

Chapter 4. Hamming Distance

4.1 Introduction

Traditional model checkers have used breadth-first or depth-first search to traverse through the design's state space [6, 18]. While these search techniques are very thorough, they are not optimized towards finding bugs in large state spaces. In the field of artificial intelligence, where is also a need to search through large state spaces, there is the notion of using evaluation functions to heuristically search a large space in a "hill climbing" or best-first search fashion [29]. Since Hamming distance is an easy-to-define metric, it is used to search the state space for design flaws.

4.2 Using Hamming Distance as an Evaluation Function

Hamming distance was originally conceived for detection and correction of errors in digital communication [10]. It is simply defined as the number of bits that are different between two bit vectors [10]. In the context of prioritized model checking, the minimum Hamming distance between the state being explored and the set of error states is used as an evaluation function to guide the search.

The intuition is that those states that have a lower Hamming distance to the largest enlarged target are explored first. Hopefully, the states with very few bits differing from the enlarged target will require very few cycles to reach that target. While the distance in the state graph between a state and the largest enlarged error target is not necessarily correlated to their Hamming distance, when the state has reached an enlarged target, the Hamming distance is zero. Consequently, Hamming distance can be valuable although crude evaluation function.

4.3 Computing the Minimum Hamming Distance

To use Hamming distance as an evaluation function, an algorithm to compute the *minimum* Hamming distance between a single state and a set of states was developed. This algorithm is one of the evaluation functions for the prioritized model checking algorithm as shown in Figure 4, page 16. Figure 21, page 42 shows the algorithm that finds the minimum Hamming distance between a single state, P and a set of states, S ; both of which are described by a BDD. Since BDD S may have eliminated redundant variables, it may *not* have a node at every level. However, BDD P is a single state: Its BDD has exactly 1 path to True and has one BDD node at *every* level.

Computing the Hamming distance requires recursively traversing each level of BDD P and S in lock step. Step 1 is the base case, where the Hamming distance computation is done. In step 2, the traversal of the S BDD has resulted in going out of the set S ; consequently, infinity is returned as the Hamming distance. If both P and S are on the same BDD variable, then the algorithm computes the Hamming distance of the subgraphs and returns the minimum. In steps 6 and 12, because BDD P and BDD S have the same bit values, the Hamming distance is simply that of the subgraph.

Figure 21. Hamming Distance Evaluation Function

```
HammingDistanceEvaluationFunction(BDD p, BDD s)
1.  if ((p == True) && (s == True))
    return 0;
2.  if (s == False)
    // The state being described is no longer in S,
    // so return a large number
    return infinity;

3.  if (StoredHammingDist(s))
    // The Hamming Distance of s has been previously
    // computed and stored
    return (StoredHammingDist(s));

4.  if (BDDVariable(p) == BDDVariable(s))
    // p and s are on the same BDD variable
5.  if (BDDThen(p) == False)
    // BDDElse(p) has the path to True
6.  // since p and s have the same bit value,
    // the Hamming distance is that of the subgraph
    d0 = HammingDistance(BDDElse(p), BDDElse(s));
7.  // p and s have different bit values,
    // Hamming distance is that of subgraph plus 1.
    d1 = HammingDistance(BDDElse(p), BDDThen(s)) + 1;
8.  // Store the result for future reference
    StoredHammingDist(s) = Min(d0, d1);
9.  return (StoredHammingDist(s));
10. else
    // BDDThen(p) has the path to True
11. d0 = HammingDistance(BDDThen(p), BDDElse(p)) + 1;
12. d1 = HammingDistance(BDDThen(p), BDDThen(p));
13. StoredHammingDist(s) = Min(d0, d1);
14. return (StoredHammingDist(s));
15. else
    // p and s are not on the same BDD variable
16. if (BDDThen(p) == False)
    // BDDElse(p) has the path to True
17. p' = BDDElse(p);
18. else
19. p' = BDDThen(p);
20. return HammingDistance (p', s)
```

