Validation with Guided Search of the State Space

C. Han Yang
Stanford University
Gates Building, Room 312
Stanford, CA 94305-9030
hyang@verify.stanford.edu

David L. Dill
Stanford University
Gates Building, Room 349
Stanford, CA 94305-9030
dill@cs.stanford.edu

ABSTRACT

In practice, model checkers are most useful when they find bugs, not when they prove a property. However, because large portions of the state space of the design actually satisfy the specification, model checkers devote much effort verifying correct portions of the design. In this paper, we enhance the bug-finding capability of a model checker by using heuristics to search the states that are most likely to lead to an error, first. Reductions of 1 to 3 orders of magnitude in the number of states needed to find bugs in industrial designs have been observed. Consequently, these heuristics can extend the capability of model checkers to find bugs in designs.

Keywords

Model checking, Guided search, Verification

1. Introduction

The complexity of modern chip designs has stretched the ability of the verification techniques and methodologies. Traditional verification techniques use simulators with handcrafted or random test vectors to validate the design. Unfortunately, generating handcrafted test vectors is very labor-intensive, and one is never certain which cases random testing have missed. Model checking techniques [2][7], like Murφ [3], SMV [8], are other means for checking the compliance between implementation and specification. In practice, the primary value of model checkers has been to search for design errors, not to prove correctness.

This paper proposes to optimize model checking for bug finding by using heuristics to search the part of the state space that is most likely to contain design flaws. The property being model checked is described in an assertion checker, which describes an anomalous condition. If the heuristics do find problems, they usually find the problem in substantially fewer states than conventional model checkers, which use breadth-first or depth-first search. When a design is too large for model checking to run to completion, our method is much more likely to find an error before the program exhausts available time or memory. Consequently, these guided search heuristics give the verification engineer another tool that can handle larger designs than traditional model checking.

DAC 98, June 15-19, 1998, San Francisco, CA USA
ISBN 1-58113-049-x/98/06…$5.00

Several heuristics are discussed in this paper. The first one is *Target Enlargement*, where the error states are enlarged so they can be found with less searching. The second technique is to use *Hamming distance* [4] as the search metric. The third technique, called *Tracks*, uses approximate preimages to help find the violations to assertions. Finally, the last technique uses explicit hints, called *Guideposts*, given by designer to help direct the search. Based on our experience with large industrial examples, we find that Target Enlargement combined with Tracks and Guideposts can consistently find errors much faster than breadth-first search.

The heuristics presented here are more sophisticated, more effective, and are applied to more designs than those found in [11] and [12]. Using Target Enlargement and Hamming distance for guided search were first proposed in [11]. Yuan et al. independently applied similar techniques to two simple designs. In this paper, Target Enlargement and Hamming distance are applied to a much larger set of design examples with wider range in performance.

In the following section, some terminology will be defined. Section 3 discusses the basic guided search algorithm. Section 4 discusses the designs to which these techniques were applied. In sections 5-8, each of the heuristics will be discussed. Section 9 discusses the overall experimental results. Finally, section 10 has some concluding remarks.

2. Background

A finite state machine (FSM) is a 6-tuple, $(S, I, O, \delta, \lambda, s_0)$, where S is the set of states, I is the set of input values, O is the set of output values. $\delta : S \times I \rightarrow S$ is the next state function, $\lambda : S \times I \rightarrow O$ is the set of output function, and $s_0 \in S$ is the initial state. In this paper, let $B = \{0, 1\}$. V is the set of Boolean state variables in the system. Each state is a bit vector, which we represent mathematically as a map from the set of state variables to Boolean values. Hence, S is the set of all such bit vectors, which we write as $[V \rightarrow B]$.

If A is a set of states in S, the preimage of A under δ is defined as follows.

Definition 1: Preimage

$$PreImage(\delta, A) = \{s \mid \exists i \in I . \exists s' \in A . s' = \delta(s, I)\}$$

In essence, *PreImage* is the set of states that can reach a state in A in one transition.

The projection of a set of state onto another set of states with respect to a set of variables V_i is defined as follows.

Definition 2: Projection

$$Proj_{V_i}(T) = \{t' \in [V \rightarrow B] \mid \exists t \in T . \forall v \in V_i . t'(v) = t(v)\}$$

In other words, the projection of T onto V_i is the set of bit vectors, which have the same values for variables in V_i as some bit vector in T (and have *any* value for variables not in V_i).

