Extraction of NAT Causal Structures Based on Bipartition

Yang Xiang
School of Computer Science
University of Guelph
Canada

Abstract

Non-impeding noisy-And Trees (NATs) provide a general, expressive, and efficient causal model for conditional probability tables (CPTs) in discrete Bayesian networks (BNs). A BN CPT may either be directly expressed as a NAT model or be compressed into one. Once CPTs in BNs are so expressed or compressed, complexity of inference (both space and time) can be significantly reduced. The most important operation in encoding or compressing CPTs into NAT models is extracting the NAT structure from interaction patterns between causes. The existing method does so by referencing a NAT database and an associated search tree. Although both are constructed offline, their complexity is exponential on the number of causes. In this work, we propose a novel method for NAT extraction from causal interaction patterns based on bipartition of causes. The method does not require the support of a NAT database and the related search tree, making NAT extraction more efficient and flexible.

Introduction

We consider expressing BN CPTs as or compressing them into multi-valued NAT models (Xiang 2012b), where NAT stands for Non-impeding noisy-And Tree or NIN-AND Tree. Once so expressed or compressed, inference efficiency (space and time) can be significantly improved (Xiang 2012a). For instance, two orders of magnitude speedup in lazy propagation is achieved in BNs where the number of parents per node is bounded at 11 (Xiang and Jin 2016).

A number of space-efficient models exist, including noisy-OR (Pearl 1988), noisy-MAX (Henrion 1989; Diez 1993), CSI (Boutilier et al. 1996), recursive noisy-OR (Lemmer and Gossink 2004), tensor-decomposition (Vomlel and Tichavsky 2012), and cancellation model (Woudenberg, van der Gaag, and Rademaker 2015). Merits of NAT models include being based on simple causal interactions (reinforcement and undermining), expressiveness (recursive mixture, multi-valued), generality (generalizing noisy-OR, noisy-MAX and DeMorgan (Maaskant and Druzdzel 2008)), and support of much more efficient inference (Xiang and Jin 2016).

To compress a target BN CPT into a NAT model, the following method has been applied (Xiang and Liu 2014; Xiang and Jiang 2016). A partial PCI (Pairwise Causal Interaction) pattern is first extracted from the target CPT. From the PCI pattern, compatible candidate NATs are retrieved. Which candidate NAT becomes the final choice is determined by parameterization.

In that framework, candidate NATs are extracted from the PCI pattern using a NAT database and an associated search tree. A target CPT is defined over an effect and its n causes. Each n value is associated with a distinct space of alternative NATs, which populate the NAT database indexed by n. The size of the NAT database grows super-exponentially on n (see Table 1 below). The main contribution of this work is a novel method that extracts NATs from PCI patterns without using NAT databases and related search trees.

Background

We briefly review background on NAT models and further details can be found in (Xiang 2012b). Consider an effect e and its set of n causes $C = \{c_1, ..., c_n\}$ that are multi-valued and graded. The domain of e is $D_e = \{e^0, ..., e^\eta\}$ $(\eta \geq 1)$, where e^0 is *inactive*, $e^1, , ..., e^\eta$ are *active*, and a higher index signifies higher intensity (graded). The domain of c_i is $D_i = \{c_i^0, ..., c_i^{m_i}\}$ $(m_i > 0)$. An active value may be written as e^+ or c_i^+.

A causal event is *success* or *failure* depending on if e is active at a given intensity, is *single-* or *multi-causal* depending on the number of active causes, and is *simple* or *congregate* depending on the effect value range. More specifically,

$$P(e^k \leftarrow c_i^j) = P(e^k | c_i^j, c_z^0 : \forall z \neq i) \ \ (j > 0)$$

is the probability of a *simple single-causal success*.

$$P(e \geq e^k \leftarrow c_1^{j_1}, ..., c_q^{j_q}) =$$

$$P(e \geq e^k | c_1^{j_1}, ..., c_q^{j_q}, c_z^0 : c_z \in C \setminus X),$$

is the probability of a *congregate multi-causal success*, where $j_1, ..., j_q > 0$, $X = \{c_1, ..., c_q\}$ $(q > 1)$, and it may be denoted as $P(e \geq e^k \leftarrow \underline{x}^+)$. Interactions among causes may be reinforcing or undermining as defined below.