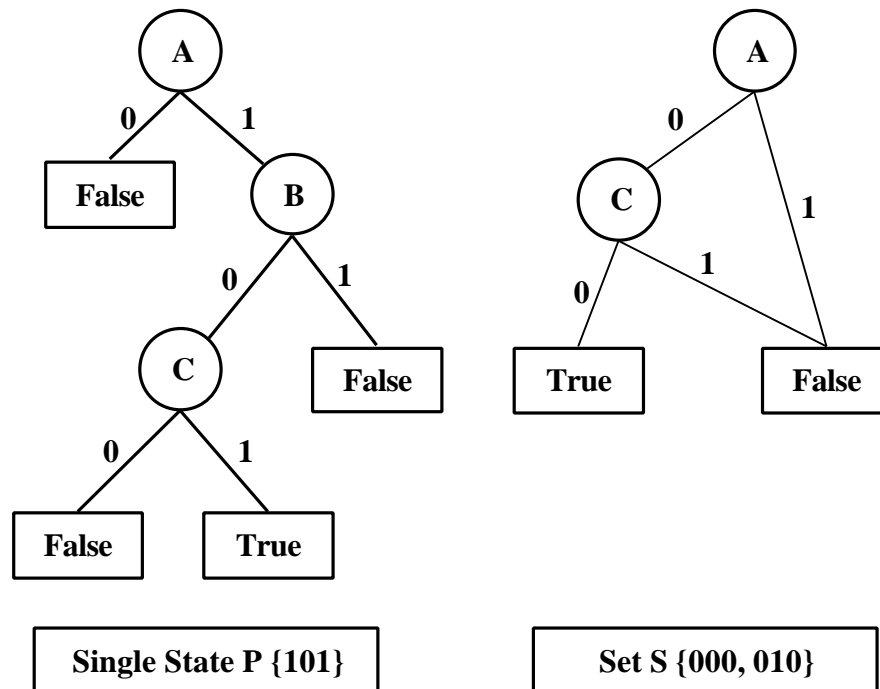
However, in steps 7 and 11, where the bit values are different, then the Hamming distance is that of the subgraph added to the one bit that is different. After the Hamming distance has been computed for each node, they are stored for future reference since many paths from the root to the leaf may share common nodes (steps 3, 8 and 13). When BDD S has a redundant node (steps 16-20), the Hamming distance is simply that of the

next variable since a redundant node is simply both children having the same subgraph. Since this algorithm traverses through the BDD S , the complexity is $O(\text{Nodes in } S)$.

4.3.1 Computing Minimum Hamming Distance Example

Figure 22 shows an example of the Hamming distance algorithm. State P is a single state $\{101\}$, and Set S has $\{000\}$ and $\{010\}$.

Figure 22. Hamming Distance Example



1. **Variable A.** If $A = 1$, then the bit value for variable A is the same for P and S . Unfortunately, with $A = 1$, BDD S immediately reaches `False`, and the Hamming distance for this subgraph is infinity. With $A = 0$, then the bit value is different for the 2 BDDs. Consequently, the Hamming distance is that of the subgraph plus 1.
2. **Variable B.** BDD S has eliminated the redundant node B . Thus, the algorithm continues with the next variable.

3. **Variable C.** If $C = 1$, BDD S again reaches `False`: The Hamming distance is again infinity. However, if $C = 0$, then the bit value for P and S will be different. So, the Hamming distance is 1. Returning this result to the parent node A , the minimum overall Hamming distance is the distance at node C added to the distance at node A . Therefore, the minimum Hamming distance is 2.

4.4 Previous Hamming Distance Results

Yuan *et al.* have demonstrated using Hamming distance with Target Enlargement to find bugs [31]. On a single design, they were able to show that using Hamming distance with Target Enlargement was able to reduce the number of explored states in random simulation from 50 states to 3.

4.5 Hamming Distance Experimental Results

Table 7, page 45 lists the result for applying Hamming distance to the benchmark designs. These results are also shown in Figure 23, page 45 and Figure 24, page 46. All the Hamming distances are measured against the largest enlarged target. In the case of the communication module, this simple heuristic worked exceptionally well. In CM with 6 and 8 slots, the reduction in the number of states found was about 1 to 2 orders of magnitude, and the bug in CM (8 Slots) was found for the first time. In other designs, Hamming distance's performance was less stellar. Even with the help of Hamming distance, the bug in MC2 (Big) was not found within the memory limit of 160 Mbytes. While Hamming distance reduced the number of visited states from 10-50% for most other FLASH designs, the result was actually worse than Target Enlargement in the case of LSC.

