Exploiting Global Structures in Bayesian Network Compilation by Zero-suppressed BDDs

Daisuke Tokoro, Ai Fukunaga, Kiyoharu Hamaguchi, Toshinobu Kashiwabara, and Shin-ichi Minato

Graduate School of Information Science & Technology, Osaka University,
1-5 Yamadaoka, Suita, Osaka, 565-0871, Japan
{d-tokoro,a-fukung}@ics.es.osaka-u.ac.jp
{hama,kashi}@ist.osaka-u.ac.jp
minato@ist.hokudai.ac.jp

Abstract. This paper addresses an algorithm for probabilistic inference for Bayesian Networks (BNs). Our algorithm uses Zero-suppressed BDDs for compiling BNs. We introduce "separation variable" to reflect global structures of BNs, which provides more compact ZBDDs. We show some experimental results to compare our method with the state-of-the-art tool.

Key words: Bayesian Networks, probabilistic inference, Bayesian Networks compilation, zero-Suppressed BDDs, recursive conditioning

1 Introduction

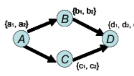
In [4, 5], Darwiche et al. proposed a logical approach for Bayesian Network (BN) inference, and showed dramatic improvement for some networks which have large local structures, such as determinism and context specific independence (CSI) [7], which are features based on specific values in conditional probability tables (CPTs) such as 0.0 in their columns. In their method, a BN is regarded as a Multi-Linear Function (MLF), and the MLF is factored into an Arithmetic Circuit (AC). One can answer multiple queries by evaluating and differentiating the AC in linear time to the AC size. In the factoring process, each BN is encoded to a Conjunctive Normal Form (CNF) of which model corresponds to MLF terms, and then the CNF is factored into an AC utilizing Recursive conditioning (RC) [3] with full caching [1] for exploiting global structures. RC is a method for exact inference for BNs based on network decomposition in form of decomposition trees, or dtrees.

On the other hand, in the method of [1], first, a BN is encoded to Zero-Suppressed BDDs (ZBDDs) [2] and then the ZBDDs are converted to an AC. A ZBDD is a graph-based structure for representing a combinatorial item set (e.g. $\{ab, ac, c\}$). Together with routines for calculating union, intersection etc., ZBDDs can handle sets of extremely large size, which are useful in solving various

[1] For simplification, we refer to RC with full caching simply as RC.

Presented at the 19th International Conference on Inductive Logic Programming (ILP'09).

$$
\begin{aligned}
&\lambda_{a_1}\lambda_{b_1}\lambda_{c_1}\lambda_{d_1}\theta_{a_1}\theta_{b_1|a_1}\theta_{c_1|a_1}\theta_{d_1|b_1c_1}\\
+\ &\lambda_{a_1}\lambda_{b_1}\lambda_{c_1}\lambda_{d_2}\theta_{a_1}\theta_{b_1|a_1}\theta_{c_1|a_1}\theta_{d_2|b_1c_1}\\
+\ &\lambda_{a_1}\lambda_{b_1}\lambda_{c_1}\lambda_{d_3}\theta_{a_1}\theta_{b_1|a_1}\theta_{c_1|a_1}\theta_{d_3|b_1c_1}\\
+\ &\lambda_{a_1}\lambda_{b_1}\lambda_{c_2}\lambda_{d_1}\theta_{a_1}\theta_{b_1|a_1}\theta_{c_2|a_1}\theta_{d_1|b_1c_2}\\
+\ &\cdots\\
+\ &\lambda_{a_2}\lambda_{b_2}\lambda_{c_2}\lambda_{d_3}\theta_{a_2}\theta_{b_2|a_2}\theta_{c_2|a_2}\theta_{d_3|b_2c_2}
\end{aligned}
$$



