

Utilizing Don't Care States in SAT-based Bounded Sequential Problems

Sean Safarpour
Dept. Elec. & Comp. Eng.
University of Toronto
Toronto, ON
ssafarpo@eecg.toronto.edu

Andreas Veneris
Dept. Elec. & Comp. Eng.
University of Toronto
Toronto, ON
veneris@eecg.toronto.edu

Goerschwin Fey
Dept. Comp. Sci.
Bremen University
Bremen, Germany
fey@informatik.uni-bremen.de

Rolf Drechsler
Dept. Comp. Sci.
Bremen University
Bremen, Germany
drechsle@informatik.uni-bremen.de

ABSTRACT

Boolean Satisfiability (SAT) solvers are popular engines used throughout the verification world. Bounded sequential problems such as bounded model checking and bounded sequential equivalence checking rely on fast and robust SAT solvers. In this work, we introduce a technique that improves the performance of the underlying SAT solver for bounded sequential problems by taking advantage of a design's don't care states. We develop cost effective methods of filtering, replicating and applying the don't care states to the original problem thus reducing the search space. Experiments demonstrate the effectiveness of the proposed method on ISCAS'89 benchmarks.

Categories and Subject Descriptors

J.6 [Computer-Aided Engineering]: Computer-aided design (CAD)

General Terms

Algorithms, Performance, Verification

Keywords

Don't Care States, Satisfiability, Bounded Model Checking, Sequential Equivalence Checking, Unreachable states

1. INTRODUCTION

Boolean Satisfiability (SAT) solvers are common engines in verification applications today [1] [9] [18]. Much improvement has been made in the area of both general and problem specific SAT solvers [10] [12] [16]. One popular im-

provement approach is exploiting problem specific information such as signal correlations [10], structural implications [17], BDD-based learning [2] [6], and circuit don't cares [14]. For sequential applications, the state space of the circuit can provide valuable information to the SAT solver [2] [7]. In this paper, we seek to improve the SAT solver performance specifically for bounded sequential applications such as Bounded Model Checking (BMC) and Bounded Sequential Equivalence Checking (BSEC).

In BMC and BSEC problems, the sequential circuit is verified for correctness only within a finite number of clock cycles [1] [18], also known as a *bound*. The main advantage of these techniques is that they can often find counterexamples or errors in the design when other, more complete techniques fail due to resource constraints. The rationale behind the recent success of these techniques arises from the fact that, in most cases, under normal operation a circuit is exercised only for a finite number of clock cycles. Overall, BMC and BSEC are popular techniques employed during design verification [1] [18].

In BMC and BSEC, sequential don't care conditions, or simply Don't Care States (DCS), are especially important as they can provide valuable insight into the circuit's behavior over time [2] [4]. Obtaining don't care states can be done in several ways; while many DCS are inherently known to the designers, others can be found through reachability analysis [8] and through efficient approximation methods using Binary Decision Diagrams (BDD) [5] [11] [13].

In this work, we propose an approach that uses DCS to improve the SAT solver performance for BMC and BSEC problems. First, DCS are found and extracted using existing BDD techniques [5]. Second, using our developed heuristics, non beneficial DCS are identified and discarded. Given the nature of the problem, these heuristics eliminate DCS that overcrowd the original problem and those that do not provide quick implications. In the third step, the remaining DCS are replicated and applied to all time frames of the original problem. Finally, the DCS enriched problem is solved using a conventional SAT solver [12]. Since finding DCS through efficient and approximate techniques can be fast,

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

GLSVLSI'05, April 17–19, 2005, Chicago, Illinois, USA.
Copyright 2005 ACM 1-59593-057-4/05/0004 ...\$5.00.

our approach is viable for non-trivial SAT-based bounded sequential problems. Extensive experiments on BMC and BSEC problems demonstrate and confirm the effectiveness of DCS.

This work is presented as follows. The next section provides an overview of the previous work in this field. Section 3 contains background information and definitions used throughout the paper. Section 4 proceeds to describe the procedure of applying don't care states to BMC and BSEC problems, while section 5 describes the developed heuristics in detail. The experiments are presented in section 6 and section 7 concludes this paper.

