An Overview of Compiling Constraint Networks into AND/OR Multi-valued Decision Diagrams

Project Report for CS 275

William Lam

willmlam@ics.uci.edu

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 compiles a given graphical model into a smaller representation. AND/OR search spaces are an improved framework for search based inference algorithms for graphical models in general. The main idea is that it uses the underlying structure of the graphical model it provide a more compact search space due to possible independencies between variables in the model. Decision diagrams in general are used to represent functions as a directed acyclic graph, which can offer a very compact representation of these functions [4].

A motivation of compiling constraint networks into AOMDDs is that algorithms for performing queries on decision digrams are very fast [1]. We can compile a constraint network into an AOMDD once and perform quick online queries.

This report summarizes the main ideas in [4] and provides some empricial results to extend its work. There are more recent results presented in [5, 3], but these are for weighted CSPs and Bayesian networks. Although constraint networks can be viewed as a special case of these two, experiments on them can still be useful to see how AOMDDs function on unweighted networks. Additionally, CNF formulas are also tested in this project.

Background

We start with presenting the two frameworks, how each applies to constraint networks, and how they are combined to summarize the work of [4]. This section provides an extended description of MDDs and also presents the later notion of *semantic treewidth* introduced in [5].

AND/OR Search

AND/OR search spaces allow us to use the underlying structure of the network to save on work over the standard search techniques. AND nodes represent the decomposition of the problem into independent subproblems, which allows us to remove many nodes in the standard search space which only has OR nodes representing only decision points. We first need to find a *pseudo tree* of the constraint graph to define the structure of the *AND/OR search tree* for a constraint network. A pseudo tree $\mathcal{T}$ of a graph is a rooted tree with the same nodes and that each arc in the graph is a backarc in $\mathcal{T}$. An AND/OR search tree then uses the structure of $\mathcal{T}$ as follows.

- An OR node corresponds to variables in the network, and are labeled X_i.
- Each OR node has child AND nodes labeled $\langle X_i, x_i \rangle$, corresponding to the values X_i can take on.
- Each AND node has child OR nodes corresponding to the children of variable X_i in $\mathcal{T}$.

Some nodes may actually correspond to identical subproblems, and we can therefore *unify* them to achieve a *context minimal AND/OR search graph.* We can do this by identifying contexts of a variable, which are the variables that are ancestors of the variable in $\mathcal{T}$ and are connected in the constraint graph. Nodes are unifiable if all of the variables in their context are assigned the same values since this suggests that they become independent based on that conditioning on variables it depends on.

Decision Diagrams

A *decision diagram* is a compilation of a function. Let us consider the most basic version, a *binary decision diagram (BDD).* A BDD represents a function $f : \mathbb{B}^N \Rightarrow \mathbb{B}$ via a directed acyclic graph. The graph has two terminal nodes 0 and 1 that represent the right-hand side of the mapping. Each internal node then corresponds to one of the N boolean variables on the left-hand side of the mapping and has two child pointers representing the values it can take on. If the internal node represents variable i, then its children point to some other internal node corresponding to a variable $i < j \leq N$, where j is not necessarily the same for both children, or a terminal node. As suggested by the rule for children, the variables are in some ordering. This restriction allows us to merge isomorphic nodes in the BDD to achieve a more compact graph. Furthermore, we eliminate redundant nodes, which are nodes whose child arcs point the the same node. By enforcing maximum merging and removing redundancies, we achieve a canonical BDD representing f, known as a *reduced ordered binary decision diagram (OBDD)* [1].

A *multi-valued decision diagram (MDD)* is a generalization of a BDD. Here, a function $f : \mathbb{X}_1 \times ... \times \mathbb{X}_N \Rightarrow \mathbb{B}$ is represented instead, where each $\mathbb{X}_i$ is some finite domain of values. Instead, each internal node corresponding to variable i has $|\mathbb{X}_i|$ children representing the values it can take on. The other rules are the same.