Table 7: Hamming Distance (Large Target) Experimental Results

Design	Visited States		Explored States	
	Target Enlargement	Hamming Distance (Large Target) + Target Enlargement	Target Enlargement	Hamming Distance (Large Target) + Target Enlargement
CM (4 Slots)	197	197	83	81
CM (6 Slots)	17,650	17,176	6,437	6,348
CM (8 Slots)	>2,362,324	417,671	>145,498	901
Inbox	38,913	38,913	12,802	12,802
MC1	57,089	38,913	47,874	22,019
MC2 (Small)	119,802	119,802	113,920	113,594
MC2 (Big)	>4,799,484	>4,793,500	>2,278,144	>2,274,048
LSC	224,767	343,128	223,744	245,817

Figure 23. Hamming Distance (Large Target) Visited States

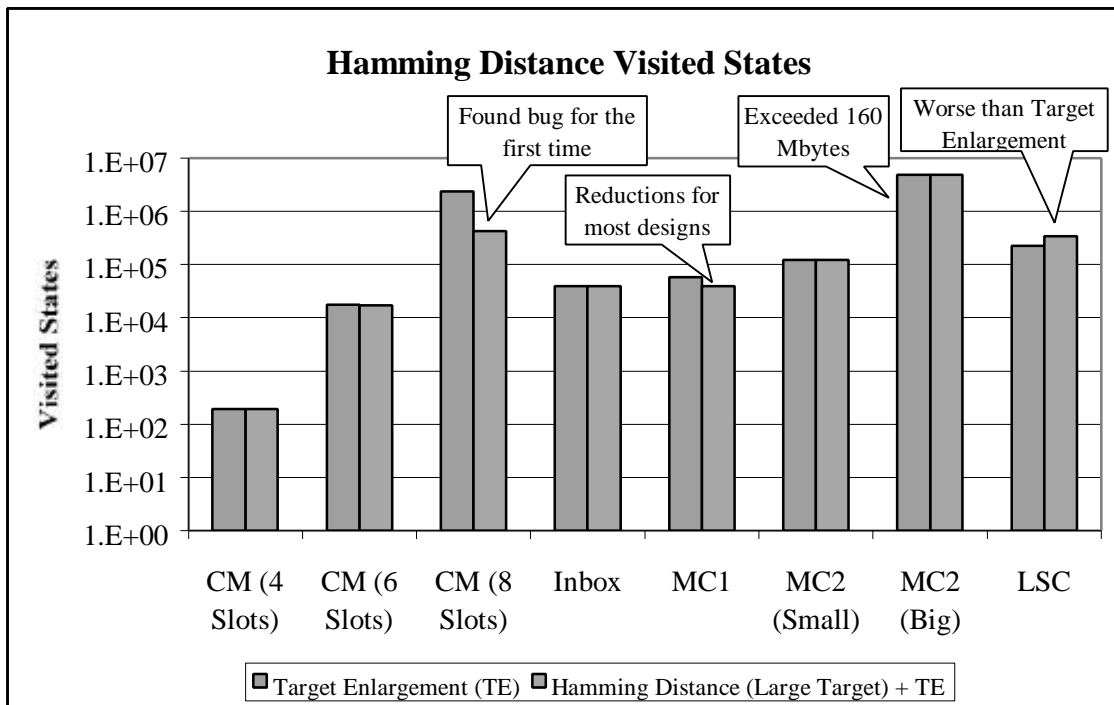
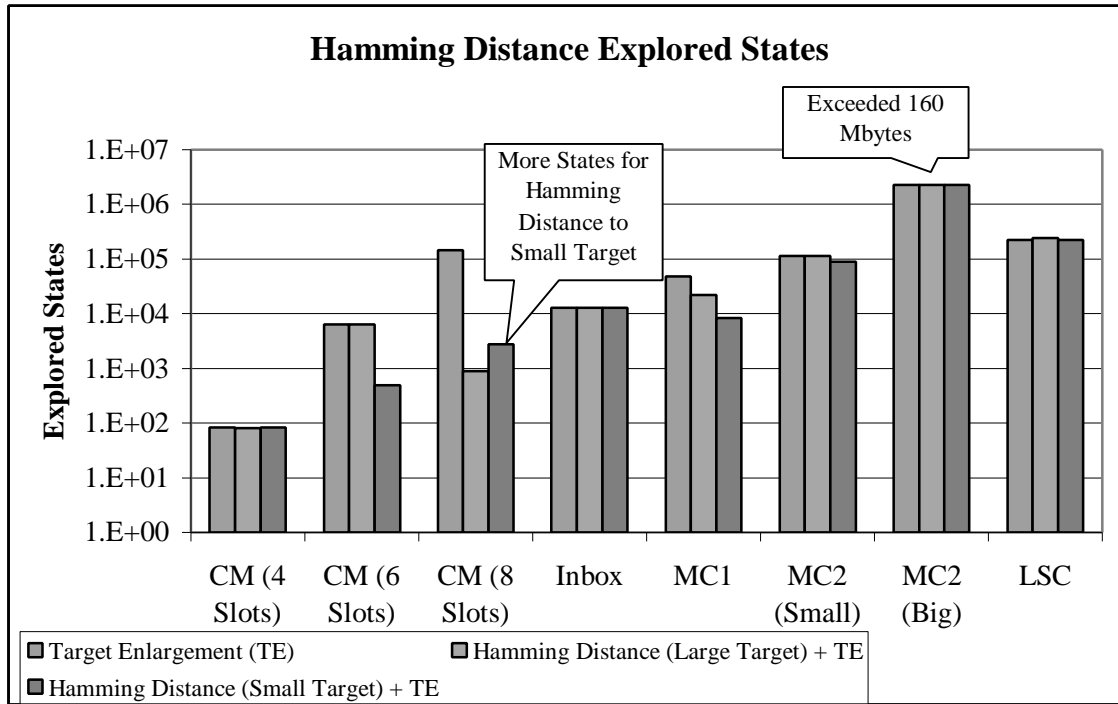