Definition 1 *Let e^k be an active effect value, $R = \{W_1, W_2, ...\}$ be a partition of a set $X \subseteq C$ of causes, $R' \subset R$, and $Y = \cup_{W_i \in R'} W_i$. Sets of causes in R reinforce each other relative to e^k, iff*

$$\forall R' \ P(e \geq e^k \leftarrow \underline{y}^+) \leq P(e \geq e^k \leftarrow \underline{x}^+).$$

They undermine each other iff

$$\forall R'\ P(e \geq e^k \leftarrow \underline{y}^+) > P(e \geq e^k \leftarrow \underline{x}^+).$$

A NAT has multiple NIN-AND gates. A *direct* gate involves disjoint sets of causes $W_1, ..., W_m$. Each input event is a success $e \geq e^k \leftarrow \underline{w}_i^+$ $(i = 1, ..., m)$ and its output event is $e \geq e^k \leftarrow \underline{w}_1^+, ..., \underline{w}_m^+$. Fig. 1 (a) shows a direct

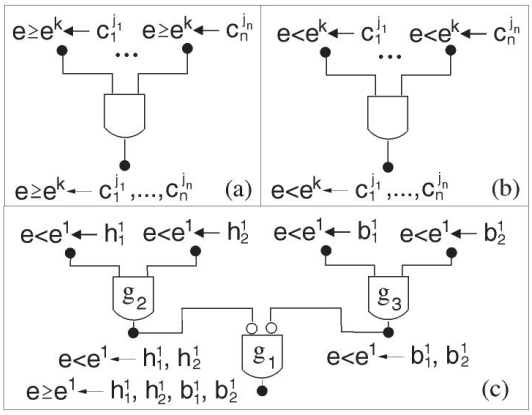

Figure 1: (a) A multi-valued direct NIN-AND gate. (b) A dual NIN-AND gate. (c) A NAT.

gate where each W_i is a singleton. Probability of the output event is

$$P(e \geq e^k \leftarrow \underline{w}_1^+, ..., \underline{w}_m^+) = \prod_{i=1}^{m} P(e \geq e^k \leftarrow \underline{w}_i^+),$$

which encodes undermining causal interaction. Each input event of a *dual* gate is a failure $e < e^k \leftarrow \underline{w}_i^+$ and its output event is $e < e^k \leftarrow \underline{w}_1^+, ..., \underline{w}_m^+$, as shown in Fig. 1 (b). Probability of the output is

$$P(e < e^k \leftarrow \underline{w}_1^+, ..., \underline{w}_m^+) = \prod_{i=1}^{m} P(e < e^k \leftarrow \underline{w}_i^+),$$

which encodes reinforcement. Fig. 1 (c) shows a NAT, where causes h_1 and h_2 reinforce each other, so do b_1 and b_2, but the two groups undermine each other.

A NAT can be depicted by a Root-Labeled-Tree (RLT).

Definition 2 *Let T be a NAT. The RLT L of T is a directed graph obtained from T as follows.*

1. *Delete each gate and direct its inputs to output.*
2. *Delete each non-root label.*
3. *Replace each root label by the corresponding cause.*

Fig. 2 shows a NAT and its RLT. The leaf of RLT corresponds to leaf gate of the NAT. When the leaf gate is dual (or direct), we say that leaf of the RLT is dual (or direct).

The leaf gate of a NAT is at level-one. A gate that feeds into the leaf gate is at level-two, and so on. We refer to levels of non-root nodes of RLTs in the same way, as they correspond to gates in NATs. All gates in the same level have the same type (dual or direct) and gates in adjacent levels differ. An RLT and a leaf type uniquely specifies a NAT.

