Jnode	Simulation
9Vliw007	140	286	110
9Vliw010	867	329	96
9Vliw004	953	804	92
9Vliw001	1057	567	93
9Vliw008	1450	239	114
9Vliw005	3126	740	88
Total	**7593**	**2965**†	593

†About 2.5x speedup without counting the simulation time.

TABLE IV: IMPROVED RESULTS FOR SAT CASES WITH IMPLICIT LEARNING

V. Signal Correlation Guided Explicit Learning

In explicit learning, a sequence of likely unsatisfiable sub-problems are created based upon signal correlations computed by the random simulation. Internally, the C-SAT-Jnode solver would try to solve each sub-problem one by one. After finishing solving all the sub-problems (this point will be discussed more in Section V-C later), all

the learned information is stored in the learned gates which are then used to solve the original SAT problem. Four aspects are worth mentioning in our implementation.

- When solving each sub-problem, the solver may not complete the solving. In our current implementation, the solver stops after accumulating 10 learned gates. We note that the primary goal of solving each sub-problem is to learn more circuit information. Hence, it is not necessary to solve a sub-problem completely. Not completely solving each sub-problem differentiates our approach from a typical *equivalence check point matching* approach commonly adopted in structural equivalence checking. We found that for pure circuit benchmark examples, aborting the solving of sub-problems earlier does not make much of the difference as compared to solving them completely. However, for those examples in the format of CNF form, a strategy of solving each sub-problem completely may significantly degrade the overall run time efficiency. This is understandable because the the effectiveness of the explicit incremental strategy highly depends on the circuit structure. If such a multi-level circuit structure does not exist, the overhead of solving the many sub-problems may easily out-weight the potential efficiency gain.
- When solving each sub-problem, the solver follows the topological ordering. Each sub-problem can be based upon two internal signals or upon one signal and the constant 0.
- At this point, our C-SAT-Jnode is the version including the implicit learning as well.
- The usage of J-node decision is crucial in this case. By following the J nodes, the search process for solving each sub-problem can be (more) restricted within the two *cones of logic* headed by the two correlated signals (or the cone of logic headed by the single signal correlated to 0). With this feature, solving each sub-problem can be done on the original circuit, instead of explicitly creating another sub-circuit for each sub-problem. This allows us to implement the explicit incremental strategy more efficiently.

		Individual					
		Signal Pair		Signal Vs. 0			
circuit	ZChaff	Time	Num.	Time	Num.	Both	Simu.
C1355.equiv	3.7	0.2	257	0.63	141	0.13	0.01
C1908.equiv	4.6	0.24	163	0.67	113	0.21	0.02
C3540.equiv	53	3.08	510	26	86	1.98	0.03
C5315.equiv	56	1.36	736	12	319	0.42	0.04
C7552.equiv	215	4.78	1064	18	342	3.81	0.07
Sub-Total	**332.3**	**9.66**	—	**57.3**	—	**6.21**	0.17
C3540.opt	41	1.33	531	24.5	86	1.01	0.03
C5315.opt	127	1.76	829	23.6	325	0.78	0.04
C7552.opt	433	5.18	1112	27.37	320	4.02	0.08
Sub-Total	**601**	**8.27**		**75.47**		**5.81**	0.15
Total	**933.3**	**17.93**		**132.77**		**12.02**†	0.32
C6288.equiv	*	**22.4**	1968	*	67	**9.6**	0.05

*Aborted after 7200 seconds.

*About 77x speedup over ZChaff without counting simulation.

TABLE V: IMPROVED RESULTS FOR UNSAT CASES WITH EMPLICIT LEARNING

Table V summarizes the results for the unsatisfiable examples. Three new columns of results are shown: "Signal Pair," "Signal Vs. 0," and "Both." In the first column, explicit learning was done based upon correlations on pairs of signals. In these experiments, signal correlations with the constant 0 were ignored, and were considered in the second column. Similarly, in the experiments of the second column, correlations on pairs of signals were ignored. Results on these two column separately demonstrate the effect from each type of the correlation learning. In each case, the numbers of sub-problems created are also included (as "Num."). In the "Both" column, both types of the signal correlations were used. Several things can be observed:

(1). Only considering correlations with 0 is less effective than only considering pairs of signals. **(2).** Considering both types of correlations is better than only considering each type individually. **(3).** By comparing the results in the "Both" column to ZChaff column, for "circuit.equiv" examples, the "Sub-total" results show more than 50x speedup. For "circuit.opt" examples, the speedup is more than 100x. Averaging on the two, the speedup is about 75x. This demonstrates that our *incremental learn-from-conflict* strategy was able to effectively learn useful information by taking advantage of the circuit structure. **(4).** The most noticeable results are for C6288. The new solver now is able to finish the run in only 9.6 seconds.