3. Guided Search Algorithm

The guided search algorithms are implemented in a verifier called Murφ++. Designs can be described directly in Murφ++ (which is similar to C++) or be translated from Verilog [5]. The algorithm for the verifier is shown in Figure 1. During the search, there are two types of states, *explored states* and *visited states*. Explored states are those that have their next states generated, and visited states are those that have been encountered in the search but whose next states have *not* been generated. Given an initial assignment to all the state variables that is in the PriorityQueue, Murφ++ retrieves a visited state and finds all next states. Because Murφ++ is able to represent all combinations of inputs symbolically, all next states are found at once. Then, the evaluation function is used to compute the score for each next state. If the state has not been seen before, then all values for the state variables are stored in a hash table, and the state is placed in the priority queue. For breadth-first search, the evaluation function is simply a constant. Otherwise, the search heuristic will provide a score for each next state, and the state is stored in the priority queue according to its score. Backtracking is used to find all reachable states. When no more states are found, the verification process is complete. Assertion checker, which describes what should not happen in the design, is also coded as part of the description. If an assertion is violated during the generation of the reachable states, an error trace is printed out.

```
1. Put Start State in PriorityQueue
2. While (PriorityQueue not empty)
3.     Dequeue State from Priority-Queue
4.     Find set of Next States
5.     Use Evaluation Function to find
       score for each Next State
6.     If Next States not in HashTable
7.         Put into PriorityQueue
8.     Check for Assertion violation for
           each Next State
```

Figure 1 Guided Search Algorithm

4. Benchmark Designs

Several realistic industrial designs are used to evaluate the effectiveness of the heuristics. They are briefly described below.

4.1 S3.mp Data Link Level Protocol

Sun Microsystems' S3.mp distributed shared memory computer uses workstations connected with a gigabit serial link to form a parallel computer [9]. The data link level protocol used in the serial link, which operates over a slotted ring [10], was first developed in the Murφ verification system [3]. The description of this protocol is about 1200 lines of Murφ++ code. For a ring with 4, 6, and 8 slots, there are 61, 85, and 107 state bits, respectively.

4.2 FLASH Examples

Many of the benchmarks are portions of the Stanford FLASH (FLexible Architecture for Shared memory) multiprocessor [6]. These designs were all converted from the original Verilog code that was annotated to extract the control logic [5]. Many of the assertions were also translated directly from the FLASH validation suite in Verilog. All of the bugs that we have found were actually encountered by the FLASH design team through their validation testing.

Table 1 lists some information about each design. There are two versions of MC2, big and small. The only difference between the two is the range of inputs that were considered. For the smaller designs, it is easier to discuss some of the behavior of the heuristics (in Section 7.1).

Design	State Bits	Min Cycles to Violate Assertion
Inbox with Cache Control (Inbox)	174	5
Memory Controller State Machine (MC1)	39	17
Memory Controller with SDRAM Control (MC2)	180	39
Load/Store Control (LSC)	112	49

Table 1 Flash Benchmarks

5. Target Enlargement

As the name suggests, Target Enlargement is an effort to make the set of states that will violate the assertion, called error states, bigger.[1] The preimage of these error states is the set of states that in one cycle can reach an error state. If it is possible to reach a state in the preimage from a start state, then it is also possible to reach an error state. Each successive preimage potentially describes an even larger set of states that can reach the error states. The larger target increases the opportunity for the guided search to find a path to the error states and consequently reduces the amount of searching that needs to be done. Because computing preimages can be a memory intensive operation, Target Enlargement computes successive preimages until a given computer memory limitation is reached.

5.1 Target Enlargement Experimental Results

Table 2 shows the number of cycles of target enlargement that was possible with each of the designs within 200 Mbytes of memory. While the size of the Binary Decision Diagrams(BDDs) is not very large, computing the next larger preimage for all the examples except Inbox, MC1, and MC2 (Small), exceeded the memory limit [1].

Design	Max Cycles of Target Enlargement	BDD Size of Largest Target Enlargement
CM (4 Slots)	4	55,301
CM (6 Slots)	4	33,374
CM (8 Slots)	4	25,235
Inbox	5	28
MC1	5	95
MC2 (Small)	4	1,457
MC2 (Big)	6	23,885
LSC	4	13,172

Table 2 Target Enlargement

Table 3 shows the number of visited states and explored states. The number of visited states includes those that eventually became explored states. The number of visited states is an indication of the size of the hash table that holds the reachable states. The number of explored states is an indication of the efficiency of the search heuristic. Fewer explored states would mean a better search heuristic. For some designs, like MC2 (Big),

[1] Yuan et al. refer to *target enlargement* as *retrograde analysis*.

it was not possible to find the set of reachable states within 160 Mbytes of memory.