It is clear that MDDs allow us to encode sets over a domain $(\mathbb{X}_1 \times ... \times \mathbb{X}_N)$. Elements that are in the set are represented by following a path in the decision diagram that leads to the terminal node labeled as 1. Otherwise, the path leads to the 0 terminal node.

There is also a notion of combining MDDs via an *apply* operator. Given that the variables in both MDDs have the same ordering, we can combine them via a recursive process that traverses both graphs down to their terminals. Once here, the terminals are then compared and the appropriate terminal node of the new BDD is then returned. For example, if we wish to compute the result of unioning the two graphs, we would evaluate the terminal values through a logical OR function and return the appropriate terminal. The recursion then pops back and builds the structure of the decision diagram. More details of this algorithm are in [1].

Since a constraint network can be expressed as the set of its solutions, we can represent a constraint network with a MDD that represents that set. We can also take advantage of the canonicity of a MDD to decide whether two constraint networks are identical, given that both networks are compiled into a MDD using the same variable ordering. Since combining constraints are performed by the join operator, our base case at the terminal nodes in the apply routine would be to multiply the two terminal values [4].

Putting them Together: AOMDD

The context minimal AND/OR search graph does not capture the case of redundant nodes and a MDD is an OR graph. We can therefore combine the two frameworks to get all of their benefits. This yields the *AND/OR Multi-valued Decision Diagram.* The basic idea is that the child pointers are instead AND nodes that follow the same idea as those in AND/OR search space. Each OR node and its AND nodes are viewed as a *meta-node*, which is what we count to determine the size of a given AOMDD.

To compile a constraint network into an AOMDD, there are two methods. The first, presented in [4] is similar to Bucket Elimination [2]. The buckets are organized with the same structure as the pseudo tree. We create an AOMDD for each constraint in the network, placing them into the deepest bucket possible, depending on the constraint of the scope. We then perform the *apply* operation to join the AOMDDs in each bucket, passing this as a message up the tree to another bucket. The compiled AOMDD is finally complete after joining all of the messages and constraints in the root of the tree. Figure 2 illustrates the algorithm on the network shown in Figure 1.

Since AOMDDs are based on the more specific pseudo-tree (as opposed to a variable ordering in MDDs), [4] also defines a new *apply* operator that combines two AOMDDs based on whether their pseudo trees are compatible. Namely, this means that there is some $\mathcal{T}$ which we can embed both pseudo trees in. Figure 3 in the Appendix shows the pseudocode.

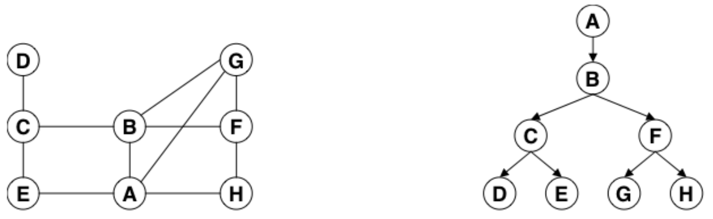


Figure 1: Example constraint network and pseudo tree. The constraints are $C_1 = F \vee H, C_2 = A \vee \neg H, C_3 = A \otimes B \otimes C, C_4 = F \vee G, C_5 = B \vee F, C_6 = A \vee E, C_7 = C \vee E, C_8 = C \otimes D, C_9 = B \vee C$. The ordering for the pseudo tree is $d = (A, B, C, D, E, F, G, H)$. (Taken from [4].)

