Empirical Evaluation of AND/OR Multivalued Decision Diagrams for Inference

Student: William Lam, Advisor: Rina Dechter

Donald Bren School of Information and Computer Sciences
University of California, Irvine, CA 92697, USA
{willmlam,dechter}@ics.uci.edu

Abstract. AND/OR Multi-valued Decision Diagrams (AOMDD) were shown to provide a more compact representation of discrete-domain real-valued functions compared to other decision diagram variants [1]. We show the performance of AOMDDs on inference tasks in graphical models. We introduce the elimination operator to AOMDDs, which in conjunction with the combination operator introduced in previous work, yields a full bucket elimination (BE) scheme using AOMDDs as an alternative function representation to tables. We show that we are able to solve instances that do not fit in main memory when using tables.

1 Introduction

AND/OR Multi-valued Decision Diagrams (AOMDDs) combine the two frameworks of AND/OR search spaces and multi-valued decision diagrams (MDDs) to create a framework that compactly represents discrete-domain functions such as those in discrete graphical models [1]. The AND/OR search space is a more compact search space for search-based inference algorithms in graphical models compared to OR search spaces. For problems with decomposition into subproblems, the AND/OR search space captures this. Decision diagrams are generally used to represent functions compactly [2].

The key algorithm for combining AOMDDs, *apply*, first introduced in [3] was never implemented before. Our work also extends upon previous work by introducing the elimination operator to AOMDDs. With these two algorithms in place, this yields the full bucket elimination [4] scheme using AOMDDs as an alternative function representation to tables. We provide the first empirical results demonstrating the algorithm and contrasting its performance with the BE algorithm using tables.

Similar work is presented in [5, 6], where an algebraic decision diagram (ADD) structure is considered. In [6], ADDs are extended with affine transformations to capture additive and multiplicative structures in graphical models. However, AND structure is still not exploited in these alternative decision diagram variants and they are restricted to variables with binary domains.

We start with presenting preliminaries by defining graphical models and counting queries.

Definition 1 ***(graphical model/counting query)*** *A graphical model is a tuple $\mathcal{R} = \langle X, D, F, \otimes \rangle$, where $X = \{X_1, ..., X_n\}$ is a set of variables, $D = \{D_1, ..., D_n\}$ is the set of the respective finite domains of the variables in X, $F = \{f_1, ..., f_r\}$ is a set of real-valued functions defined over a subset of variables $S_i \subseteq X$, and $\otimes$ is a combination operator (i.e. $\prod, \sum, \bowtie$). The graphical model represents a global function computed by $\otimes_{i=1}^{r} f_i$. For a CSP, the number of solutions is the number of assignments which do not violate any constraints. For a weighted CSP, the weighted solution count is the sum of the weights of all solutions such that no constraint is violated (having a weight of 0). For graphical models representing probability distributions, this is likelihood/ partition function computation. Formally the task is to find $\sum \prod_{i=1}^{r} f_i$*

We refer the reader to previous work for background on AND/OR search spaces and decision diagrams [1, 2]. The basic idea of AOMDDs is augmenting context-minimal AND/OR search graphs to remove redundant nodes (or equivalently, augmenting weighted MDDs with AND nodes.) Overall, AOMDDs exploit determinism and context-specific independence [7] to achieve compactness. More details on AOMDDs are in [1].