BCD	Prb(*D*|*B*,*C*)
b1c1d1	$\theta_{d1\|b1c1} = 0.0$
b1c1d2	$\theta_{d2\|b1c1} = 0.5$
b1c1d3	$\theta_{d3\|b1c1} = 0.5$
b1c2d1	$\theta_{d1\|b1c2} = 0.2$
b1c2d2	$\theta_{d2\|b1c2} = 0.3$
b1c2d3	$\theta_{d3\|b1c2} = 0.5$
b2c1d1	$\theta_{d1\|b2c1} = 0.0$
b2c1d2	$\theta_{d2\|b2c1} = 0.0$
b2c1d3	$\theta_{d3\|b2c1} = 1.0$
b2c2d1	$\theta_{d1\|b2c2} = 0.2$
b2c2d2	$\theta_{d2\|b2c2} = 0.3$
b2c2d3	$\theta_{d3\|b2c2} = 0.5$

A	Prb(*A*)
a1	$\theta_{a1} = 0.4$
a2	$\theta_{a2} = 0.6$

AB	Prb(*B*|*A*)
a1b1	$\theta_{b1\|a1} = 0.2$
a1b2	$\theta_{b2\|a1} = 0.8$
a2b1	$\theta_{a2\|b1} = 0.8$
a2b2	$\theta_{a2\|b2} = 0.2$

AC	Prb(*C*|*A*)
a1c1	$\theta_{c1\|a1} = 0.5$
a1c2	$\theta_{c2\|a1} = 0.5$
a2c1	$\theta_{c2\|b1} = 0.5$
a2c2	$\theta_{c2\|b2} = 0.5$

Fig. 1. An example of a BN and its MLF.

combinatorial problems as shown in [8]. In this ZBDD-based approach, an MLF is regarded as a combinatorial item set, and is converted to a ZBDD using standard ZBDD library routines with some modification. ZBDDs have features providing compact representations, which naturally utilize local structures in BNs.

In this paper, we propose a BN compilation method utilizing both of local and global structures with ZBDDs. We first encode a given BN to a ZBDD using "separation variables", and then convert the resulting ZBDD to an AC. Separation variables are introduced so that we can utilize a decomposition structure for a BN, and prevent ZBDDs from blowing up. Our method can be implemented in quite a straightforward manner using standard ZBDD library routines enhanced with a few procedures, after constructing a decomposition tree for a BN.

We test our method with some benchmark networks, and compare the compilation and inference time with the state-of-the-art compiler.

Related to the work in this paper, a ZBDD-based EM algorithm has been proposed[9], and used in programming language in symbolic statistical modeling PRISM[10]. We expect that enhancing the EM algorithm for our ZBDD-based BN representation would provide more expressive and powerful models for such logic programming languages with statistical models incorporated.

2 Background

2.1 Bayesian networks and Multi-Linear functions

BN is a directed acyclic graph (DAG), where each node of a BN corresponds to discrete random variable X and is related to CPT which represents the conditional probabilities of the value of X with given parent variables.

In [4, 5], BN is represented as MLF which consists of two types of variables, an *indicator variables* λ_x for each value $X = x$, and a *parameter variable* $\theta_{x|u}$ for each CPT parameter Pr(x|u). For example, a BN and its MLF are shown in Fig.1. The terms in the MLF are in one-to-one correspondence with the rows of the joint distribution for the network. With some evidence e, we can calculate Pr(e) by adding all the joints containing e, that is, by setting indicators that contradict e to 0 and the other indicators to 1.

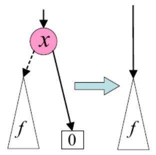


Fig. 2. The ZBDD reduction rule.

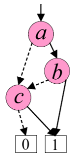


Fig. 3. A ZBDD representing the combination item set $S = \{ab, ac, c\}$.

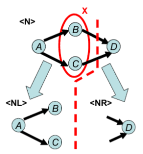


Fig. 4. A decomposition of $\langle N \rangle$ to $\langle NL \rangle$ and $\langle NR \rangle$ by a cutset X.

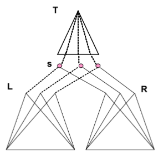


Fig. 5. ZBDD decomposition using separation variable s for Fig.4.