Design	Visited States		Explored States	
	Breadth-first Search	Target Enlargement	Breadth-first Search	Target Enlargement
CM (4 Slots)	32,773	197	12,784	83
CM (6 Slots)	386,994	17,650	379,088	6,437
CM (8 Slots)	>2,362,324	>2,362,324	>432,177	>145,498
Inbox	77,313	38,913	35,842	12,802
MC1	117,889	57,089	90,320	47,874
MC2 (Small)	161,577	119,802	149,210	113,920
MC2 (Big)	>4,799,484	>4,799,484	>2,278,144	>2,278,144
LSC	246,553	224,767	237,508	223,744

Table 3 Target Enlargement Experimental Results

Table 3 shows that Target Enlargement can have significant reductions in the number of visited and explored states to find a violation of the assertions. While Yuan et al. also reported some reductions in the number of states needed to find bugs on two small examples, Table 3 illustrates the use of Target Enlargement on a larger and more realistic set of examples [12]. In addition, the reduction on this set of examples can be up to two orders of magnitude, which is much larger than shown by Yuan et al.. Unfortunately, the effectiveness of Target Enlargement varies dramatically from one design to another.

5.2 Analysis of Target Enlargement

Target Enlargement is most effective when it substantially increases the probability of visiting a target state, assuming a search that randomly generates inputs. Consider an example where Target Enlargement is not very effective: Suppose there is a sequence of *n* pipeline registers between the point where an erroneous signal is generated and the assertion that is eventually violated by the signal. In this case, the probability of hitting an error becomes 1 when the signal is generated *n* cycles before it is actually detected. Target Enlargement will not improve the search significantly until $n + 1$ cycles of preimages have been computed. One of our benchmarks, MC2, has this behavior. On the other hand, if one out of a thousand inputs reaches an error state in the next cycle, one cycle of target enlargement will increase the probability of visiting the error state by a thousand fold. MC2's first cycle of target enlargement exhibits this behavior. Because Target Enlargement's ability to increase the error probability strongly depends on the design, the location of the inputs and the location of the assertions, the performance of Target Enlargement varies widely across the benchmark designs. Despite the dependency on the design, Target Enlargement consistently reduces the number of states needed to find violation to the assertions; consequently, it is used in conjunction with all other heuristics described later in this paper.

6. Hamming Distance

The first search heuristic is Hamming distance, which is defined as the number of bits that are different between two bit vectors [4]. In generating the set of reachable states, Hamming distance can be used as a search metric [11][12]. Those states that have the lowest Hamming distance to the largest Target Enlargement are explored first. In essence, we hope states with very few bits differing from the enlarged target will require very few cycles to reach that target. The minimum Hamming distance between a single state and the enlarged target is computed using an algorithm that has a complexity which is linear with respect to the size of the BDD of the enlarged target.

6.1 Hamming Distance Experimental Results

Table 4 and Table 5 show the result for applying Hamming distance to the benchmark designs. At first, the Hamming distance is measured against the largest enlarged target. In the case of the communication module, this simple heuristic worked exceptionally well. In the case of CM with 6 and 8 slots, the reduction in the number of states found was about 1 to 2 orders of magnitude. In other designs, Hamming distance's performance was less stellar. For the FLASH examples, the reduction in the number of states for most designs ranged from 10-50%. For the case of LSC, the result was actually worse than Target Enlargement.

Design	Visited States		
	Target Enlargement	Hamming Distance (Large Target) + Target Enlargement	Hamming Distance (Small Target) + Target Enlargement
CM (4 Slots)	197	197	179
CM (6 Slots)	17,650	17,176	1,092
CM (8 Slots)	>2,362,324	417,671	5,947
Inbox	38,913	38,913	12,913
MC1	57,089	38,913	28,161
MC2 (Small)	119,802	119,802	94,618
MC2 (Big)	>4,799,484	>4,793,500	>4,793,500
LSC	224,767	343,128	225,344