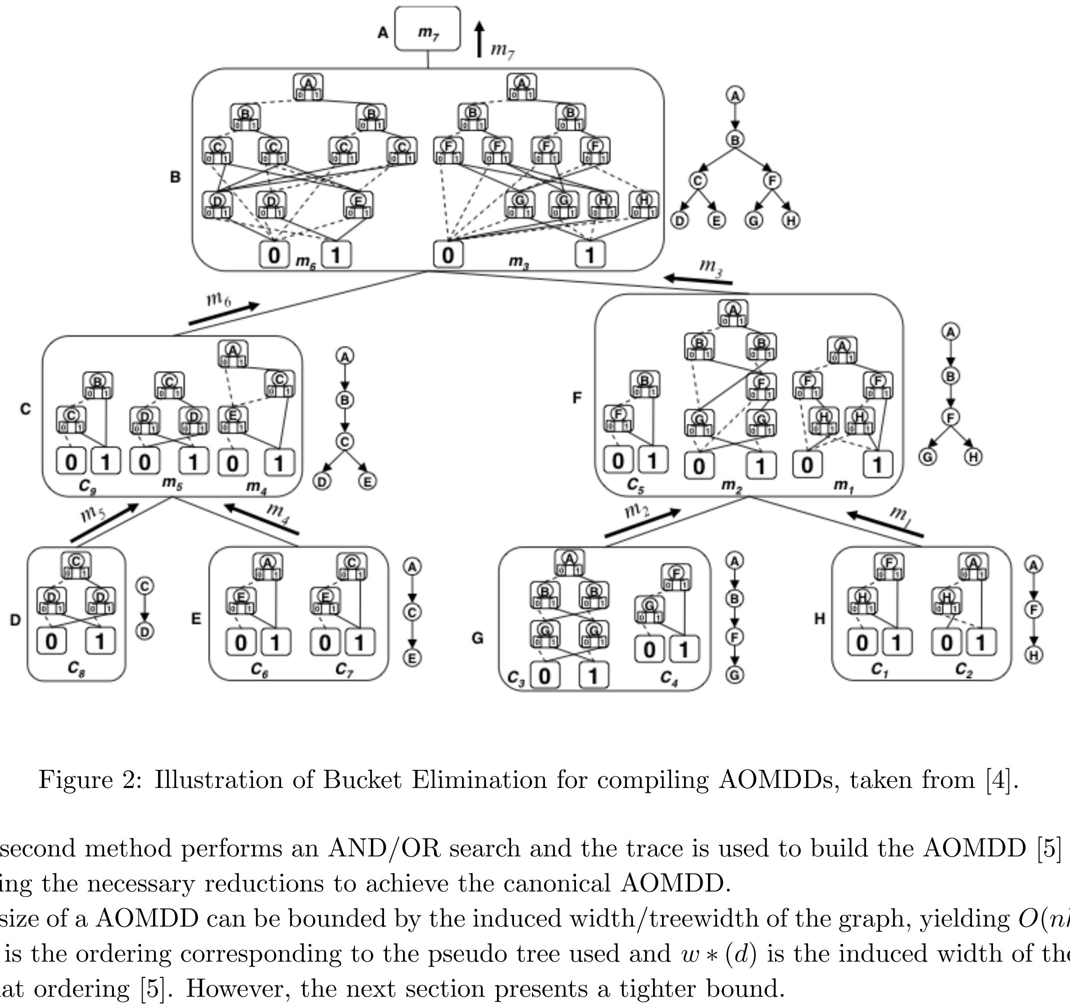


Figure 2: Illustration of Bucket Elimination for compiling AOMDDs, taken from [4].

The second method performs an AND/OR search and the trace is used to build the AOMDD [5] by also performing the necessary reductions to achieve the canonical AOMDD.

The size of a AOMDD can be bounded by the induced width/treewidth of the graph, yielding $O(nk^{w*(d)})$, where d is the ordering corresponding to the pseudo tree used and $w*(d)$ is the induced width of the graph along that ordering [5]. However, the next section presents a tighter bound.