2. PREVIOUS WORK

It is generally understood that BDD-based and SAT-based methods are complementary in their approaches to verification problems and much work has been done to combine these two techniques [1] [2] [6]. More specific to our topic, Cabodi et al [2] and Gupta et al [7] use BDDs to find approximate reachable and unreachable states to simplify the task of SAT solvers on BMC problems.

Cabodi et al [2] focus on BMC problems that seek to answer the question whether there exists a path from a source state S to a target state T of length k . Due to their problem statement, they are able to perform both approximate forward and reverse traversal of the problem from state S or T using BDDs. Their approach exploits the *reachable* states that exist for each time frame and applies them to the Conjunctive Normal Form (CNF) problem. They also explore different methods of “dumping” or converting the BDDs into CNF. Similarly, Gupta et al [7] use the *reachable* states in each time frame to reduce the search space. If illegal state information is provided, they propose doing BDD-based bounded pre-image computation to get sets of unreachable states specific to each time frame. However, no experimental evidence is provided in that paper to substantiate the claims for the case of unreachable states. The above approaches concentrate on *sequences* of *reachable* states for BMC. In other words, they provide redundant information at each step of the BMC problem to speed up the SAT solver. Furthermore, both methods simply apply all the BDD information to the original problem without performing any filtering. In contrast, our work focuses on a design's general don't care states or *unreachable states regardless of a particular time frame*. The advantage of this approach is that the DCS information applies to all time frames as opposed to a specific time frame. Furthermore, not all DCS are applied to the problem; we present two heuristics that replicate useful DCS while eliminating DCS that might not benefit the application. The proposed heuristics evaluate the usefulness of the DCS in terms of their ability to produce implications without overcrowding the original problem. Finally, our method applies to both problems with initial state constraints such as BMC and those without initial state constraints such as BSEC. We demonstrate that in BMC problems the DCS provide redundant information while in BSEC problems, they provide previously unavailable information. The details of our approach are outlined in section 4 and 5.

3. BACKGROUND

3.1 Notation

In this paper, we consider synchronous sequential circuits and refer to the memory elements as latch variables or state variables. We translate the combinational part of a circuit into its respective CNF as outlined in [9]. A *SAT solver* refers to a DPLL-based search algorithm based on the techniques of [3] and [16]. In this work, the same names are used for circuit lines and SAT variables alike. *Clause size* denotes the number of literals in a clause.

A *literal* is a Boolean variable or its complement. We refer to a *cube* as a conjunction of literals [4]. We say that a cube A *covers* another cube B , if the literals of A are a subset of the input literals in B [4]. In other words, for every *minterm* in B the same minterm exists in A .

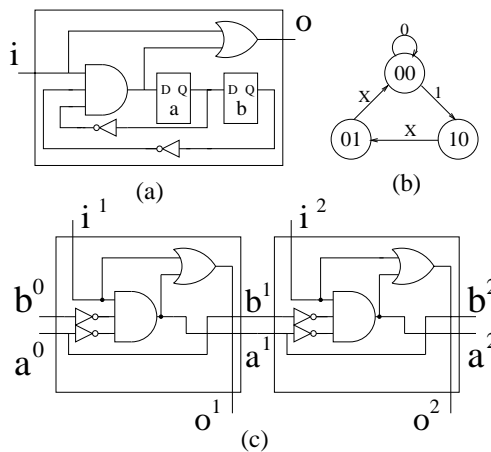


Figure 1: The ILA model for Sequential Designs

Typically, in bounded sequential problems, the behavior of the circuit is modeled using the *Iterative Logic Array* (ILA) representation (also known as the *time frame expansion* model) [8]. The ILA is created by *unfolding* or replicating the Finite State Machine (FSM) k times. In this representation, each instance of the FSM in the ILA is called a *time frame*. We use k throughout this paper to refer to the number of times a circuit's FSM is unfolded. For example, consider the circuit in Figure 1 (a) and its transition diagram depicted in (b). Here, the FSM is extracted and unfolded twice ($k = 2$) resulting in the circuit in (c).

Through this paper, a variable v^i refers to the corresponding variable v in the original circuit in time frame i . The time frames in an ILA are linked together by connecting the output of the latch in time frame i to the respective input of the latch in time frame $i + 1$. Note that latch input at time frame 1 represents the initial state condition while latch output at the last time frame can be ignored/removed if it is not a primary output.