Table 4 Hamming Distance Visited States

Design	Explored States		
	Target Enlargement	Hamming Distance (Large Target) + Target Enlargement	Hamming Distance (Small Target) + Target Enlargement
CM (4 Slots)	83	81	83
CM (6 Slots)	6,437	6,348	499
CM (8 Slots)	>145,498	901	2,735
Inbox	12,802	12,802	12,802
MC1	47,874	22,019	8,451
MC2 (Small)	113,920	113,594	88,442
MC2 (Big)	>2,278,144	>2,274,048	>2,260,480
LSC	223,744	245,817	222,357

Table 5 Hamming Distance Explored States

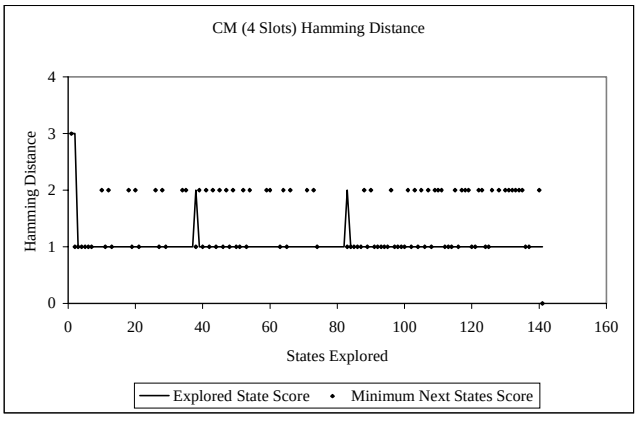

Figure 2 Hamming Distance Example

For example, Figure 2 shows the explored state's Hamming distance state and the minimum Hamming distance of its next states for a small example, CM (4 Slots). The start state has a relatively large Hamming distance. Soon, the search rapidly reduced the Hamming distance to 1. Unfortunately, these states did not lead to states that were even closer. Eventually, the search had to resort to states with Hamming distance of 2. After

exhausting more states with Hamming distance of 1, the search again had to find a state with distance 2. Eventually, the search found the enlarged target. Throughout the search, all the next states had a Hamming distance of 1 or 2. This is typical of Hamming distance, which tends not to be very discriminating.

Our experiments show an interesting interaction between Hamming distance and Target Enlargement. While Hamming distance is "aiming" toward a target, Target Enlargement is really making the target larger and therefore less discriminating, which negates some of the effectiveness of Hamming distance. In fact, when Hamming distance is "aimed" toward the smallest target, which is the least fuzzy, the results were better. Using this technique, another order of magnitude of improvement can be achieved with the CM. In addition, all the FLASH benchmarks also showed improvement over Hamming distance with large target.

Based on these experimental results, it appears that Hamming distance can be a useful heuristic in finding the violations of assertions. On a few small examples, Yuan et al. reported modest reduction in number of steps needed to find the error [12]. This much larger set of design benchmarks suggests that while Hamming distance can reduce the number of states needed to find the error by up to several orders of magnitude, its performance is also very inconsistent across designs.

7. Tracks

As can be seen from the results in Section 5, for many of the designs, it is not possible to do Target Enlargement for many cycles. However, *approximate* preimages that are based on a subset of the state variables may be computed for more cycles than Target Enlargement. In practice, a subset of the state variables can control most of the behavior of the design. This subset may be the state variables from a few of the state machines within the design, or may be a few key state variables within a FSM that dictates its behavior. For example, in the control logic of FSMs, the Verilog description usually has a **`switch`** statement that depends on a state variable that determines much of the behavior. Track computes a series of approximate preimages based on a given set of variables that strongly control the behavior of the system. With multiple tracks implicitly conjoined, it may be possible to construct a sufficiently accurate preimage that aids the guided search.

Tracks are defined formally in Equation 1.

Equation 1 Track Computation

$$T_i^{k+1} = PreImage[\delta, Proj_{V_i}(T_i^k)]$$

where T is one layer of one track, i is the Track number, V_i is the variables in Track i, and k is the layer number, or number of cycles away from the largest enlarged target.

T_i^0 is the largest enlarged target. In essence, the next layer of a track is the preimage of the projection of the last layer onto the variables in the track. Because all the variables that are not in the track are projected out, the resulting preimage is always a superset of the true preimage. The projection also reduces the size of the BDD and thus enables computation of more preimages.

Our experience shows that the following guidelines are useful in choosing the variables in a track.