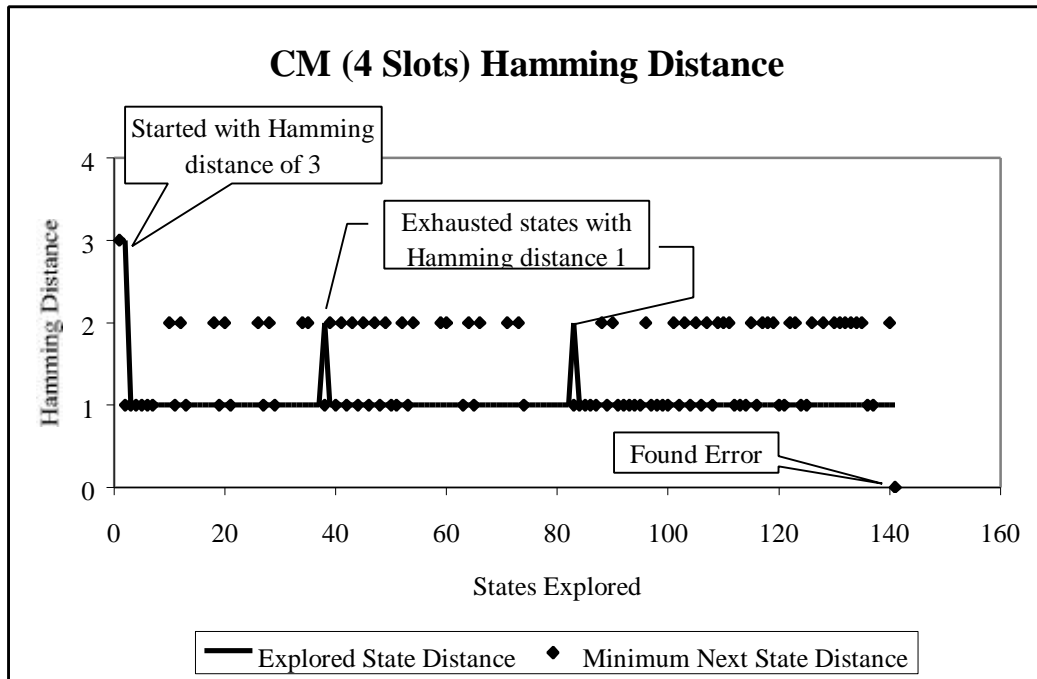
Figure 24. Hamming Distance (Large Target) Explored States