3.2 Bounded Model Checking

In BMC, the problem is to verify that a given circuit correctly implements a set of properties for a finite number of clock cycles k starting from some legal initial state [1]. This problem can be formulated as a SAT problem by unfolding the circuit for k time frames, generating a property circuitry

that verifies the correctness at each time frame, constraining the initial state and converting the problem to CNF which is given as input to the solver. Figure 2 illustrates a BMC problem for $k = 3$ for the circuit in Figure 1. A SAT solver can solve the problem returning the result **SATISFIABLE** if a counter-example to the property is found; otherwise, it cannot find such an assignment and returns **UNSATISFIABLE**.

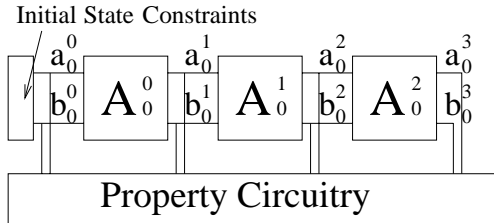


Figure 2: BMC problem structure

3.3 Bounded Sequential Equivalence Checking

In BSEC, the problem is to verify that two designs implement the same function within a bound of a certain number of clock cycles k [18]. This method of sequential equivalence is especially effective when verifying retimed circuits or pipelined circuits. The problem can be reformulated by unfolding the two designs for k time frames, tying all primary inputs together, and generating a miter circuit to check the equivalence of the primary outputs. Figure 3 illustrates a BSEC problem for $k = 3$ time frames for the circuit in Figure 1. An important difference between the BMC and BSEC problems is that there are no initial state constraints for BSEC.

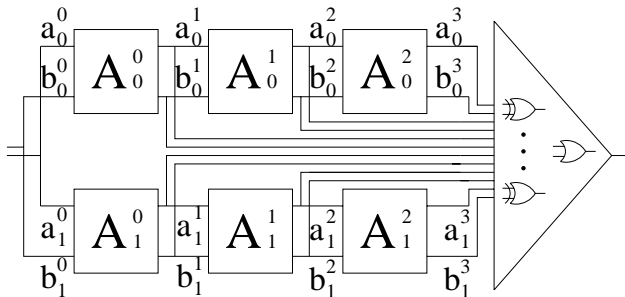


Figure 3: Miter construction for SEC problem

3.4 Don't Care States

Let the set of states that are *reachable* in a FSM be denoted as S_r [8]. Intuitively, these are all the states that are visited by traversing the FSM starting from a set of initial states S_0 . The *unreachable states*, S_u , are all of the states not in S_r [8]. *Reachability analysis* can yield the reachable states of a design. Reachability analysis is a computationally intensive task for large circuits and many approximation methods exist which can efficiently provide subsets of S_r or S_u [5] [11] [13]. In this paper, the terms unreachable states and don't care states are used interchangeably. Note that

finding the DCS is not the topic of this work but we focus on filtering and applying beneficial sets of DCS to improve the performance of SAT in BMC and BSEC.

4. UTILIZING DON'T CARE STATES

In sequential circuits the set of unreachable states can often take a large fraction of the total state space [13]. Therefore, pruning the search space associated with the don't care states is expected to reduce the overall search space during BMC and BSEC. More specific to SAT solvers, adding clauses to encode the DCS will generate more implications and decrease the number of backtracks. Next, we discuss the details of encoding don't care states as CNF clauses.

We refer to a clause encoding don't care states as a *DCS clause*. To prune the search space associated with DCS, sets of don't care states (given in terms of functions or cubes) are converted into CNF. Each variable appearing in the don't care states function is mapped to a literal taking the complemented phase of the variable value. For example, to prevent the combination $\{a = 1, b = 1\}$ from occurring, the clause $(\bar{a} + \bar{b})$ is generated. In this case, as soon as variable a (b) is assigned a 1, either an implication results which assigns $b = 0$ ($a = 0$), or a conflict is detected.