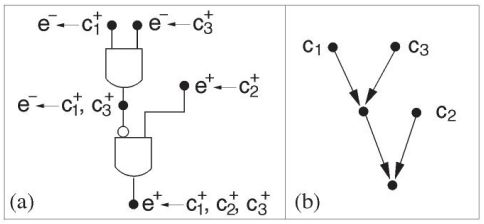

Figure 2: (a) A NAT. (b) The RLT of NAT in (a).

A NAT T has a single leaf z. For $n \geq 3$, leaf z has at least two parents. Let v be a parent of z. If v is a root, we refer to v as a *root parent* of z. If v is a non-root, it is the leaf of a subtree in T. We refer to the subtree as being *induced by leaf* z. In Fig. 2 (b), c_2 is a root parent of the leaf. The leaf has one induced subtree with its *root set* $\{c_1, c_3\}$.

Each NAT uniquely determines the pairwise causal interaction between each pair of causes c_i and c_j $(i \neq j)$, denoted by PCI bit $pci(c_i, c_j) \in \{u, r\}$ (u for reinforcing and r for reinforcing) ((Xiang and Truong 2014)). The value of $pci(c_i, c_j)$ is determined by the common gate of c_i and c_j at the highest level. The NAT in Fig. 1 (c) has $pci(h_1, h_2) = r$ since g_2 is dual and $pci(h_1, b_2) = u$ since g_1 is direct. A PCI pattern (the collection of PCI bits over all pairs of causes) uniquely determines a NAT.

Root Bipartition in NATs

Although a PCI pattern uniquely determines a NAT, it is not obvious which NAT matches a given PCI pattern (over n causes) and the number of candidate NATs is $O(2^{n^2})$. In (Xiang and Liu 2014), a method is proposed to extract a NAT over n causes from its PCI pattern by using a NAT database (for n causes) and a search tree. Both the size of the NAT database and that of the search tree are $O(2^{n^2})$ (generated off-line before compressing target CPTs).

In this work, we propose a novel method of NAT extraction that does not require support of NAT database and the search tree. We assume $n = |C| \geq 3$. We analyze a PCI pattern through its *PCI matrix* (see Fig. 3). In the left, an RLT with $n = 6$ is shown. With leaf type being dual, it uniquely specifies a NAT (hence the caption). The PCI matrix in the right is equivalent to the PCI pattern of the NAT, except each PCI bit is duplicated in diagonally symmetric locations. The diagonal locations are empty.

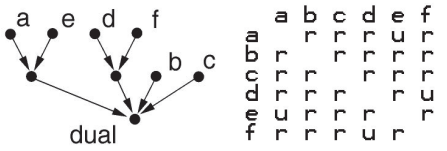

Figure 3: A NAT and its PCI matrix

The leaf of the NAT has two root parents (b and c) and two induced subtrees with root sets $\{a, e\}$ and $\{d, f\}$. Consider dividing root nodes into two *root groups*, where the root set

of each leaf-induced subtree must be contained in one group. Start with root sets $\{a, e\}$, $\{d, f\}$, $\{b\}$ and $\{c\}$. If one root group is made of a single root set, there are 4 groupings. If one root group is made of two root sets, such as $X = \{a, b, e\}$ and $Y = \{c, d, f\}$, there are 3 groupings. Hence, there are 7 distinct root groupings. We refer to such root groupings as *subtree-consistent* root bipartitions.

Consider matrix cells at the intersection of rows indexed by X and columns indexed by Y. The result is the same for each of the three rows: (r, r, r). The same is true if X and Y are switched. The following theorem generalizes the example to reveal an important property of PCI matrices, on which this work is based.

Theorem 1 *Let T be a NAT over C. Let X and Y be a subtree-consistent root bipartition, where $X \neq \emptyset$, $Y \neq \emptyset$, $X \cap Y = \emptyset$, and $X \cup Y = C$. Then, one of the following holds.*

1. $\forall x \in X, \forall y \in Y,\ pci(x, y) = r$
2. $\forall x \in X, \forall y \in Y,\ pci(x, y) = u$

Proof: The leaf of T has at least two parents. Each parent is either a root or the leaf of a subtree induced by leaf of T. Hence, root bipartition into $X \neq \emptyset$ and $Y \neq \emptyset$, where $X \cap Y = \emptyset$ and $X \cup Y = C$, is always possible.