4.6 Analysis of Hamming Distance

Since there are large discrepancies in the reduction of visited and explored states among the designs, a more detailed study was made in how Hamming distance influence the search. Figure 25, page 47 shows the Hamming distance during the search. The reset state started with a Hamming distance of 3. Then, it soon found states with Hamming distance 1. At two separate occasions during the search, the priority search exhausted all states with Hamming distance 1 and had to resort to states with Hamming distance 2. Eventually, the search found the error state.

Figure 25. Analysis of Hamming Distance



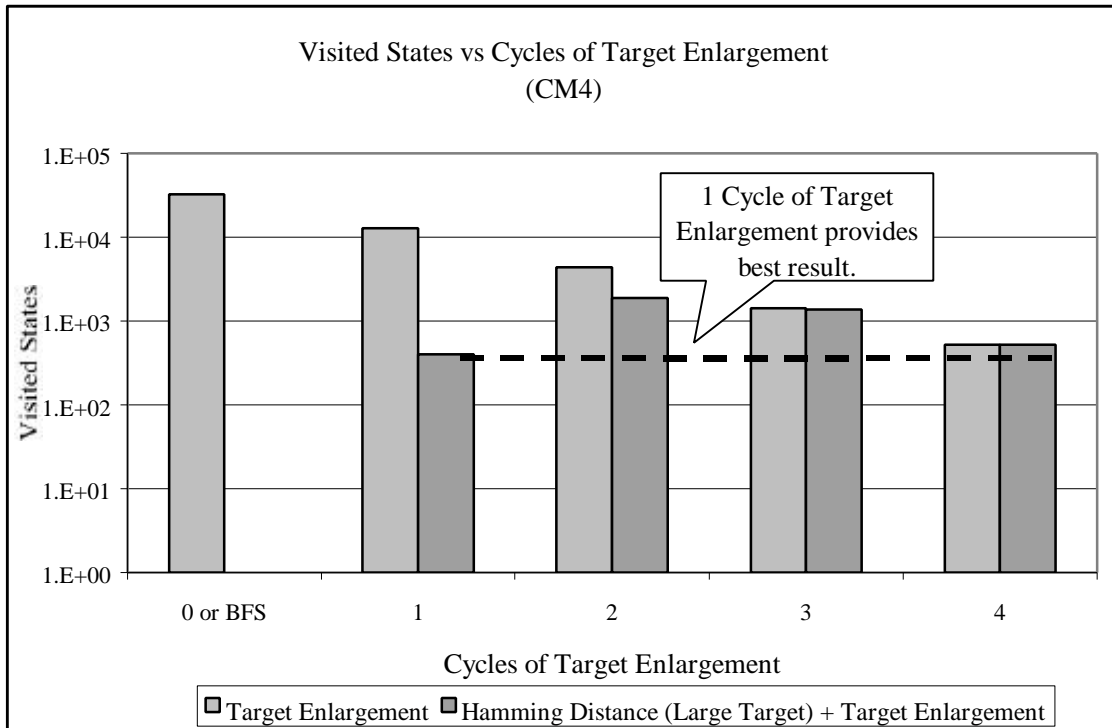
Since much of the curve in Figure 25 is flat and has a distance of 1, it is clear that Hamming distance is not very discriminating in the states that are explored, and it does not really guide the search toward the error. Despite the poor level of discrimination, Hamming distance was able to show dramatic reduction in the number of states needed to be visited and explored. This fact reaffirms the suspicion that exploring the state space in a breadth-first fashion is very inefficient in finding bugs. Any amount of discrimination, even that provided by Hamming distance, can help to reduce the number of states needed to find a bug.