Given the repetitive nature of bounded sequential problems, each don't care states cube can be reused for all time frames. For example, if $\{a = 1, b = 0, c = 1\}$ is a don't care states cube, then the following don't care states clauses can be applied to the problem to prevent this combination from occurring in all k time frames:

$$(\bar{a}^0 + b^0 + \bar{c}^0) \cdot (\bar{a}^1 + b^1 + \bar{c}^1) \cdot \dots \cdot (\bar{a}^{k-1} + b^{k-1} + \bar{c}^{k-1}) \cdot (\bar{a}^k + b^k + \bar{c}^k)$$

As discussed earlier, BMC and BSEC problems are quite similar in structure. However, the fact that BMC problems often constrain their initial state while BSEC problems do not, is very important for the application of DCS. In BMC, since the initial state is applied to the problem, latch variables in all time frames are *implicitly* constrained to the reachable states via the circuit structure. In other words, when a SAT solver returns a **SATISFIABLE** result for BMC problems, all latch variables inherently get assigned a reachable state value. As a result, the don't care state clauses act as redundant clauses for BMC similar to conflict clauses. Intuitively, don't care states provide the SAT solver "short cuts" to find a solution quickly.

In BSEC, where the initial state variables are not constrained, latch variables can take on any value, even those corresponding to illegal or unreachable states. As a result, in BSEC, the DCS information is not redundant and provides previously unavailable insight into the problem. The effect of the don't care states for BSEC problems is the pruning of large sections of the search space. In the next section we discuss two heuristics which are specifically tuned to improve the performance of BMC and BSEC problems.

5. PERFORMANCE HEURISTICS

Enriching the CNF with the don't care states clauses reduces the problem search space. However, it is well known that adding extra clauses (redundant or not) to problems does not necessarily improve the SAT solver performance [6] [12]. Here, we discuss a set of heuristics we use to minimize

the overhead generated by the DCS clauses while maximizing their ability to generate implications.

Typically, adding extra clauses to the SAT solver degrades the performance due to at least one of the following reasons.

1. *The added clause information is available directly in another clause already present in the CNF.* In this case, the performance is reduced because the new clause does not provide any new information while adding maintenance overhead. For example, adding clause $(a + b + c)$ if clause $(a + b)$ is already in the CNF does not provide any benefits because the corresponding cube is covered by that of $(a + b)$.
2. *The added clause contains a relatively large number of literals.* In DPLL-based SAT solvers [3], clauses with only one unassigned literal imply values on variables. Broadly speaking, implied values from clauses are desirable [12] as they generate more implications. When a clause contains many literals, the probability of this clause implying values early in the solving process decreases. This is because many assignments must be made before all but one literal remains unassigned. Therefore, the benefits of clauses with many literals are either not noticed, or noticed very late in the SAT solving process.
3. *Too many clauses are added to the problem.* When, a relatively large number of clauses are added to the problem CNF, the overhead associated with the book keeping of clauses/literals can exceed the benefits.

The heuristics discussed here overcome the three degradation concerns described above. The first concern is addressed by first compacting all don't care states cubes using BDDs [8]. Since the DCS information is encoded in BDDs already, compacting them and extracting the DCS cubes is a trivial task. This step ensures that all don't care state clauses provide unique information to the SAT solver. Furthermore, by combining the cubes (functions), we in turn create clauses with smaller number of literals. From this point forward, we deal with the DCS cubes explicitly as opposed to symbolically within BDDs. The second and third concerns are addressed by the following filtering heuristics.

For BMC problems, each added clause corresponding to a DCS acts similar to a conflict clause. Since these clauses provide redundant information, they are most useful when containing a relatively small number of literals. As a result, we filter out and discard any DCS cubes with more than t literals. Experiments suggest that $t = 5$ is most beneficial. After the filtering process, any don't care states remaining are replicated k times and applied to all k time frames.

For BSEC problems, the don't care states information is not redundant and can be beneficial even if they contain many literals. Here, the primary focus is to add as many clauses as possible without over crowding the problem CNF. To do this, we develop a heuristic which ensures that the number of don't care state clauses does not exceed those of the original problem statement. Algorithm 1 describes the details of this procedure.

The filtering process in this algorithm, first sorts all the don't care states cubes in ascending size. Next, starting with the smallest-size cubes, each cube is converted to a CNF clause, replicated for each time frame, and added to

Algorithm 1 Filter and apply don't care states for BSEC

```

sort cubes in ascending size
#cubes_added = 0
for all cubes starting with smallest do
  if #cubes_added ≤ #gates_in_circuit then
    for each time frame do
      get time frame latch variables
      convert cube to CNF clause
      add clause to CNF problem
      #cubes_added = #cubes_added + 1
    end for
  else
    break loop
  end if
end for

```

the CNF problem. This process continues until the number of cubes added is greater than the number of total gates in the unfolded circuit. It should be noted that the time complexity of the filtering heuristic is $O(n \cdot k)$, where n is the number of cubes and k is the number of time frames. However, since k is a constant, the filtering process becomes a linear function of n and the sorting function dominates the overall procedure. As demonstrated in the experiments section, these heuristics are cost efficient and result in substantial performance improvements.