Semantic Treewidth

We finish this section by discussing the notion of *semantic treewidth*. The idea is that for a given network, there exists many, possibly sparser, equivalent networks, that is, their set of solutions are the same. Therefore, the complexity of the problem may not necessarily be bounded by the induced width of a given network. With slight modifications to the definition in [5] to restrict our attention to constraint networks, the semantic treewidth relative to a pseudo tree $\mathcal{T}$ of a constraint network R, denoted as $sw_{\mathcal{T}}(R)$ is the smallest treewidth taken over all equivalent networks $\mathcal{R}$, and $\mathcal{T}$ is a pseudo tree for all of these networks. The semantic treewidth of a constraint network R is the smallest semantic treewidth over all pseudo trees that can express the set of solutions. Instead of using treewidth to bound the size of an AOMDD relative to a pseudo tree $\mathcal{T}$, we can use the semantic treewidth instead, yielding $O(nk^{sw_{\mathcal{T}(R)}})$.

Experiments

The source code for compiling AOMDDs from weighted CSPs was provided by Radu Marinescu. All constraint networks were represented by weighted CSPs with an upper bound of 1, where the task is minimization. Inconsistent tuples in constraints are given a weight of 1, while consistent tuples are given a weight of 0. This encodes a constraint network as a weighted CSP that the code compiles into an AOMDD.

The method used to compile is the top-down approach via AND/OR Branch-and-Bound search. (Unfortunately, the Bucket Elimination compilation scheme that is presented in [4] has not been implemented.[1]) For all of the experiments, a static mini-bucket heuristic with an i-bound of 4 was used for AND/OR Branch-and-Bound. Psuedo-trees were found using a Min-Fill ordering.

The notation used in the tables is as follows:

- n - Number of variables in the network
- c - Number of constraints in the network
- w* - Induced width (treewidth) of the graph
- h - Height of the pseudo tree
- |d| - Maximum domain size in the network

We encoded the Langford(2,k) problem[2] as a constraint network and as a SAT formula. The size of the standard OR search space was computed by hand, since we know that the size of this space is $|\mathbb{X}_1 \times ... \times \mathbb{X}_N|$. The results are presented in Tables 1 and 2. We only count OR nodes in the context-minimal AND/OR search space since AOMDD size is measured by the number of metanodes. Therefore, this allows us to see the any further merges of nodes performed by compiling to an AOMDD. The amount of savings we get over the standard OR search space is very clear. Although the results lack a comparison for using the MDD framework only, many merges done by the context-minimal AND/OR search space framework would have also been done by the MDD framework.

We also tested on instances of Uniform Random 3-SAT benchmarks[3]. Each instance has 20 variables with around 80-90 clauses. All of them were also satisfiable. Table 3 presents the results. The ratio of the context-minimal AND/OR search space to AOMDD is more apparent here. A SAT formula can sometimes be satisfied given assignments only to a subset of the variables. Therefore, any unassigned variables have no bearing on the result. In a decision diagram, these are the redundant nodes that are removed.

Tests were also attempted on a set of instances of 50 variables with around 218 clauses, but these exceeded a 30 minute time limit. Before timing out the decision diagram size reached an order of 10^6 with a cm/aomdd ratio of about 8, which may suggest that the ratio gets better as the problem increases.

[1] Personal communication, Rina Dechter.
[2] `http://www.cs.st-andrews.ac.uk/~ianm/CSPLib/prob/prob024/index.html`
[3] http://www.cs.ubc.ca/ hoos/SATLIB/benchm.html

langford(2,k)	n	w*					
	c	h	|d|	#OR	time	#cm(OR)	#aomdd
	3	2		24	0.0010	5	5
k=3	4	2	4			ratio(cm/aomdd)=	**1.00**
						ratio(or/aomdd)=	**4.80**
	5	4		6720	0.0640	278	236
k=5	10	4	8			ratio(cm/aomdd)=	**1.18**
						ratio(or/aomdd)=	**28.47**
	7	6		3991680	0.3380	1431	1005
k=7	21	6	12			ratio(cm/aomdd)=	**1.42**
						ratio(or/aomdd)=	**3971.82**
k=9	9	8	16	4151347200	1800*	125363	51220
	36	8				ratio(cm/aomdd)=	**2.45**