V-A. The Ordering Of Explicit Learning

In our incremental learn-from-conflict implementation, the ordering of the explicit learning follows the topological order of the signals. The reason was explained in Section II-A before. What if we disturb that ordering? In this section, we consider two more experiments: one by reverse the ordering of explicit learning, and the other by randomly selecting sub-problems for solving. Results are shown in Table VI.

	The Ordering of Explicit Learning		
Circuit	Topological	Reverse	Random
C1355.equiv	0.13	1.56	0.55
C1908.equiv	0.21	1.17	0.91
C3540.equiv	1.98	52	20
C5315.equiv	0.42	8.69	4.8
C7552.equiv	3.81	21	16
Sub-total	**6.21**	**84.42**	**42.26**
C6288.equiv	9.6	*	*

*Aborted after 7200 seconds.

TABLE VI: EFFECTS FROM THE ORDERING OF EXPLICIT LEARNING

As expected, if the topological ordering is disturbed, the effectiveness of the incremental learn-from-conflict strategy may degrade. As we can observe, a random ordering is better than the reverse ordering, and both are inferior to the topological ordering. One noticeable result is that if we do not follow the topological ordering, then the solver would not be able to complete the run for C6288.

Results in Table VI demonstrate the importance of following the topological ordering in the incremental learning process. Therefore, if a problem input is in the CNF format, then applying the incremental strategy does not guarantee that the explicit learning process indeed is following the topological order in the original circuit structure which the problem is derived from. For this reason, we suspect that for those satisfiable benchmark examples considered earlier, where they partially are specified in the CNF form, the proposed explicit learning may lead to inferior results to the implicit learning approach. Without following the topological ordering, the effectiveness of the incremental learning is degraded, and the overhead associated with the incremental process can out-weight the potential efficiency gain from the explicit learning.

V-B. Performance Degradation to The Satisfiable Examples Containing CNF Formatted Inputs

Table VII shows the anticipated degraded results for the satisfiable cases. However, we note that with the incremental strategy, although the 2x speedup shown in Table IV before is no longer there, the performance of the C-SAT-Jnode solver remains comparable to ZChaff. It is interesting to note that among these examples, an easier problem for ZChaff would be a bit harder for our circuit solver. The reverse is

Circuit	ZChaff	C-SAT-Jnode (Both)	Simulation
9Vliw007	140	855	110
9Vliw010	867	1897	96
9Vliw004	953	1011	92
Sub-Total	1960	3763	298
9Vliw001	1057	793	93
9Vliw008	1450	1914	114
9Vliw005	3126	1314	88
Sub-Total	5633	4021	295
Total	**7593**	**7784**	593

TABLE VII: RUN TIME DEGRADATION FOR SAT CASES IN EXPLICIT LEARNING

also true: A harder problem for ZChaff would be a bit easier for our solver to run.

V-C. On The Amount of Explicit Learning

In this section, we conduct controlled experiments: Instead of solving all the sub-problems, our solver would try to solve the sub-problems of which their topological locations are before a certain boundary. For example, given a percentage 50%, by following the topological ordering, the solver only considers the correlations involving the first half of the signals. The correlations involving the second half of the signals are not used in the explicit learning. In this way, we can study the impact on the efficiency of the solver based upon the amount of explicit learning.

	Conduct only a % of Explicit Learning							
Circuit	0.1	0.2	0.4	0.5	0.7	0.9	0.95	1
C3540.equiv	49	27.5	38.7	31.2	4.5	7	2.2	1.98
C5315.equiv	16.5	16.6	3.4	2.7	3	2.4	0.5	0.42
C7552.equiv	21.3	12.1	20.7	18.9	8.7	5.9	4.9	3.81
Sub-total	86.8	56.2	62.8	52.8	16.2	15.3	7.6	6.21
C6288.equiv	*	*	*	*	*	91.8	15.8	9.6

*Aborted after 7200 seconds.

TABLE VIII: THE EFFECT OF PARTIAL LEARNING ON UNSAT CASES

Tables VIII and IX present the results on the controlled experiments. The percentage number in each case is denoted and "1" indicates 100%. In the case of 100%, the results are the same as the "Both" columns shown in Table V and Table VII earlier.

We see that for "circuit.equiv" cases, there is a clear trend that the less amount of explicit learning is involved, the less effective the solver would be. For the satisfiable cases in Table IX, the trend is totally reversed although it is not so smooth. We note that there is a significant difference between 50% and 100% explicit learning.