6. EXPERIMENTS

In this section we present results confirming the effectiveness of don't care states in bounded sequential problems. The experiments are conducted on a Sun Blade 100 with a 550 MHz Sparc processor and 1.6GB of memory. The BMC and BSEC platforms are built on top of the SAT solver zChaff [12]. The circuits from the ISCAS'89 benchmark suite are used for both BMC and BSEC problems. A time out of 12000 seconds is used for all experiments. The don't care states are obtained using the BDD package CUDD [15]. For all benchmarks smaller than s9234 the complete unreachable states are found. For the larger benchmarks a simple approximation technique that uses state abstraction similar to the one in [5] is employed. Here, only a subset of the state variables are considered during reachability analysis. The result is a superset of the reachable states yielding a subset of the unreachable states. For all benchmark circuits not appearing in our experiments, such as s5378 and s35932, our reachability methods did not return any don't care states.

Table 1 provides some information about the benchmarks and their corresponding DCS. The name of the benchmarks appear in column one, while column two and three contain the number of gates and latches in each benchmark, respectively. Column four shows the number of the DCS cubes while column five presents the time required to find them using approximate or exact techniques. It should be noted that finding sets of DCS often requires relatively little time when compared to the time required for BMC or BSEC which are procedures repeated multiple times in the digital VLSI design cycle. This makes the proposed methods practically viable and time efficient.

ckt name	# org gates	# latches	#DCS cubes	DCS CPU time
s298	75	14	187	0.07
s382	99	21	165	1.15
s386	118	6	10	0.01
s344	101	15	1544	11.34
s349	104	15	1705	18.6
s510	179	6	3	0.12
s526	141	21	4584	24.00
s526n	140	21	4584	24.08
s641	107	19	293	1.23
s713	139	19	293	1.22
s820	256	5	5	0.03
s832	262	5	5	0.02
s1196	388	18	3702	1.29
s1238	490	18	3414	1.19
s1488	550	6	7	0.05
s1494	558	6	7	0.04
s9234	2027	228	7	2.39
s13207	2573	669	10	1.49
s15850	3448	597	20	1.16
s38417	8709	1636	1253	299.07
s38584	11448	1452	8	7.79

Table 1: Circuit and DCS properties

6.1 BMC results

We first present the experiments for the BMC problems. Here, we use BMC to verify safety properties, where a given property is checked to hold in every time frame. Each property consists of arbitrary assignments to the state variables. In other words, we are verifying that some arbitrary state can be present in each time frame. Each problem is run 10 times with different random assignments to ensure the fair evaluation of the technique. For example, in one run, the state assignment $\{a=1, b=0, c=1\}$ is checked to hold in all time frames while in the next run the assignment $\{a=0, b=0, c=0\}$ could be checked.

The results of the BMC problems are presented in Table 2. Column one lists the names of the circuits and column two lists the bound k over which the properties are verified. The solve times for the basic BMC problem are presented in column three, while the solve times with DCS heuristic and the corresponding speed ups are presented in columns four and five respectively. These experiments demonstrate that performance improves and it never degrades. One benchmark that stands out is s820, where a performance improvement of over 600 times is observed. In this particular case, the DCS clauses directly guide the SAT solver to a solution which takes approximately 0.1s for each experiment. On the other hand the basic BMC technique without DCS information, varies in solve time from 5s to 108s. By ignoring circuit s820, the average speed up for BMC problems with the DCS heuristic is 1.51 times; however, when it is included, the improvement surpasses 32 times.