Obviously, the size of an MLF can be exponential in size, but we can factor the MLF to a structure of smaller size. One of such structures is an AC, which is a rooted DAG, where an internal node represents the sum or product of its children, and a leaf represents a constant or variable. Once we construct an AC, we cannot only calculate $\Pr(e)$ but also answer various queries in linear time to the AC size. We say this process compiling BNs based on MLFs.

2.2 MLF factorization using Zero-suppressed BDD

Zero-suppressed BDD (ZBDD) [2] is a variant of binary decision diagram (BDD) for efficient manipulation of combinatorial item sets, which has the following reduction rules, and gives canonical forms under a fixed variable ordering.

- Delete all nodes whose 1-edge directly points to a 0-sink node, and jump through to the destination of 0-edge, as shown in Fig.2
- Share equivalent nodes as in ordinary BDDs.

An example of a ZBDD representing combinatorial item set $S = \{ab, ac, c\}$ is shown in Fig.3. Each of the paths from root to 1-sink corresponds to the item which contains the variables appeared as source nodes node of 1-edges in the path.

An MLF is a polynomial formula of indicator and parameter variables. Each MLF can be regarded as a combinatorial item set, since each term is simply a combination of variables. By regarding parameter and indicator variables as ZBDD variables, we can factor MLF to ZBDD. An example of a MLF and a ZBDD representing it is shown in Fig.6. ZBDDs can exploit local structures of determinism which corresponds that a parameter variable equal to 0 by simply deleting the item containing the variable.

3 BN compilation using ZBDD and Recursive Conditioning

3.1 Recursive Conditioning and ZBDD factorization

RC [3] uses conditioning and case analysis to decompose a network into smaller subnetworks, each of which can be handled independently and recursively. That

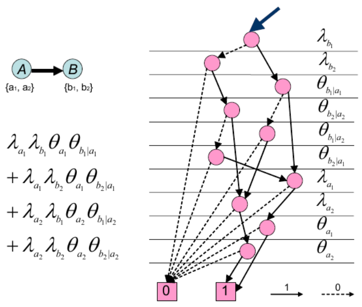

A
B
{a1, a2}
{b1, b2}
$\lambda_{a_1}\lambda_{b_1}\theta_{a_1}\theta_{b_1|a_1}$
$+\lambda_{a_1}\lambda_{b_2}\theta_{a_1}\theta_{b_2|a_1}$
$+\lambda_{a_2}\lambda_{b_1}\theta_{a_2}\theta_{b_1|a_2}$
$+\lambda_{a_2}\lambda_{b_2}\theta_{a_2}\theta_{b_2|a_2}$
λ_{b_1}
λ_{b_2}
$\theta_{b_1|a_1}$
$\theta_{b_2|a_2}$
$\theta_{b_1|a_2}$
$\theta_{b_2|a_1}$
λ_{a_1}
λ_{a_2}
θ_{a_1}
θ_{a_2}
0
1
1
0

Fig. 6. An example of a BN and its ZBDD.

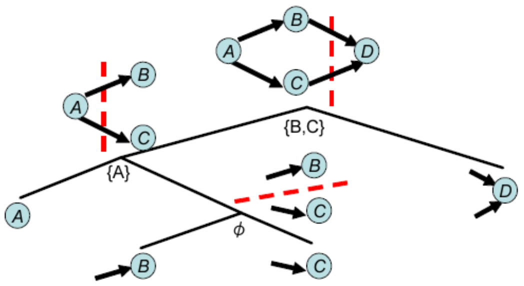


Fig. 7. A dtree representing a decomposition process for BN Fig.1.

is, BN is decomposed to two subnetworks by a set of BN nodes called cutset. Fig.4 shows that a network $\langle N\rangle$ is decomposed into two subnetworks of $\langle NR\rangle$ and $\langle NL\rangle$ by a cutset X containing B and C. We can handle each of the decomposed subnetworks independently, and also recursively until they are decomposed to a single node. This decomposition results in a decomposition tree (dtree). Fig.7 is the dtree for BN in Fig.1. The leaf nodes of the dtree are decomposed single nodes, which correspond to CPTs.