Also notice that as the percentage of explicit learning move below 90%, the solver would not finish solving C6288. This says that C6288 is an extreme case where solving the problem instance requires take full advantage of the incremental strategy.

	Conduct only a % of Explicit Learning				
Circuit	0.5	0.7	0.8	0.95	1
9Vliw007	286	671	324	1846	855
9Vliw004	804	516	794	645	1011
9Vliw010	329	1816	192	1374	1897
9Vliw008	239	1976	863	602	1914
Total	1658	4979	2173	4467	5677

TABLE IX: THE EFFECT OF PARTIAL LEARNING ON SAT CASES

VI. Conclusion and Future Work

Table VI shows additional results. We note that "sxxxxx.scan" examples are sequential circuits where all state holding elements are treated as primary inputs. We conjecture that because of this change in the circuit structure (circuit depth becomes more shallow), the effectiveness of our learning techniques degrades as compared to the combinational circuit examples "Cxxxx" shown before.

Circuit	ZChaff	Implicit	Explicit	Simulation
9Vliw009	1006	784	829	117
9Vliw017	1007	175	913	109
9Vliw001	1057	567	793	93
9Vliw024	1375	965	1282	107
9Vliw021	1666	1069	1345	99
9Vliw015	2209	985	1270	97
9Vliw019	2936	849	1448	129
Sub-Total	11256	5394 (2.08x)	7780 (1.44x)	751
c2670.equiv	1.89	0.89	0.35	0.01
c1908.opt	6.5	0.64	0.18	0.01
s13207.scan.equiv	16	12	7.5	0.29
s15850.scan.equiv	44	13	2.61	0.56
s35932.scan.equiv	322	170	46	0.2
s38417.scan.equiv	547	157	10	1.83
s38584.scan.equiv	882	236	66	3.58
Sub-Total	1819.39	589.53(3x)	132.64(13.7x)	6.48

TABLE X: RESULTS FOR ADDITIONAL SAT AND UNSAT CASES

In conclusion, this paper describes our implementation of a circuit-based SAT solver whose design philosophy is to take advantage of the signal correlations and circuit topological structure. We propose a new SAT solver design concept called *incremental learn-from-conflict* where the solver learning process is carefully guided by signal correlation information. We differentiate between the *implicit* learning and the *explicit* learning approaches both implemented in our solver. We discuss their strengths and compare their performance. Although we do not consider that our circuit-based solver is superior to ZChaff in general on solving CNF-based problem instances, we do conclude that if our solver is able to take advantage of the circuit structural information, significant performance improvements can be obtained. For the future work, we will continue the development of our solver for handling sequential circuits directly.

References

[1] M.Moskewicz, C.Madigan, Y.Zhao, L.Zhang, and S.Malik, Chaff: Engineering an efficient SAT solver. *Proc. ACM/IEEE Design Automation Conference 2001*.

[2] L.Zhang, C.Madigan, M.Moskewicz, and S.Malik. Efficient conflict driven learning in a Boolean satisfiability solver. *ICCAD 2001*.

[3] H. Zhang. SATO: An Efficient Propositional Prover. *Proc. of International Conference on Automated Deduction*, Vol 1249, LNAI, 1997, pp. 272-275

[4] J.P.Marques-Silva and K.A.Sakallah. GRASP: A Search Algorithm for Propositional Satisfiability. *IEEE Transactions on Computers, vol.48, pp. 506-521 1999.*

[5] T. Larrabee. Test Pattern Generation Using Boolean Satisfiability. In *IEEE Transactions on Computer-Aided Design*, pages 4-15, Jan, 1992.

[6] A.Kuehlmann, M.Ganai, and V.Paruthi. Circuit-based Boolean Reasoning. *DAC 2001*.

[7] Slawomir Pilarski and Gracia Hu. SAT with Partial Clauses and Back-Leaps. In *Proc. ACM/IEEE Design Automation Conference 2002*

[8] M.K.Ganai, L.Zhang, P.Ashar, A.Gupta, and S.Malik. Combining strengths of circuit-based and CNF-based algorithms for a high-performance SAT solver. In *Proc. ACM/IEEE Design Automation Conference 2002*

[9] M.N. Velev. http://www.ece.cmu.edu/ mvelev Benchmark Suites, October 2000.

[10] Miron Abramovici, Melvin A. Breuer, and Arthur D. Friedman, Chapters 3,5, and 6: Logic Simulation, Fault Simulation, Test Generation, *Digital Systems Testing and Testable Design*, W.H.Freeman, 1990.