4.7 An Improved Hamming Distance Search

All the results shown in Table 7, page 45 are from measuring the Hamming distance between the state being explored and the largest enlarged target. Figure 26, page 48 shows the result of measuring the Hamming distance against the varying level of Target

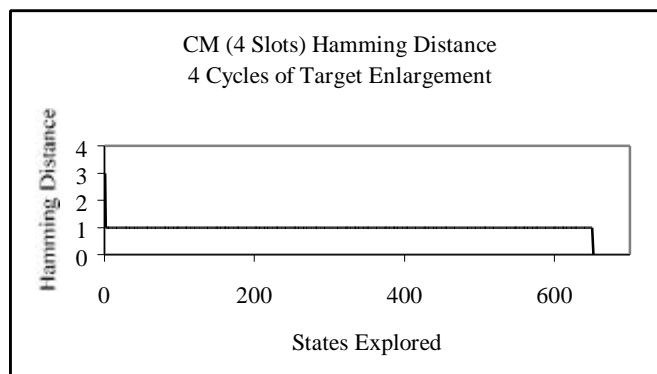
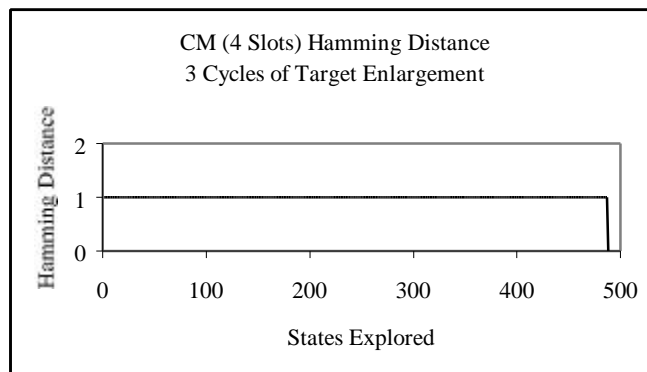
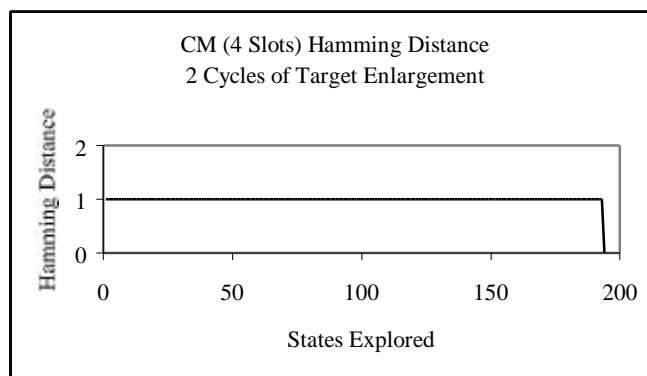
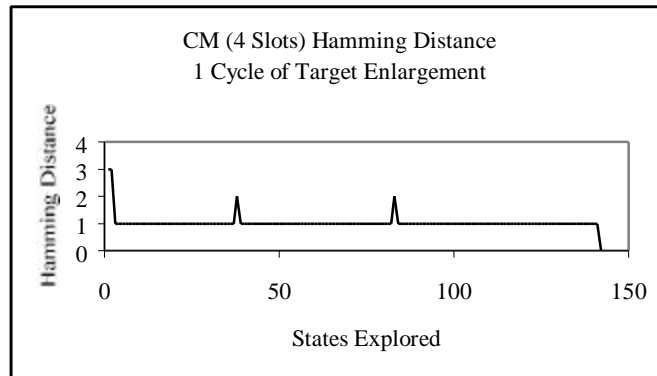
Enlargement. The result is also plotted with the result from Target Enlargement only to compare the improvement due to Hamming distance. Without any Target Enlargement (or just breadth-first search), it was not possible to measure the Hamming distance. Consequently, there is no result with Hamming distance and 0 cycles of Target Enlargement.

Figure 26. Measuring Hamming Distance Against Varying Levels of Target Enlargement



In Figure 26, each additional cycle of Target Enlargement reduces the number visited states. However, the results from Hamming distance for 2, 3, and 4 cycles of Target Enlargement are actually worse than the result from just 1 cycle of Target Enlargement. Figure 27, page 49 shows the Hamming distance during the search with various levels of Target Enlargement.