2 Algorithms

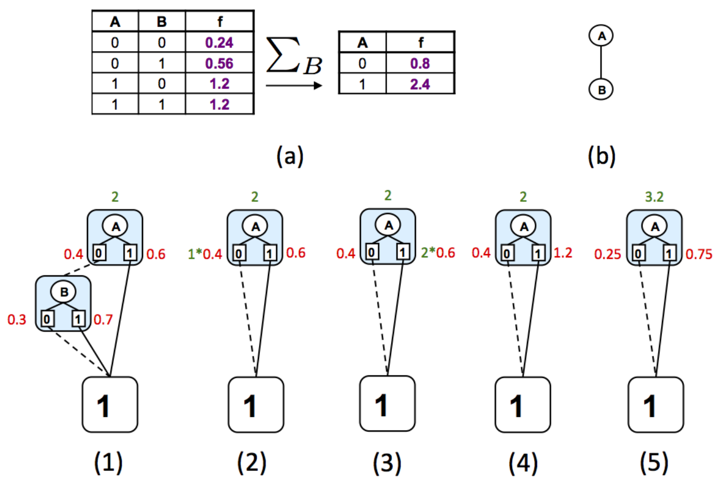
In this work, we include the reduction rule by redundancy and use AOMDDs as an alternative to a tabular representation of the functions and messages in bucket elimination. To perform this, we require a method of applying the combination operator to AOMDDs and the elimination operators. For AOMDDs, it is presented here for the first time.

The main operation to perform the combination of two AOMDDs is the *apply* operator. It is stated that the runtime of apply is quadratic in the size of the input AOMDDs. We omit the full details of the algorithm for space issues and refer the reader to previous work [1].

There are difference and restrictions of the operation when compared to decision diagrams without AND decomposition. Since AOMDDs further compress a function representation by taking advantage of decomposition in the pseudo tree [1], operations on it are bound by the same rules as variable elimination. Namely, once we consider a fixed variable ordering, eliminating a variable whose children are not yet eliminated would induce edges in the induced graph between all of its neighbors. Equivalently, this means we would be changing the order of sum and product operators.

A basic description of the algorithm is as follows. From the embedded pseudo tree of the AOMDD, we create a list of *relevant* variables by tracing a path from the leaf node representing the elimination variable to the root of the tree. This creates a direct path from the root of the AOMDD down to the elimination variable without the need to explore other branches of the AOMDD. We then create a reverse BFS ordering based on list of relevant variables. If the node is an elimination variable, we eliminate the node by performing the necessary operator and promote the weight to the parent. Otherwise, we normalize the node (making its AND children weights sum to 1) and pass on the normalization constant to the parent.

One caveat to note is that the metanode for a variable we are eliminating may not be present in the decision diagram due to the reduction rules. This is an issue when the elimination operator is summation. We must compensate for any missed metanodes, which we can identify if we see an ancestor of the elimination variable connected to a terminal metanode. Since nodes would be missing only if it were redundant, we can multiply the weight of that ancestor metanode by the domain size of the elimination variable. However, in the process after eliminating a node, the intermediate structure looks identical to that of when the input diagram does not have the node due to redundancy reduction. Therefore, we keep track of which nodes already received a weight from a child node to distinguish between these two.



A	B	f
0	0	0.24
0	1	0.56
1	0	1.2
1	1	1.2

$\xrightarrow{\sum_B}$

A	f
0	0.8
1	2.4

Fig. 1. Example of elimination on AOMDDs. The state of the AOMDD is shown through the process.

We demonstrate the algorithm on a small example, shown in Figure 2. The function tables above (a) the AOMDDs demonstrate the operation performed in a standard representation. We are interested in summing out variable B. The embedded pseudo tree (b) is used to determine the set of relevant variables, which in this case is $\{A, B\}$.

We begin with the AOMDD shown at (1), which represents the same function as the input table. Visiting the relevant nodes in a reverse BFS order, we visit the metanode B first, eliminate it, and propagate its result up to the parent AND node in metanode A, shown in (2). At (3), we are left with only metanode A. Checking the 0 AND node, since it received a weight from something, we are done with it. Checking the 1 AND node, since it has not received a weight, we multiply it by the domain size of B, which is 2 in this case. The result is now shown at (4). Finally, we normalize the AND node weights of metanode A and propagate its normalization term up to the root, yielding the resulting AOMDD in (5), which represents the same function as the output table.

In the cases of maximization and minimization, we do not encounter the same problem since these operators choose one from the set of values, which has no effect on functions where all the output values are identical. Conditioning can also be considered a form of elimination and also does not suffer from the issues that summation encounters for the same reasons.