6.2 BSEC results

For BSEC problems, each benchmark circuit is generated from two identical circuits over k time frames as described in section 3.3. The bound $k = 10$ is used for all but a few

ckt name	# time frames	basic BMC CPU (sec)	DCS with heuristic CPU (sec)	speed up
s298	100	0.82	0.79	1.04
s344	100	3.84	1.59	2.42
s349	100	3.86	1.64	2.35
s382	100	8.19	6.97	1.18
s386	100	1.35	1.34	1.01
s510	100	2.08	2.07	1.00
s526	100	6.62	2.32	2.85
s526n	100	2.62	1.10	2.38
s641	100	2.47	2.40	1.03
s713	100	6.27	6.27	1.00
s820	100	708.36	1.10	643.96
s832	100	127.56	33.3	3.83
1196	100	5.14	5.05	1.02
s1238	100	4.89	4.86	1.01
s1488	50	804.74	656.56	1.23
s1494	50	687.04	575.00	1.20
s9234	50	147.93	146.10	1.01
s13207	30	15.67	15.49	1.01
s15850	30	140.78	86.05	1.64
s38417	25	32.88	32.45	1.01
s38584	25	30.84	30.6	1.01
Average				32.10
Average*				1.51

* : average taken without s820

Table 2: DCS applied to BMC problems

problems where memory or time limits were exceeded. This construction results in hard SAT instances where the entire search space must be considered.

Table 3 presents the results of the BSEC problems. The names of the circuits are listed in column one while the times to solve the basic BSEC problems are listed in column two. To compare the benefits of the DCS heuristics for BSEC, column three and four contain the times and speed ups respectively of adding the DCS to the initial time frame only. Column five and six contain the solve times and speed ups with the DCS heuristic for BSEC problems. We notice, that no benchmarks demonstrated a reduction in performance due to the DCS heuristics similar to the BMC problems. For instance, for benchmark s344, by applying the DCS to the initial time frame we see a performance improvement of 0.60 times (reduction in performance); however, when applying the BSEC heuristic to the DCS, we notice an improved performance of 1.78 times. The results of Table 3 show a substantial average performance improvement of 3.96 times is achieved.

To provide more insight into the effect of the DCS for bounded sequential problems, Figure 4 plots the sum of solve times for all benchmarks for the BSEC problems against the bound k . In this figure, the dark bars represent the time required to solve the problems without any DCS knowledge, while the lighter ones represent the solve time with the DCS heuristic for BSEC. Notice that as the bound size k increases, the performance gap between the two methods widens. This behavior illustrates that the significance of the DCS becomes more pronounced and important for larger (“harder”) problems.

ckt name	basic BSEC	1 time frame DCS CPU (sec)	speed up	with DCS heuristic CPU (sec)	speed up
s298	10.76	4.69	2.29	5.53	1.95
s382	112.17	123.45	0.91	94.08	1.19
s386	5.61	5.69	0.99	4.47	1.26
s344	18.03	30.13	0.60	10.14	1.78
s349	31.32	47.03	0.67	23.41	1.34
s510	9.14	5.21	1.75	5.72	1.60
s526	1818.16	738.14	2.46	1020.18	1.78
s526n	1127.64	254.48	4.43	722.04	1.56
s641	2448.10	1241.81	1.97	152.11	16.09
s713	1596.24	263.82	6.05	108.18	14.76
s820	568.16	631.38	0.90	478.67	1.19
s832	569.35	670.80	0.85	511.09	1.11
s1196	41.59	43.21	0.96	34.31	1.21
s1238	39.14	36.9	1.06	32.3	1.15
s1488	142.06	128.31	1.11	122.37	1.16
s1494	201.49	128.53	1.57	91.12	1.50
s9234	64.72	75.04	0.86	62.84	1.03
s13207	356.87	302.45	1.18	292.02	1.22
s15850	10318.11	463.62	22.75	411.75	25.06
s38417	timeout	timeout	-	timeout	-
s38584	3355.07	3234.42	1.04	2864.91	1.17
Average			2.72		3.96

Table 3: DCS applied to BSEC problems

7. CONCLUSION

In this work, we proposed a method of using don't care states to improve the performance of SAT solvers on bounded sequential problems. We used fast BDD techniques to extract the DCS cubes. We developed two robust heuristics which discard less beneficial DCS cubes by evaluating their ability to produce implications in SAT problems that outweigh the overhead. Due to the repetitive nature of the problems, the remaining DCS are applied to every time frame thus increasing the influence of the DCS. As a result, the DCS prune sections of the search space thus increasing the SAT solver efficiency. The benefits of DCS for both BMC and BSEC problems was demonstrated through experiments.

