A Comparison Study between the CUDD and BuDDy OBDD Package Applied to AI-Planning Problems

Rune M. Jensen

September 2002
CMU-CS-02-173

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

This report describes a comparison study between the CUDD and BuDDy OBDD package. The performance of the two packages is evaluated on three sets of AI planning problems from the AIPS-98 and AIPS-00 planning competition. Our experiments indicate that CUDD has a slight implementation overhead compared to BuDDy. However, for some problems this overhead is overcome by the fact that CUDD can perform negation in constant time.

This work was supported in part by the Danish Research Agency and the United States Air Force under Grants Nos F30602-00-2-0549 and F30602-98-2-0135. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA), the Air Force, or the US Government.

Keywords: OBDD manipulation, AI planning

1 Introduction

Currently, no comparison studies exist between the CUDD and BuDDy OBDD package [7, 4]. The object of the experiments described in this report is to provide such a study for a particular class of problems. The problems considered are classical AI planning problems. A planning problem is a graph search problem. Vertices of the graph denote world states and edges denote deterministic state transitions. The input to a planning problem is the search graph, an initial state, and a set of goal states. The output is a path leading from the initial state to one of the goal states.

OBDDs [2] are applied in the usual way to perform symbolic search [6]. Thus, in each iteration of a forward (backward) search algorithm, all next states (previous states) are computed via the image (preimage) computation. In this study, we apply forward and backward search for three planning problems: Gripper (AIPS-98), Logistics (AIPS-00 round 1), and Blocks (AIPS-00 round 1) [5, 1]. Our implementation uses standard techniques to reduce the complexity of the OBDD operations. In particular, we partition the transition relation into a disjunctive partitioning according to an upper bound of the OBDD size of each partition. This lowers the complexity of computing the transition relation and performing image and preimage computations. In addition, we apply frontier set simplification. However, in this study, Coudert, Berthet, and Madre's minimization technique [3] of the frontier set has not been applied.

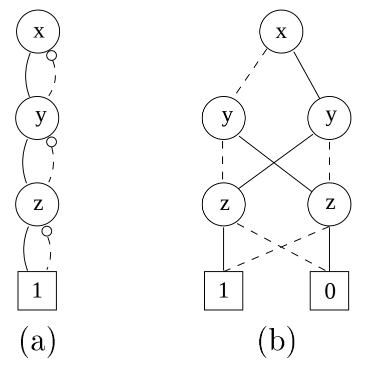

Figure 1: (a) CUDD representation. (b) BuDDy representation.

The initial cache size and the initial number of allocated slots in the unique table of the OBDD packages have been hand-tuned for best performance in each experiment. We also adjusted the upper bound of the size of the OBDDs in the disjunctive partitioning for best performance. All experiments were carried out on a Linux Redhat 7.1 PC with a 500 MHz Pentium III CPU, 512 KB L2 cache and 512 MB RAM. The time limit (TIME) was 300 seconds and the memory limit (MEM) was 300 MB.

The most significant difference between the CUDD and BuDDy package is that only CUDD applies complemented edges. A key result of the comparison experiment is therefore to determine to what extend CUDD can benefit from this representation when applied to planning problems. Our experiments indicates that CUDD has a slight implementation overhead compared to BuDDy. However, for some problems this overhead is overcome by the fact that negation can be done in constant time.

	CUDD					BuDDy				
p	T	T_r	T_s	T_e	$\|f\|$	T	T_r	T_s	T_e	$\|f\|$
2	0.17	0.05	0.04	0.03	103	0.17	0.05	0.03	0.01	103
4	0.36	0.12	0.17	0.08	253	0.30	0.10	0.12	0.03	253
6	0.61	0.22	0.31	0.19	470	0.40	0.17	0.13	0.06	470
8	1.41	0.39	0.64	0.34	755	1.25	0.28	0.32	0.11	755
10	1.76	0.56	1.09	0.65	1108	1.55	0.39	0.48	0.22	1108
12	2.50	0.83	1.52	0.87	1528	1.98	0.55	0.75	0.35	1528
14	3.57	1.28	2.11	1.16	2016	2.45	0.73	1.03	0.53	2016
16	5.34	1.76	3.43	2.01	2571	3.15	0.95	1.53	0.77	2571
18	7.20	2.37	4.68	2.68	3194	3.99	1.2	2.09	1.10	3194
20	9.53	3.12	6.22	3.52	3884	4.90	1.5	2.67	1.50	3884

Table 1: Gripper results for forward search.

	CUDD					BuDDy				
p	T	T_r	T_s	T_e	$\|f\|$	T	T_r	T_s	T_e	$\|f\|$
2	0.15	0.05	0.03	0.01	143	0.18	0.05	0.04	0.00	143
4	0.32	0.12	0.13	0.02	265	0.28	0.10	0.09	0.01	365
6	0.62	0.22	0.32	0.04	686	0.55	0.19	0.26	0.02	686
8	1.06	0.38	0.56	0.07	1107	1.53	0.30	0.55	0.05	1107
10	1.70	0.61	0.96	0.10	1627	2.02	0.42	0.89	0.08	1627
12	2.62	0.88	1.57	0.26	2245	2.68	0.59	1.37	0.12	2245
14	3.82	1.35	2.26	0.21	2964	3.47	0.78	1.94	0.16	2964
16	5.27	1.87	3.18	0.27	3782	4.64	1.01	2.88	0.23	3782
18	7.95	2.51	5.25	0.35	4698	5.95	1.27	3.94	0.31	4698
20	9.97	3.13	6.60	0.42	5714	8.57	1.60	6.19	0.39	5715

Table 2: Gripper results for backward search.