For each pair of $x \in X$ and $y \in Y$, their only common gate is leaf of T. Hence, if leaf of T is dual, $\forall x \forall y\ (pci(x, y) = r)$ is true. If the leaf is direct, $\forall x \forall y\ (pci(x, y) = u)$ is true. □

The uniformity of causal interaction between root groups raises the question whether subtree-consistent root bipartitions may be identified from PCI matrices. The following theorem answers this positively.

Theorem 2 *Let T be a NAT over C, and X, Y be nonempty subsets of root nodes in T ($X \cap Y = \emptyset$, $X \cup Y = C$) such that one of the following holds.*

1. $\forall x \in X, \forall y \in Y,\ pci(x, y) = r$
2. $\forall x \in X, \forall y \in Y,\ pci(x, y) = u$

Then, X and Y form a subtree-consistent root bipartition for the NAT T.

Proof: From Theorem 1, root groups exist such that conditions 1 and 2 hold. We show that if root groups X and Y are not subtree-consistent, neither condition 1 nor 2 holds.

Suppose that $X, Y \subset C$ are nonempty, $X \cap Y = \emptyset$ and $X \cup Y = C$, such that X and Y are not subtree-consistent. That is, there exists a subtree ST induced by the leaf of T, such that one root node of ST satisfies $x \in X$ and another root node of ST satisfies $y \in Y$.

Denote the leaf of T by z. Since z has at least two parents and subtrees induced by z do not share causes, there exists a root node v such that either v is on a subtree H induced by z with $H \neq ST$ (see Fig. 4) or v is a root parent of z (not shown).

Depending on whether z is dual or direct and whether $v \in X$ or $v \in Y$, there are four mutually exclusive and exhaustive cases.

(a) Dual z and $v \in X$

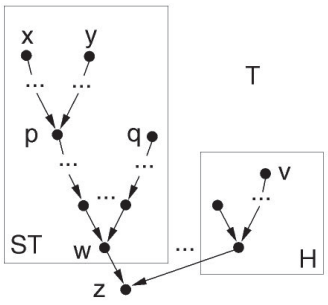

Figure 4: Illustration of proof for Theorem 2

(b) Dual z and $v \in Y$

(c) Direct z and $v \in X$

(d) Direct z and $v \in Y$

With case (a), assume dual z and $v \in X$. For x and y, either $pci(x, y) = u$ or $pci(x, y) = r$. Suppose $pci(x, y) = u$. Then condition 1 does not hold. Since the only common gate of v and y is z (dual), we have $v \in X$, $y \in Y$, and $pci(v, y) = r$. Hence, condition 2 does not hold either.

Next, suppose $pci(x, y) = r$. Then condition 2 does not hold. Let w be the leaf of subtree ST. Since z is dual, w must be direct. From $pci(x, y) = r$, node w cannot be the common gate of x and y at the highest level. That is, there exists an ancestor p of w that is the common gate of x and y at the highest level, and p is dual. This implies that x and y are contained in the the same subtree induced by w.

Since w has at least two parents, there exists either another subtree (not containing x, y, p) induced by w or there exists a root parent of w. Let q be a root in that subtree (see Fig. 4) or be the root parent (not shown). Since w is the common gate of x, y, q at the highest level and w is direct, we have $pci(x, q) = u$ and $pci(q, y) = u$. If $q \in X$, $pci(q, y) = u$ violates condition 1. If $q \in Y$, $pci(x, q) = u$ violates condition 1. Hence, condition 1 does not hold either.

Since cases (a) through (d) are symmetric, the above proof on case (a) can be adapted to cases (b) through (d). □

NAT Identification by PCI Matrices

Based on Theorem 2, we propose the following algorithm suite that extracts a NAT from its PCI matrix.