Figure 27. Hamming Distance with Various Number of Target Enlargement



While all the curves shown in Figure 27, page 49 are relatively flat, Hamming distance with only 1 cycle of Target Enlargement shows the greatest amount of discrimination. Since a larger target is more likely to have some state that has relatively close Hamming distance to the state being explored, *increasing* the target size only *decreases* the ability to discriminate among states. Consequently, each additional cycle of Target Enlargement does not allow Hamming distance to become more discriminating. Rather, the reduction in the visited states is only due to the larger size of the target (Figure 26, page 48).

4.7.1 Refinement of Hamming Distance

The results from Figure 26, page 48 and Figure 27, page 49 suggest that measuring the Hamming distance against a smaller target is likely to increase the level of discrimination among the states that can be explored. Consequently, the following refinement was made: Hamming distance is measured against the *smallest* target while the search is terminated when the *largest* target is reached. Table 8 lists the results using this refinement. These results are also plotted in Figure 28, page 51 and Figure 29, page 51.

Table 8: Hamming Distance (Small Target) Experimental Results

Design	Visited States		Explored States	
	Target Enlargement	Target Enlargement + Hamming Distance (Small Target)	Target Enlargement	Target Enlargement + Hamming Distance (Small Target)
CM (4 Slots)	197	179	83	83
CM (6 Slots)	17,650	1,092	6,437	499
CM (8 Slots)	>2,362,324	5,947	>145,498	2,735
Inbox	38,913	12,913	12,802	12,802
MC1	57,089	28,161	47,874	8,451
MC2 (Small)	119,802	94,618	113,920	88,442
MC2 (Big)	>4,799,484	>4,793,500	>2,278,144	>2,260,480
LSC	224,767	225,344	223,744	222,357