With an elimination operator, this yields a full BE algorithm for inference using AOMDDs as a function representation (AOMDD-BE). The complexity remains the same as standard BE, as in the worst case, the AOMDD has as many AND nodes as the number of entries in the table. However, for some problem structures, the AOMDD size can be far smaller than the table size.

3 Experiments

For all tables in this section, for each problem instance, we report number of variables (n), induced width (w), height of the pseudo tree (h), maximum domain size (k), time, and memory usage. The algorithms were implemented in C++ (64-bit) and the experiments were run on 2.6 GHz machines with 24GB of RAM.

The following evaluates the AOMDD-BE algorithm, which is the same as bucket elimination, but uses AOMDDs to represent all functions. We ran experiments on the UAI 2006 evaluation problems and genetic linkage analysis networks, available at `http://graphmod.ics.uci.edu`. In each table, we compare the time and memory usages of standard BE vs. AOMDD-BE. Times reported as "OOM" indicate that the algorithm exceeded our memory bound. Results on memory usage are based on the usage of the cache storing nodes of the AOMDDs. For instances where BE runs out of memory, we simulated its execution by only passing information about scope sizes to compute the memory usage.

UAI 2006 benchmarks. Results are presented in Table 1. In columns 5 and 6, we see the runtimes for BE and AOMDD-BE, while the last two columns show the the memory usages of BE and AOMDD-BE.

We see that our scheme is able to solve some problems which do not fit in standard main memory. These problems have structures that AOMDDs exploit well. Namely, the functions of these problems have many zero values that can be represented easily by AOMDDs. In addition, AOMDDs are able to take advantage of functions that have many values that are the same, but not necessarily zero. Such functions are present in a number of the instances on which it outperforms BE based on memory usage. However, there must be a significant amount of compression before we get any memory savings. Namely, as each node contains information to capture the structure of the problem, it means that much more memory is used when representing a function which has many different values.

problem	n	w	h	k	time (s) [BE]	time (s) [AOMDD-BE]	Mem (MB) [BE]	Mem (MB) [AOMDD-BE]
BN_22	2425	5	575	91	1	13	**26.93**	581.27
BN_28	24	5	9	10	1	13	**1.79**	568.36
BN_30	1156	48	179	2	OOM	38	1.50E+10	**245.93**
BN_32	1444	56	219	2	OOM	4384	4.45E+12	**3006.08**
BN_34	1444	55	220	2	OOM	145	2.30E+12	**515.45**
BN_40	1444	55	235	2	OOM	91	1.82E+12	**322.76**
BN_42	880	23	54	2	21	2	314.04	**21.62**
BN_46	499	22	49	2	18	<1	248.97	**1.99**
BN_49	661	44	59	2	OOM	1188	7.83E+08	**2991.78**
BN_53	561	48	95	2	OOM	4063	8.43E+09	**3303.48**
BN_61	667	44	61	2	OOM	17	9.46E+08	**235.72**
BN_65	440	61	95	2	OOM	1062	Overflow*	**2843.65**
BN_84	360	20	24	2	4	22	**24.76**	546.21
BN_92	422	22	33	2	26	23	**187.43**	433.65

name	n	w	h	k	time (s) [BE]	time (s) [AOMDD-BE]	Mem (MB) [BE]	Mem (MB) [AOMDD-BE]
pedigree1	334	15	61	4	2	14	**23.61**	210.09
pedigree9	1118	25	137	7	550	5301	7499.77	**4030.34**
pedigree18	1184	19	102	5	7	200	**136.13**	959.28
pedigree20	437	21	58	5	131	291	1393.90	**1030.66**
pedigree23	402	20	58	5	19	52	**241.57**	532.46
pedigree25	1289	23	86	5	146	1284	**2037.69**	2999.84
pedigree30	1289	20	102	5	13	307	**220.63**	1044.76
pedigree33	798	24	116	4	347	883	4277.26	**1368.42**
pedigree37	1032	20	62	5	OOM	3535	251109.68	**7992.43**
pedigree38	724	16	67	5	OOM	2201	172249.65	**6253.16**
pedigree39	1272	20	83	5	46	400	**772.20**	1555.68
pedigree44	811	24	79	4	516	3795	6153.63	**4782.29**