Table 1: Comparison of search space sizes with time of compilation. The time limit was 30 minutes. *The last instance timed out, so the ratio or/aomdd is left out since the AOMDD may not have necessarily finished compiling.

langfordsat(2,k)	n	w*					
	c	h	|d|	#OR	time	#cm(OR)	#aomdd
	9	6		2^{9}	0.0010	28	27
k=3	19	8	2			ratio(cm/aomdd)=	**1.04**
						ratio(or/aomdd)=	**18.96**
	30	22		2^{30}	0.8800	13152	7168
k=5	147	28	2			ratio(cm/aomdd)=	**1.83**
						ratio(or/aomdd)=	$\mathbf{1.50 \times 10^{5}}$
	63	48		2^{63}	70.5130	131175	59045
k=7	499	58	2			ratio(cm/aomdd)=	**2.22**
						ratio(or/aomdd)=	$\mathbf{1.56 \times 10^{14}}$
k=9	108	84	2	2^{108}	1800*	659010	262279
	1187	100				ratio(cm/aomdd)=	**2.45**

Table 2: Same as Table 1, but with the problem encoded as a CNF/SAT problem. *Timed out.

The next set of experiments demonstrates the canonicity of AOMDDs and demonstrates the notion of semantic treewidth. We created constraint networks with only the equality constraint. It is clear that the graph structure can be just a chain as equality is transitive. On the other hand, we can also explicitly place constraints over each pair of variables yielding a complete graph. The induced width of any chain graph is 1, but the induced width of a complete graph is equal to the number of vertices in the graph minus 1. Table 4 presents the results. We can see that the number of AOMDD nodes is the same, despite the drastic differences in the induced widths of each graph. This suggests to us that the semantic treewidth of this problem is 1.

Conclusion

The combination of the two frameworks results in a framework that is superior to either one in terms of the search space. It allows us to encode the same information about a given problem exactly in an exponentially more compact form when compared with the standard OR search pace. The feature of canoncity is also useful, as compiling a constraint network into an AOMDD can reveal whether is truly as hard as its induced

problem	n	c	w*	h	time	#cm(OR)	#aomdd	ratio(cm/aomdd)	ratio(or/aomdd)
uf20-001	20	86	15	19	0.1780	3059	1607	**1.90**	**652.51**
uf20-002	20	88	14	17	0.0790	1322	743	**1.78**	**1411.27**
uf20-003	20	86	15	18	0.3510	6129	2655	**2.31**	**394.94**
uf20-004	20	90	15	19	0.4340	7551	3379	**2.23**	**310.32**
uf20-005	20	90	14	17	0.0960	1591	827	**1.92**	**1267.93**
uf20-006	20	86	14	19	0.2640	4419	2375	**1.86**	**441.51**
uf20-007	20	87	14	17	0.0640	1128	587	**1.92**	**1786.33**
uf20-008	20	89	15	19	0.3030	5356	2347	**2.28**	**446.77**
uf20-009	20	89	13	19	0.1330	2489	1357	**1.83**	**772.72**
uf20-010	20	90	16	19	0.2690	4636	2163	**2.14**	**484.78**
uf20-011	20	85	15	18	0.1350	2512	994	**2.53**	**1054.91**
uf20-012	20	88	15	18	0.1000	1817	995	**1.83**	**1053.85**
uf20-013	20	85	15	19	0.0360	736	470	**1.57**	**2231.01**
uf20-014	20	88	15	19	0.1820	3213	1700	**1.89**	**616.81**
uf20-015	20	88	14	17	0.0410	732	395	**1.85**	**2654.62**
uf20-016	20	86	15	19	0.2750	4794	2251	**2.13**	**465.83**
uf20-017	20	89	15	19	0.2480	4452	2015	**2.21**	**520.39**
uf20-018	20	83	14	19	0.3120	5517	2934	**1.88**	**357.39**
uf20-019	20	89	15	19	0.1070	2002	1121	**1.79**	**935.39**
uf20-020	20	85	15	19	0.1110	2051	1186	**1.73**	**884.13**