Figure 28. Hamming Distance Visited States

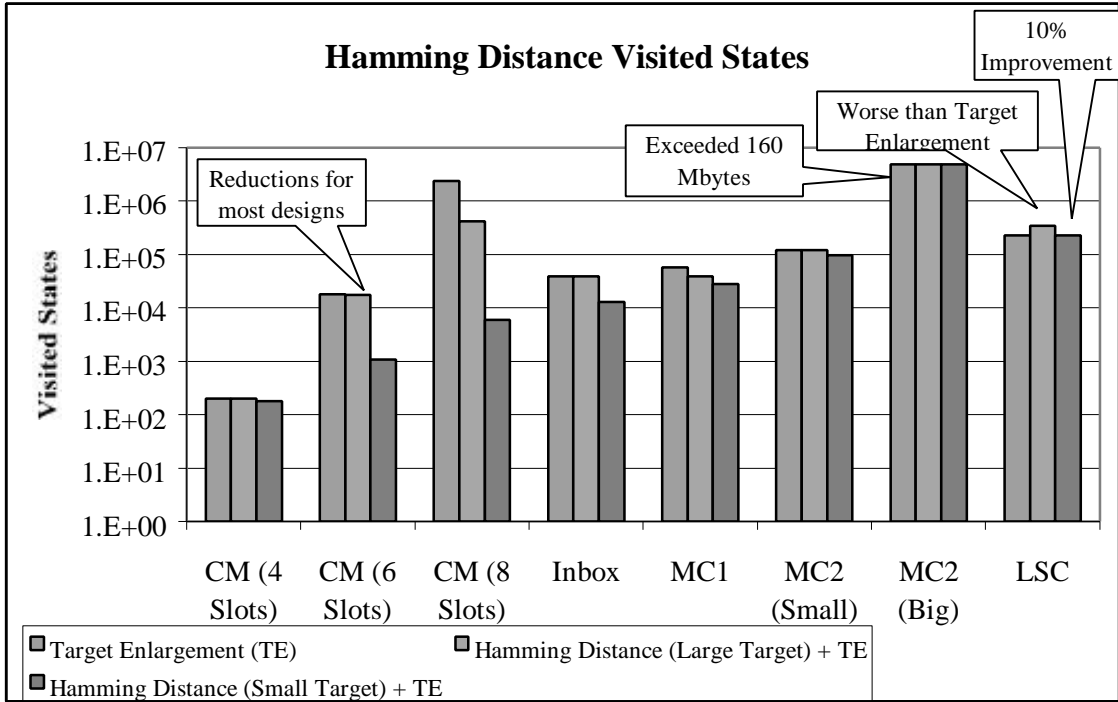
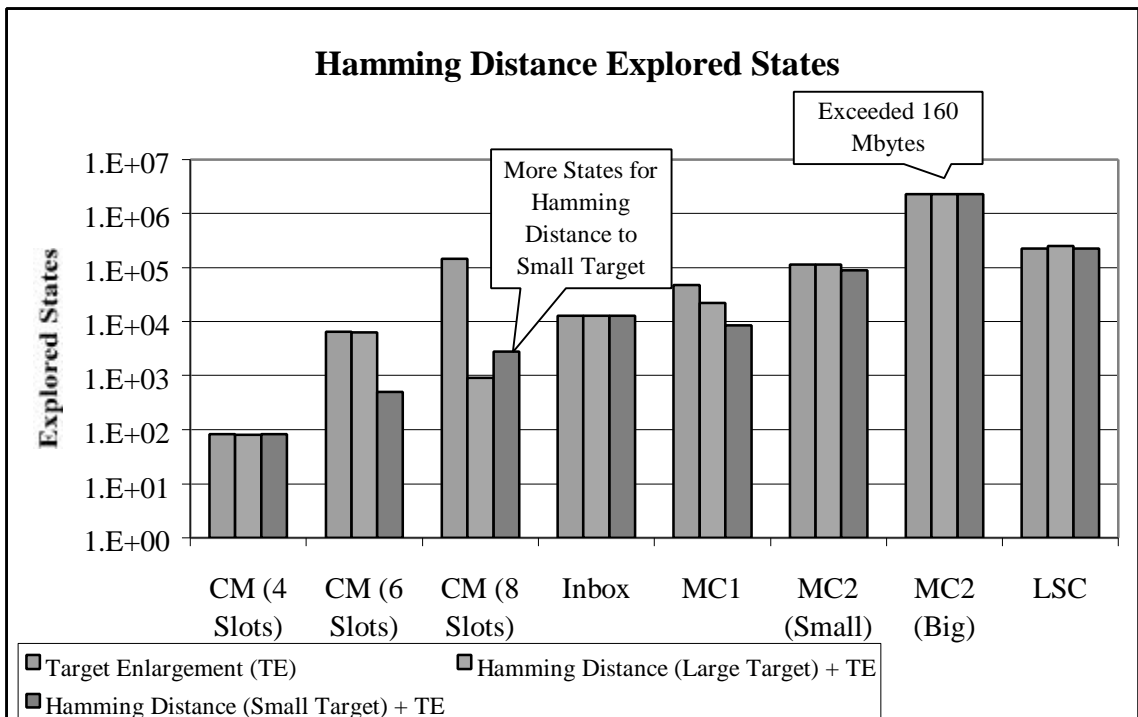


Figure 29. Hamming Distance Explored States



Measuring the Hamming distance against the smallest target does indeed produce additional reductions in the number of visited states (Figure 28, page 51). In the case of CM (8 Slots), the reduction is 3 orders of magnitude. Figure 29, page 51 also shows that the number of explored states also was reduced for most designs. However, the number of explored states for CM (8 Slots) inexplicably increased.

4.8 Conclusion

Using Hamming distance as an evaluation function demonstrated that there could be significant reduction in the number of visited and explored states in the search for bugs. This simple evaluation function confirms the intuition that breadth-first search is very inefficient in finding error states in the state space. While the reduction for some of the designs were several orders of magnitude, Hamming distance did not consistently and significantly reduce the states needed to find the bug. The inability of Hamming distance to provide a more discriminating search of the state space prevents it from being more effective.

The results presented here are more comprehensive in the number and size of designs than the study by Yuan *et al.* [31]. While their single example did show reduction in the number of explored states to reach the error, the results shown here are more comprehensive and suggest that Hamming distance can have inconsistent results. In addition, measuring the Hamming distance to the smallest target is an enhancement that can reduced the number of states even further.