2 Experiments

For each problem, we perform experiments for forward and backward search. We conduct a timing study that measures the total CPU time T, the time used to produce the transition relation T_r, the search time T_s, the solution extraction time T_e, and the average size of the OBDDs representing the search frontier $|f|$. All time measures are in seconds.

It should be noted that CUDD and BuDDy measure the size of an OBDD slightly different. For BuDDy, the size equals the number of internal nodes of the OBDD, while it for CUDD equals the sum of the number of internal and terminal nodes of the OBDD. Figure 1 shows the CUDD and BuDDy representation of an identical Boolean function. The CUDD size is 4, while it for BuDDy is 5.

2.1 Gripper

This problem scales well when using an OBDD approach. Solution paths are long, but the frontier OBDDs only grow moderately. The results are shown in Table 1 and Table 2.

2.2 Logistics

The logistics domain is hard for OBDD-based blind search due to a high growth rate of frontier OBDDs. The results are shown in Table 3 and Table 4.

	CUDD					BuDDy				
p	T	T_r	T_s	T_e	$\|f\|$	T	T_r	T_s	T_e	$\|f\|$
4	0.41	0.09	0.21	0.08	656	0.34	0.09	0.15	0.02	656
5	0.56	0.10	0.35	0.11	720	0.43	0.08	0.24	0.03	720
6	0.47	0.10	0.28	0.09	697	0.44	0.09	0.25	0.02	697
7	50.44	0.39	48.04	0.43	23910	73.11	0.37	69.84	0.12	23911
8	34.73	0.39	32.78	0.35	19767	42.60	0.35	39.36	0.11	19767
9	49.2	0.39	46.86	0.44	24864	71.24	0.35	68.00	0.11	24854
10	TIME					TIME				

Table 3: Logistics results for forward search.

	CUDD					BuDDy				
p	T	T_r	T_s	T_e	$\|f\|$	T	T_r	T_s	T_e	$\|f\|$
4	0.36	0.09	0.14	0.03	300	0.42	0.09	0.19	0.02	300
5	0.55	0.09	0.33	0.03	571	1.11	0.09	0.45	0.03	571
6	0.75	0.09	0.49	0.03	1314	1.18	0.09	0.55	0.02	1315
7	179.39	0.39	177.28	0.15	46906	272.20	0.35	267.93	0.06	46906
8	TIME					TIME				

Table 4: Logistics results for backward search.

2.3 Blocks

Similar to the Logisitics domain, frontier OBDDs in the Blocks domain tend to grow fast. The results are shown in Table 5 and Table 6.

3 Conclusion

The most striking result of these experiments is that the size of the OBDDs with complemented edges used by CUDD is almost identical with the size of the ordinary OBDDs used

p	CUDD					BuDDy				
	T	T_r	T_s	T_e	$\|f\|$	T	T_r	T_s	T_e	$\|f\|$
4	0.17	0.04	0.01	0.00	103	0.18	0.04	0.01	0.01	102
5	0.26	0.07	0.08	0.02	281	0.25	0.07	0.04	0.00	280
6	0.45	0.12	0.22	0.04	859	0.40	0.12	0.15	0.01	858
7	2.74	0.25	2.26	0.11	4042	2.89	0.23	2.12	0.04	4040
8	27.93	0.37	26.47	0.17	30436	24.01	0.34	20.82	0.05	30435
9	258.52	0.55	256.01	0.63	97135	220.38	0.49	216.01	0.11	97134
10	TIME					TIME				

Table 5: Blocks results for forward search.

p	CUDD					BuDDy				
	T	T_r	T_s	T_e	$\|f\|$	T	T_r	T_s	T_e	$\|f\|$
4	0.15	0.04	0.02	0.00	177	0.17	0.04	0.01	0.00	176
5	1.02	0.07	0.81	0.02	2342	1.53	0.08	0.91	0.01	2341
6	9.64	0.12	9.03	0.04	19159	11.64	0.14	8.66	0.00	19158
7	Time					Time				

Table 6: Blocks results for backward search.

by BuDDy[1]. However, complemented edges makes it possible for CUDD to perform negation in constant time. For search problems, an efficient negation operation may improve the complexity of pruning the search frontier from previously reached states, since a potentially large OBDD representing the previously reached states must be negated and conjoined with the OBDD representing the frontier. This seems to be an advantage for some problems. For instance, in the Logistics domain, CUDD is about twice as fast as BuDDy on the harder problems.

However, in the Blocks domain, BuDDy is slightly faster than CUDD even for hard problems. Together with the results in the Gripper domain, this seems to indicate that CUDD is implemented with a slight overhead compared to BuDDy, but that efficient negation may overcome this overhead for certain problems.

References

[1] F. Bacchus. AIPS'00 planning competition : The fifth international conference on artificial intelligence planning and scheduling systems. *AI Magazine*, 22(3):47–56, 2001.

[2] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 8:677–691, 1986.

[3] O. Coudert, C. Berthet, and J. Madre. Verification of sequential machines using symbolic execution. *Automatic Verification Methods for Finite State Machines*, pages 365–373, 1989.

[1]When inspecting the frontier OBDDs at most a size difference of one is observed

[4] J. Lind-Nielsen. BuDDy - A Binary Decision Diagram Package. Technical Report IT-TR: 1999-028, Institute of Information Technology, Technical University of Denmark, 1999. `http://cs.it.dtu.dk/buddy`.

[5] D. Long. The AIPS-98 planning competition. *AI Magazine*, 21(2):13–34, 2000.

[6] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publ., 1993.

[7] F. Somenzi. CUDD: Colorado university decision diagram package. `ftp://vlsi.colorado.edu/pub/`, 1996.