The first algorithm is InteractBtwSets. As input, it takes a set X of causes, a PCI matrix A over X, and a proper subset $S \subset X$. It determines if any of the two conditions in Theorem 2 is true for root groups S and $X \setminus S$. If so, it returns the NIN-AND gate type that corresponds to the condition. If neither condition is true, it returns null.

Algorithm 1 $InteractBtwSets(X, A, S)$
1 if $\forall x \in S, \forall y \in X \setminus S,\ pci(x, y) = r$,
2 gatetype = dual;
3 else if $\forall x \in S, \forall y \in X \setminus S,\ pci(x, y) = u$,
4 gatetype = direct;
5 else gatetype = null;
6 return gatetype;

The main algorithm is SetNatByPci and it is recursive. As input, it takes a set X of causes and a PCI matrix A over X, and returns the NAT that generates the matrix. It calls InteractBtwSets to analyze the causal interaction between alternative root groups.

Algorithm 2 $SetNatByPci(X, A)$
1 init NAT T with a leaf z only; type(z) = null;
2 init set $D = \emptyset$;
3 for each $x \in X$, do
4 if $\forall y \in X \setminus \{x\}$, $pci(x, y) = r$,
5 type(z) = dual; $D = D \cup \{x\}$;
6 add x as a parent of z in T;
7 else if $\forall y \in X \setminus \{x\}$, $pci(x, y) = u$,
8 type(z) = direct; $D = D \cup \{x\}$;
9 add x as a parent of z in T;
10 if $D = X$, return T;

11 if $D \neq \emptyset$, reduce X and A relative to D;
12 $k = |X|/2$, $D = \emptyset$, $W = \emptyset$;
13 for $i = 2$ to k, do
14 for each $S \subseteq X$ where $|S| = i$, do
15 ibs = InteractBtwSets(X, A, S);
16 if ibs $\neq$ null,
17 if type(z) = null, assign type(z) = ibs;
18 if ibs = type(z),
19 $D = X$, $W = W \cup \{S, X \setminus S\}$;

20 remove each $S \in W$ from W if for some $V \in W$, $S \supseteq V$;
21 D = union of elements in W;
22 for each $S \in W$, do
23 reduce A to matrix B over S;
24 $R = SetNatByPci(S, B)$;
25 add R to T as a subtree induced by z;
26 if $D = X$, return T;

27 R = SetNatByPci($X \setminus D$, B);
28 add R to T as a subtree induced by z;
29 return T;

In the following, we illustrate processing of SetNatByPci by examples.

Example 1 *Consider the example in Fig. 5.*

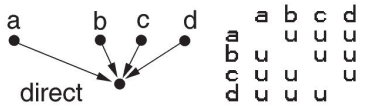

Figure 5: A NAT whose leaf has root parents only and its PCI matrix

The for loop starting at line 3 iterates for each of a, b, c, d, say, in that order. For a, it compares $pci(a, y)$ where $y \in \{b, c, d\}$. These PCI bits are in the matrix row indexed by a. Hence, the test on line 7 is passed, node a is added as a parent of leaf z, and the leaf type is set to direct. Each subsequent iteration adds another parent to z, and the correct NAT is returned in line 10. Note that the result is independent of the order in which $x \in X$ is processed.

Example 2 *Consider the example in Fig. 3. The matrix rows indexed by b and c pass the test at line 4. Hence, b and c are added as root parents of the leaf. This and the above example illustrate that the for loop in lines 3 to 9 adds all root parents of the leaf to T.*

Continuing with the example in Fig. 3, X is reduced to $X = \{a, d, e, f\}$ in line 11 and matrix A is reduced to that in Fig. 6.

Figure 6: PCI matrix in Fig. 3 is reduced relative to $\{b, c\}$.