Table 3: Results on Uniform Random 3-SAT Benchmarks. The size of the OR space is always 2^{20}.

eq(n,10)	graph	c	w*	h	time	#aomdd
n=10	chain	9	1	5	0.0240	**91**
	complete	45	9	9	0.0660	**91**
n=50	chain	49	1	25	0.1420	**491**
	complete	1225	49	49	1.1130	**491**
n=100	chain	99	1	50	0.3120	**991**
	complete	4950	99	99	5.5900	**991**

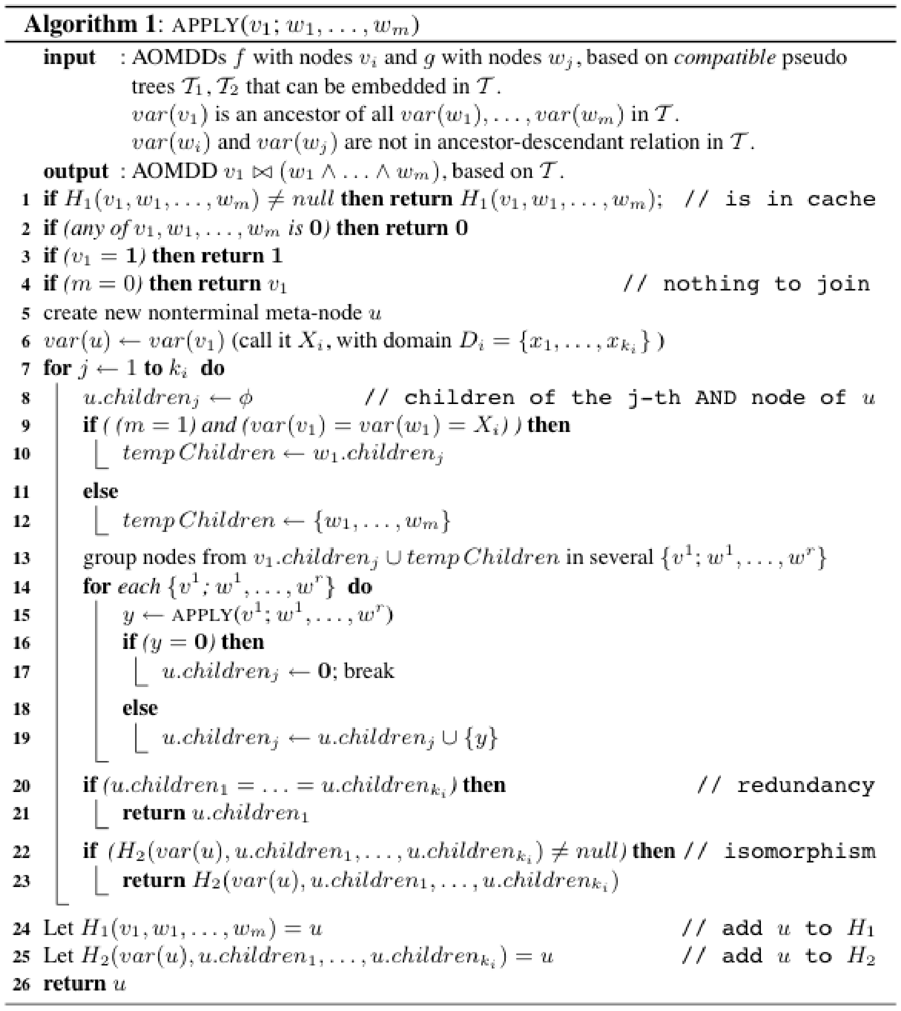
Table 4: Demonstration of canonicity and semantic treewidth.

width may suggest. The experiments presented in this report were performed on toy problems, so it may be interesting to perform experiments on real world problems to see how compact their representation can get to answer questions about how difficult they really are.