1. Identify the main FSMs in the design
2. Identify the main state variables in the FSMs
3. Rank other variables that the designer deem "important"
4. Start with the assertion and working backward toward primary inputs, include as many FSMs as possible. The size of the partition depends on the size of the BDDs for each of the preimages.

More than one track can be used to approximate the true preimage more accurately. In fact, tracks can have overlapping state variables. During the guided search, the state's score depends on to which layer the state belongs. The score is the *least* layer number in which *all* tracks contain the state. This score is the minimum cycle number that satisfies the implicit conjunction of all the tracks. Consequently, this evaluation function greedily chooses the layer number that is closest to the target. States that have the smallest score, or those that are closest to the enlarged target, are explored first.

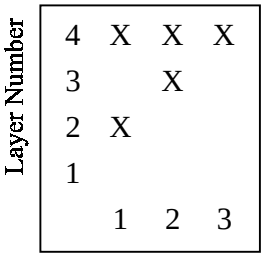

Figure 3 Tracks Evaluation Function Example

As an example of the evaluation function, Figure 3 shows a state that is in layers 2, 3, and 4 of three tracks. Because the tracks are approximate preimages, a state can actually belong to multiple layers of the same track. The evaluation function returns a value of 4 since all tracks agree that it is the minimum layer number.

7.1 Tracks Experimental Results

For most of the designs, Tracks worked quite well across all designs except LSC (Table 6).

	Visited States		Explored States	
Design	**Target Enlargement**	**Tracks + Target Enlargement**	**Target Enlargement**	**Tracks + Target Enlargement**
CM (4 Slots)	197	99	83	41
CM (6 Slots)	17,650	1,655	6,437	644
CM (8 Slots)	>2,362,324	2,313	>145,498	969
Inbox	38,913	4,162	12,802	5
MC1	57,089	11,394	47,874	76
MC2 (Small)	119,802	1,498	113,920	161
MC2 (Big)	>4,799,484	17,474	>2,278,144	4,283
LSC	224,767	210,946	223,744	208,228

Table 6 Tracks Experimental Results

Figure 4 shows the score for MC2 (Small) during the search. Because MC2 (Small) has relatively few nondeterministic inputs, it was possible to compute preimages from the assertion to the start state and plot the true distance of the explored states to the enlarged target. This example illustrates the effectiveness of the approximate preimages. As can be seen in Figure 4, the score produced by Tracks initially was routinely off by 10 cycles or more. If all states were uniformly inaccurate, then the score still would give the right guidance to the search. In fact, Figure 4 shows that the curve between the explored state score and the actual distance to have similar shape.

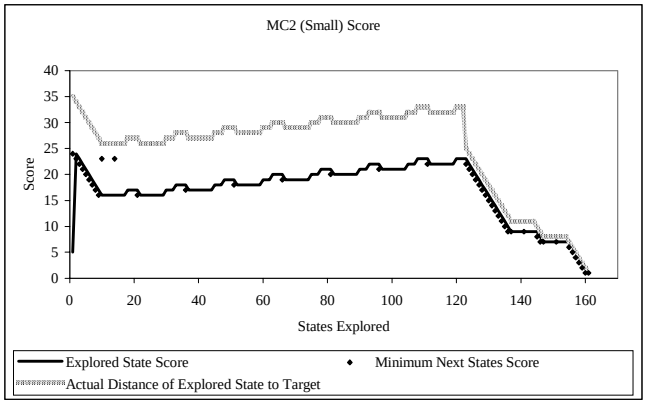

Figure 4 Tracks Score Compared with True Score

However, there was a local minimum introduced in the search due to varying amount of inaccuracy occurring at exploring states 10 and 14. Because of this local minimum, the search explored states that had good score, but were not able to lead to the enlarged target. At state 123, the search finally went back to the states that were found when exploring states 10 and 14. Eventually, exploring these resulted in finding states that eventually lead to the target zone.

Tracks allow the verification engineer to trade off between accuracy and size of BDDs. While a track with more state bits offers more accuracy, the size of the BDD usually grows faster. When the size of the BDD is too large, the verification engineer can remove some state bits and introduce several new overlapping tracks to compensate for the loss in accuracy. Consequently, Tracks enables the verification engineer to give high-level guidance to the search while working within the computer memory limitation.

Unfortunately, Tracks did not work very well with the LSC design. This particular design had many very small interacting FSMs. The other design usually had one large FSM that controlled the behavior of the system. Because of the large number of FSMs, it was very difficult to pick a series of tracks that were able to compute preimages with any accuracy. Consequently, Guidepost was added to help in the search for LSC.