Table 1. UAI 2006 benchmarks and pedigree networks. For pedigree networks, instances not shown here (7,13,19,31,34,40,41,42,50,51) run out of memory with both algorithms. (* The size in MB could not be stored within a double precision number representation.)

We generally see that for lower treewidth networks, standard BE is sufficient and has better runtime, however, it is unable to solve problems with higher treewidth due to lack of memory.

Pedigree networks. We also ran experiments on genetic linkage analysis networks (known as pedigree), for which the partition function value of many of them were not known before the work in [8], which makes use of hard disk to push the memory restrictions of solving a problem.

The results are shown in Table 1. As with the previous set or problem instances, timing results are shown in columns 5 and 6 while memory usage is shown in columns 7 and 8.

Our results are less promising on these networks. There are only two instances which AOMDD-BE manages to perform very well on, which standard BE would require about 30 times the amount of memory. For the rest that AOMDD-BE managed to solve, a large number of problems were solvable by standard BE with a shorter amount of time and less memory. Even with those where AOMDD-BE uses less memory, the runtime is often much worse, due to overhead in maintaining the properties of a canonical AOMDD. We can attribute these results this set of problems having overall less determinism and context-specific independence. However, these results also demonstrate the use of decision diagrams on non-binary networks for inference, when compared to related work using ADDs [5, 9].

4 Conclusion

For many hard problems (such as the pedigree networks), the overhead of using the minimal AOMDD structure for function representation actually results in worse performance in terms of both time and space. On other problems, such as the ISCAS networks in the UAI 2006 evaluation set, our scheme shows good performance despite having high treewidth. We demonstrated results reinforcing the potential of using AOMDDs for the classic BE algorithm. Future work would include comparing with related techniques exploiting determinism and context-specific independence such as ACE [10].

References

1. Mateescu, R., Dechter, R., Marinescu, R.: AND/OR multi-valued decision diagrams (AOMDDs) for graphical models. Journal of Artificial Intelligence Research **33**(1) (2008) 465–519
2. Bryant, R.: Graph-based algorithms for boolean function manipulation. IEEE Trans. Computers **35**(8) (1986)
3. Mateescu, R., Dechter, R.: Compiling constraint networks into AND/OR multi-valued decision diagrams (AOMDDs). In: Principles and Practice of Constraint Programming (CP 2006). (2006) 10.1007/11889205_25.
4. Dechter, R.: Bucket elimination: A unifying framework for reasoning. Artificial Intelligence **113**(1) (1999) 41–85
5. Chavira, M., Darwiche, A.: Compiling bayesian networks using variable elimination. In: Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07). (2007) 2443–2449
6. Sanner, S., McAllester, D.: Affine algebraic decision diagrams (AADDs) and their application to structured probabilistic inference. In: Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05). Volume 19. (2005) 1384
7. Boutilier, C., Friedman, N., Goldszmidt, M., Koller, D.: Context-specific independence in bayesian networks. In: Proc. of the 12th International Conference on Uncertainty in Artificial Intelligence (UAI-96). (1996) 115–123
8. Kask, K., Dechter, R., Gelfand, A.: BEEM: Bucket elimination with external memory. In: Proc. of the 26th Annual Conference on Uncertainty in Artificial Intelligence (UAI-10). (2010) 268–276
9. Gogate, V., Domingos, P.: Approximation by quantization. In: Proc. of the 27th Annual Conference on Uncertainty in Artificial Intelligence (UAI-11). (2011) 247–255
10. Chavira, M., Darwiche, A.: Compiling bayesian networks with local structure. In: Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05). Volume 19. (2005) 1306