In our method, we utilize this decomposition feature in the MLF factorization with ZBDDs by introducing new ZBDD variables called separation variables. Fig.5 shows the ZBDD for decomposition of Fig.4 using separation variable s. The paths T to L correspond to the MLF terms of the decomposed subnetwork $\langle NL\rangle$, and the paths T to R correspond to the MLF terms of the decomposed subnetwork $\langle NR\rangle$ containing separation variable s. The indicator variables related to random variables X are contained in T which represents a set of all possible instantiations for X. L and R contain, respectively, the ZBDD variables related to the corresponding subnetworks except for those in X. These ZBDDs representing decomposed subnetworks are recursively separated by separation variables, and this recursive separation follows the structure of the dtree.

3.2 Compilation method

Firstly, we decide a variable order of ZBDD. In the order, there appear first indicator and separation variables which are ordered by traversing the dtree in preorder, and last appear parameter variables ordered at will.

Next we construct ZBDDs for leaf nodes of the dtree, each of which corresponds to a CPT by considering each row of the CPT as an element in a combinatorial item set. This can be done using standard ZBDD subroutines "add" and "multiply" corresponding to '+' and '·' in MLFs respectively. Then we "merge" these ZBDDs at each internal node of the dtree traversing the dtree from leaf to root, where two decomposed subnetworks are merged to one network. For example in Fig.4, subnetworks $\langle NL\rangle$ and $\langle NR\rangle$ are merged into $\langle N\rangle$.

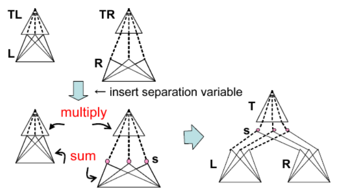


Fig. 8. The merging algorithm.

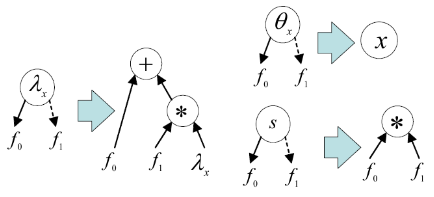


Fig. 9. The replacing rule for the ZBDD to the AC.

That is, using the dtree of Fig.7, we first merge two leaf ZBDDs B and C, resulting in ZBDD BC, and then, A and BC, resulting in ZBDD ABC. Similarly we obtain ZBDD ABCD from ABC and D. Merging of ZBDD NL and NR to ZBDD NLR at each internal node of the dtree we use is shown in Fig.8, where TL and TR contain higher level variables than separation variable s which is related to the internal node of the dtree. First, we insert the separation variable s to NR, this can be achieved using the ZBDD operation "change". Then, we calculate the Cartesian product of two ZBDDs TL and TR resulting in T using the multiplication algorithm proposed in [1] and each 0-edge of s is connected to corresponding path of L and each 1-edge of s is connected to corresponding path of R.

Finally, the resulting ZBDD is converted to an AC. This is done by conversion rules shown in Fig.9.

4 Experimental results

We applied our method and the state-of-the-art compiler provided at `http://reasoning.cs.ucla.edu/ace/` to some benchmark BNs available at `http://www.cs.huji.ac.il/labs/compbio/Repository`. We used a computer with Intel Pentium4, 2.40GHz, 2GB of memory. Experimental results are shown in Table1. In the table, AceT shows the results for ACE using the tabular variable elimination, AceC shows those using CNF based algorithm, Ace shows the common results for AceT and AceC, ZBDD shows our method, the parameter Max Clust. is ln of the maximum cluster size, and '-' means out of memory. Tabular variable elimination is the variant of [6] which is similar to our method but uses ADDs instead of ZBDDs. The detail of the implementation of this method is not available. In this result, the dtree is constructed by the method written in [3]. Our method uses elimination orders provided the benchmark site, but Ace computes the orders based on "minfill heuristic".