8. Guidepost

In addition to more automated heuristics mentioned previously, designers can also provide *hints* to the guided search. The hints, called *Guideposts*, are a series of conditions that the designer believe to be interesting or even required preconditions for the assertion checker to be violated.

Guidepost encodes the number of hints that the search has gone through in the score itself. Consequently, the score for a child state is dependent on the score of the parent state *and* on the properties in the child state. Equation 2 shows the evaluation function for Guideposts. Just like the evaluation function for Tracks, a lower score is deemed to be a state that is closer to the target. In other words, the passage of more guideposts should indicate a lower score, which is simply the number of guideposts yet to be encountered or ($T_{Guideposts} - P_{Guideposts}$). The multiplication of this term with M_{Cycles} ensures that the number past guideposts dominates over the score from Tracks. In fact, the Track score only has an effect when the number of previous guideposts is the same. With this kind of evaluation function, the score is a summary of the history of the hints that the search has encountered.

Equation 2 Guidepost Evaluation Function

$$Score = (T_{Guideposts} - P_{Guideposts}) * M_{Cycles} + Score_{Tracks}$$

Where $T_{Guideposts}$ is the total number of guideposts

$P_{Guideposts}$ is the number of guideposts that the current state and its ancestors have past through

M_{Cycles} is the maximum number of cycles in all the tracks.

$Score_{Tracks}$ is the Score from Tracks evaluation function

This evaluation function biases the search towards going through the guideposts. As the search encounters a new hint, the score improves according to the number of hints that it has encountered in the past. Consequently, the score only improves when the hints are found in the prescribed total order.

8.1 Guidepost Experimental Results

Table 7 shows the results for Guidepost. Only one guidepost was used for each design benchmark. Many of the guideposts were only to help the search to get out of a local minimum that was introduced by Tracks.

In the LSC, many FSMs control the complex operations of handling normal loading and storing of data from the cache, managing the store buffers and exceptions such as loading before storing. In the LSC, the condition of the guidepost was the FSM in the state ready to start a cache refill operation and starting a counter that would indicate the completion of the cache refill. With this simple condition, the enlarged target for the LSC can be found in about 15% of the states needed for Target Enlargement.

	Visited States		Explored States	
Design	**Target Enlargement**	**Guideposts + Target Enlargement**	**Target Enlargement**	**Guideposts + Target Enlargement**
CM (4 Slots)	197	94	83	38
CM (6 Slots)	17,650	1,650	6,437	641
CM (8 Slots)	>2,362,324	2,296	>145,498	955
Inbox	38,913	4,162	12,802	5
MC1	57,089	11,394	47,874	12
MC2 (Small)	119,802	1,834	113,920	39
MC2 (Big)	>4,799,484	9,602	>2,278,144	289
LSC	224,767	37,070	223,744	30,503

Table 7 Guidepost Experimental Results

9. Comparison of Heuristics

Figure 5 shows the number of visited states as a percentage of the breadth-first states. Similarly, Figure 6 shows percentages for the explored states (Note that the vertical axes in both of these figures are in logarithmic scale). As can be seen, Target Enlargement always reduces the search space except CM (8 Slots) and MC2 (Big), which were simply too big to find the error using Target Enlargement alone. Hamming distance with the large target did not perform well consistently; however, Hamming distance with small target did show reductions in either explored or visited states for all designs. With Tracks and Guidepost, the error states was found consistently faster.