Acknowledgements

I would like to thank Rina Dechter for pointing me to additional literature on AOMDDs for graphical models in general and Radu Marinescu for providing the source code for compiling `.wcsp` files into AOMDDs.

Appendix



```
Algorithm 1: APPLY($v_1; w_1, \ldots, w_m$)
input  : AOMDDs $f$ with nodes $v_i$ and $g$ with nodes $w_j$, based on compatible pseudo
         trees $\mathcal{T}_1, \mathcal{T}_2$ that can be embedded in $\mathcal{T}$.
         $var(v_1)$ is an ancestor of all $var(w_1), \ldots, var(w_m)$ in $\mathcal{T}$.
         $var(w_i)$ and $var(w_j)$ are not in ancestor-descendant relation in $\mathcal{T}$.
output : AOMDD $v_1 \bowtie (w_1 \wedge \ldots \wedge w_m)$, based on $\mathcal{T}$.
if $H_1(v_1, w_1, \ldots, w_m) \neq null$ then return $H_1(v_1, w_1, \ldots, w_m)$;   // is in cache
if (any of $v_1, w_1, \ldots, w_m$ is $\mathbf{0}$) then return $\mathbf{0}$
if ($v_1 = \mathbf{1}$) then return $\mathbf{1}$
if ($m = 0$) then return $v_1$                                   // nothing to join
create new nonterminal meta-node $u$
$var(u) \leftarrow var(v_1)$ (call it $X_i$, with domain $D_i = \{x_1, \ldots, x_{k_i}\}$ )
for $j \leftarrow 1$ to $k_i$ do
    $u.children_j \leftarrow \phi$            // children of the j-th AND node of $u$
    if ( ($m = 1$) and ($var(v_1) = var(w_1) = X_i$) ) then
        $tempChildren \leftarrow w_1.children_j$
    else
        $tempChildren \leftarrow \{w_1, \ldots, w_m\}$
    group nodes from $v_1.children_j \cup tempChildren$ in several $\{v^1; w^1, \ldots, w^r\}$
    for each $\{v^1; w^1, \ldots, w^r\}$ do
        $y \leftarrow$ APPLY($v^1; w^1, \ldots, w^r$)
        if ($y = \mathbf{0}$) then
            $u.children_j \leftarrow \mathbf{0}$; break
        else
            $u.children_j \leftarrow u.children_j \cup \{y\}$
    if ($u.children_1 = \ldots = u.children_{k_i}$) then                 // redundancy
        return $u.children_1$
    if ($H_2(var(u), u.children_1, \ldots, u.children_{k_i}) \neq null$) then // isomorphism
        return $H_2(var(u), u.children_1, \ldots, u.children_{k_i})$
Let $H_1(v_1, w_1, \ldots, w_m) = u$                                // add $u$ to $H_1$
Let $H_2(var(u), u.children_1, \ldots, u.children_{k_i}) = u$       // add $u$ to $H_2$
return $u$
```

Figure 3: Pseudocode for the *apply* operator for AOMDDs, as presented in [4].

References

[1] Bryant, R. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers 35*, 8 (1986).

[2] Dechter, R. *Constraint processing.* Morgan Kaufmann, 2003.

[3] Marinescu, R. *AND/OR search strategies for combinatorial optimization in graphical models.* PhD thesis, University of California, Irvine, 2008.

[4] Mateescu, R., and Dechter, R. Compiling constraint networks into and/or multi-valued decision diagrams (aomdds). In *Principles and Practice of Constraint Programming - CP 2006*, F. Benhamou, Ed., vol. 4204 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2006, pp. 329–343. 10.1007/11889205_25.

[5] Mateescu, R., Dechter, R., and Marinescu, R. And/or multi-valued decision diagrams (aomdds) for graphical models. *Journal of Artificial Intelligence Research 33*, 1 (2008), 465–519.