We cannot compare our method and Ace directly because each uses the different order. From the column Max Clust., we can see that we use better elimination orders than Ace but our method generally inferior to AceT, where the performance of the elimination order greatly depends on Max Clust. We think the reason of this result is that the MLF factorization with ZBDD cannot exploit sufficiently local structures of CSI, which is the suppression of don't cares

Table 1. experimental result

BN	Max Clust.		Compile Time(s)			AC Edge Count			Inference time(s)		
name	Ace	Zbdd	AceT	AceC	Zbdd	AceT	AceC	Zbdd	AceT	AceC	Zbdd
Pigs	17.4	12.1	0.602	41	3.43	1,267,412	1,579,748	2,033,650	0.072	0.1	0.66
Water	20.8	15.5	1.208	0.864	3.25	124,578	62,308	254,898	0.13	0.064	0.07
Mildew	21.4	14.4	1.351	2470	6.97	2,416,452	2,671,212	2,373,890	0.118	0.258	0.84
Diabetes	17.2	12.1	3.911	8603	42	15,476,258	15,648,826	25,024,900	0.631	2.241	17.65
Munin1	26.2	18.2	-	1768	-	-	23,295,630	-	-	2.632	-
Munin2	18.9	12.0	1.876	513	7	4,222,134	4,323,695	4,465,550	0.2	0.287	1.67
Munin3	17.3	12.0	1.184	240	10	2,652,334	2,469,818	4,547,260	0.134	0.189	2.01
Munin4	19.6	13.9	3.853	6.906	35	4,643,186	5,080,126	8,219,610	0.21	0.369	5.77

of the logic form rather than 0 suppression. We are now working on utilizing this feature in our method.

Compared with AceC, our method is superior to AceC in some networks, and these networks tend to have a large number of values for some random variables. This is because ZBDD can compress the multiple values of CPT better than CNF based method because of 0 suppression property.

References

1. S. Minato, K. Sato and T. Sato, "Compiling Bayesian Networks by Symbolic Probability Calculation Based on Zero-suppressed Bdds," In Proc. of 20th International Joint Conference of Artificial Intelligence(IJCAI-2007), pp.2550-2555, 2007.
2. S. Minato, "Zero-suppressed BDDs for set manipulation in combinatorial problems," In Proc. of 30th Design Automation Conf.(DAC-93), pp.272-277, 1993.
3. A. Darwiche, "Recursive conditioning," Artificial Intelligence, vol.126, No.1-2, pp.5-41, 2001.
4. A.Darwiche, "A Differential Approach to Inference in Bayesian Networks," Journal of the ACM, Vol.50, No.3, pp.280-305, May. 2003.
5. M. Chavira and A. Darwiche, "Compiling Bayesian Networks with Local Structure," In Proc. 19th International Joint Conference on Artificial Intelligence(IJCAI-2005), pp.1306-1312, Aug. 2005.
6. M. Chavira and A. Darwiche, "Compiling Bayesian Networks Using Variable Elimination," In Proc. of the 20th International Joint Conference on Artificial Intelligence (IJCAI), 2007, pp. 2443-2449.
7. Craig Boutilier, Nir Friedman, Moises Goldszmidt, and Daphne Koller, "Context.specific independence in bayesian networks," In Proc. Twelfth Conf. on Uncertainty in Artificial Intelligence (UAI-96), pp.115-123, Aug. 1996.
8. D. E. Knuth, "The art of computer programming Vol.4," Fascicle 1b, 2009.
9. M. Ishihata, T. Sato and S. Minato, "Propositionalizing the EM algorithm by BDDs," Late breaking paper at the 18th International Conference on ILP-2008, 2008.
10. T. Sato and Y. Kameya, "Parameter learning of logic programs for symbolic-statistical modeling," J. of Artificial Intelligence Research 15 , pp.391-454, 2001.