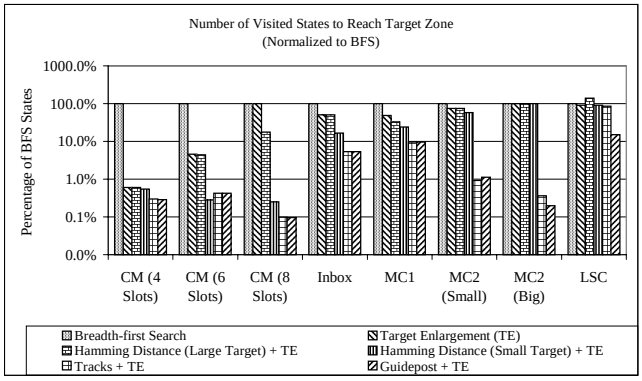

Figure 5 States Visited as a Percentage of Breadth-first Search

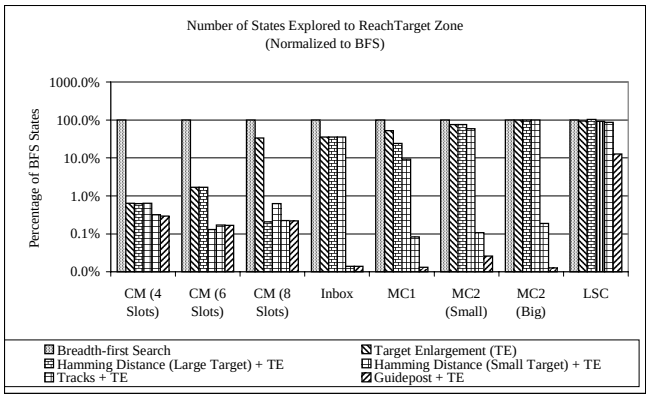

Figure 6 States Explored as a Percentage of Breadth-first Search

10. Conclusion and Future Work

Several heuristics have demonstrated to be very effective in guiding the search to find violations to assertions in realistic designs. These heuristics can find the errors in much fewer states than the breadth-first search that is used in model checking. These techniques extend the capability of model checking while providing a more powerful tool than conventional simulation.

Hamming distance is relatively easy to implement; however, its effectiveness is rather inconsistent. Techniques like Tracks and Guidepost offer more robustness. Unfortunately, they also require more designer input. This set of heuristics offers the verification engineer tradeoffs for manual labor and effectiveness in the guided search.

Guidepost can be extended to allow for creation of high level test vectors. The verification engineer only needs to specify the critical events that a test vector needs to go through. Then, the heuristic can go through the design and fill in a set of inputs that will lead the design through those critical events. Being able to specify such a high level test vectors should improve the productivity of verification engineers and merits further study.

11. Acknowledgements

Many thanks for David Nakahira and Jules Bergmann for their help in the memory controller and load/store control design examples, respectively. This work was sponsored under contract number DABT63-95-C-0049-P00005. The content of this paper does not necessarily reflect the position of the policy of the Government and no official endorsement should be inferred.

12. References

1. R. E. Bryant, Graph-based Algorithms for Boolean function Manipulation. IEEE Transactions on Computers, 6(C-35):677-691, August 1986
2. E. M. Clarke, E. A. Emerson, A. P. Sistla, *Automatic Verification of Finite-state Concurrent Systems using Temporal Logic Specifications*, ACM Transactions on Programming Languages and Systems, vol. 8, no. 2 pp. 244-63
3. D. L. Dill, A. J. Drexler, A. J. Hu, and C. H. Yang, *Protocol Verification as a Hardware Design Aid*. 1992 ICCD, pp.522-525
4. R. W. Hamming. *Error Detecting and Error Correcting Codes.* Bell System Tech Journal, 9:147-160, April 1950.
5. R. C. Ho, C. H. Yang, M. A. Horowitz, and D. L. Dill. *Architecture Validation for Processors.*, 1995 ISCA, pp. 403-413
6. J. Kuskin, D. Ofelt, M. Heinrich, J. Heinlein, R. Simoni, K. Gharachorloo, J. Chapin, D. Nakahira, J. Baxter, M. Horowitz, A. Gupta, M. Rosunblum, and J. Hennesy. *The Stanford FLASH Multiprocessor*. 1994 ISCA, pp. 302-313
7. F. Maruyama, M. Fujita, *Hardware Verification*, Computer, vol. 18, no. 2, pp 22-32, Feb 1985.
8. K. L. McMillan, *Symbolic Model Checking*. Ph.D. thesis, School of Computer Science, Carnegie Mellon University, May 1992.
9. A. G. Nowatzyk, M. C. Browne, E. J. Kelly, and M. Parkin. *S-connect: From Network of Workstations to Supercomputer Performance*. 1995 ISCA, pp. 71-82
10. A. S. Tanenbaum, *Computer Networks*, chapter 7, pp. 312-313. Prentice-Hall Inc., 1981
11. C. H. Yang, D. L. Dill, *SpotLight: Best-First Search of FSM State Space.* IEEE International High Level Design Validation and Test Workshop, presented on November 16, 1996, http://verify.stanford.edu/hyang/hldvtSlides.ps
12. J. Yuan, J. Shen, J. Abraham, A. Aziz, *On Combining Formal and Informal Verification*, 1997 CAV, pp. 376-387