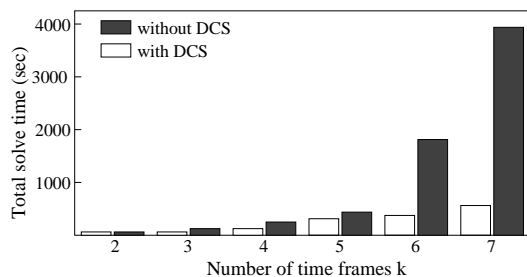


Figure 4: Solve time vs. number of time frames

8. REFERENCES

- [1] A. Biere, A. Cimatti, E.M. Clarke, O. Strichman and Y. Zhu, "Bounded Model Checking", in *Vol. 58 Advances in Computer*, Academic Press (pre-print), 2003.
- [2] G. Cabodi, S. Nocco, S. Quer, "Improving SAT-Based Bounded Model Checking by Means of BDD-Based Approximate Traversals", in *Proc. of IEEE DATE*, pp. 898-203, 2003.
- [3] M. Davis, G. Longemann and D. Loveland, "Machine Program for Theorem Proving", in *Comm. of ACM*, Vol. 5, pp. 394-397, 1962.
- [4] S. Devadas, A. Ghosh, K. Keutzer, *Logic Synthesis, Chap.3 and Chap. 7*, McGraw-Hill, 1994.
- [5] S.G. Govindaraju, D.L. Dill, A.J. Hu and M.A. Horowitz, "Approximate Reachability with BDDs using Overlapping Projections", in *Proc. of IEEE DAC*, pp. 451-456, 1998.
- [6] A. Gupta, M. Ganai, C. Wang, Z. Yang and P. Ashar, "Learning from BDDs in SAT-based Bounded Model Checking", *Proc. of IEEE DAC*, pp. 824-829, 2003.
- [7] A. Gupta, M. Ganai, C. Wang, Z. Yang, P. Ashar, "Abstraction and BDDs Complement SAT-Based BMC in DiVer", *Proc. of CAV*, pp. 206-209, 2003.
- [8] G. Hachtel and F. Somenzi, *Logic Synthesis and Verification Algorithms, Chap. 8*, Kluwer Academic Publishers, 2000.
- [9] T. Larrabee, "Test Pattern Generation Using Boolean Satisfiability," in *IEEE Trans. on CAD*, vol. 11, no. 1, pp. 4-15, 1992.
- [10] F. Lu, L.-C. Wang, K.-T. Cheng and R. Y.-Y. Huang, "A Circuit SAT Solver with Signal Correlation Guided Learning," in *Proc. of IEEE DATE*, pp. 892-897, 2003.
- [11] I. Moon, J. Jang, D. Hachtel, F. Somenzi, J. Yan and C. Pixely, "Approximate Reachability Don't Cares for CTL Model Checking", in *Proc. of IEEE ICCAD*, pp. 351-358, 1998.
- [12] M.H. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang and S. Malik, "Chaff: Engineering an Efficient SAT Solver," in *Proc. of IEEE DAC*, pp. 530-535, 2001.
- [13] K. Ravi, K. McMillan, T. Shiple and F. Somenzi, "Approximation and Decomposition of Binary Decision Diagrams", in *Proc. of IEEE DAC*, pp. 445-450, 1998.
- [14] S. Safarpour, A. Veneris, R. Drechsler and J. Lee, "Managing Don't Cares in Boolean Satisfiability", in *Proc. of IEEE DATE*, pp. 260-265, 2004.
- [15] F. Somenzi, "CUDD: CU decision diagram package", Public software, Colorado University, Boulder, 1997.
- [16] J.P.M.-Silva and K. A. Sakallah, "GRASP - A Search Algorithm for Propositional Satisfiability", in *IEEE Trans. on Computers*, vol. 48, no. 5, pp. 506-521, May 1999.
- [17] J.P.M. Silva and T. Glass, "Combinational Equivalence Checking using Satisfiability and Recursive Learning", *Proc. of IEEE DATE*, pp. 145-149, 1999.
- [18] D. Stoffel, M. Wedler P. Warkentin and W. Kunz, "Structural FSM Traversal", in *IEEE Trans. on CAD*, vol. 23, no. 5, pp. 598-619, 2004.