After line 11, all root parents of the leaf are removed from X. The nested for *loop in lines 13 to 19 processes alternative root bipartitions of X. Each bipartition made of S and $X \setminus S$ is tested starting with S size 2 (line 13). Since the root set of each subtree has at least 2 causes, an obvious upper limit of for index is $|X| - 2$, which is less efficient than limit $|X|/2$ (integer division) defined in line 12. Validity of limit $|X|/2$ is justified in the proof of Theorem 3 below.*

Lines 15 and 16 test causal interactions between S and $X \setminus S$. Grouping $S = \{a, d\}$ and $X \setminus S = \{e, f\}$ does not pass the test, but grouping $S = \{a, e\}$ and $X \setminus S = \{d, f\}$ does. As the result, root groups $\{a, e\}$ and $\{d, f\}$ are added to W (line 19).

As presented, $\{a, e\}$ and $\{d, f\}$ are added twice to W in separate iterations of the inner for loop. The duplicate will be removed in line 20. We omit the potential optimization for simplicity.

Through the for loop in lines 22 to 25, each of $\{a, e\}$ and $\{d, f\}$ will be processed by a recursive call of SetNatByPci. The corresponding subtrees (see Fig. 3) will be extracted and added to T. The initial activation of SetNatByPci terminates in line 26, returning the NAT in Fig. 3.

Example 3 *Consider the example in Fig. 7. The for loop in lines 13 to 19 add root groups $\{a, c\}$, $\{b, d\}$, $\{e, f\}$ as well as groups such as $\{a, b, c, d\}$. Although $\{a, b, c, d\}$ and $\{e, f\}$ form a subtree-consistent root bipartition, $\{a, b, c, d\}$ does not correspond to a leaf-induced subtree in the NAT of Fig. 7. Line 20 removes root groups such as $\{a, b, c, d\}$ from W before each root group is converted to a subtree.*

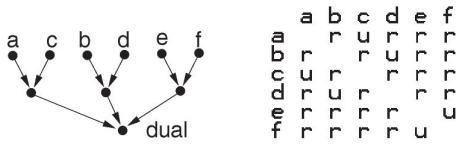

Figure 7: A NAT with 3 subtrees and its PCI matrix

Example 4 *Consider the example in Fig. 8. After the for loop in lines 3 to 9 adds e as a parent of the leaf in T, X is reduced to $X = \{a, b, c, d\}$ and matrix A is reduced accordingly (removing the last row and the last column from*

that in Fig. 8). It may appear that the loop in lines 13 to 19 would add $S = \{a, b\}$ and $Y = X \setminus S = \{c, d\}$ to W, since they satisfy $pci(x, y) = u$ for each $x \in S$ and $y \in Y$. It

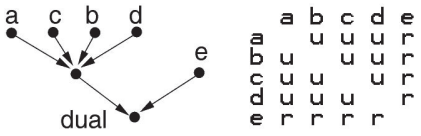

Figure 8: A NAT with one subtree and its PCI matrix

would result in incorrect subtrees with root sets S and Y. Such error is prevented by the test in line 18. Leaf z of T is dual, as determined when e was added to T. The causal interaction between S and Y requires a direct gate that differs from $type(z)$. As the result, S and Y are not added to W.

Since the NAT in Fig. 8 has a single subtree, the for loop in lines 13 to 19 cannot find a subtree-consistent root bipartition that passes the test in line 18. Thus, processing continues to line 27. The subtree with the root set $\{a, b, c, d\}$ will be extracted by recursive call in line 27 and added to T in line 28. The NAT in Fig. 8 is returned in line 29.

Soundness Analysis

The soundness of the algorithm suite in Section is established in Theorem 3.

Theorem 3 *Let Ψ be a NAT over a set X of causes and A be the PCI matrix of Ψ. Then algorithm $SetNatByPci(X, A)$ halts and returns $T = \Psi$.*

Proof: Let ρ be the leaf of Ψ. Since $n = |X| \geq 3$, ρ belongs to one of the following mutually exclusive and exhaustive cases.

1. Leaf ρ has 2 or more root parents only.
2. Leaf ρ has 2 or more non-root parents and 0 or more root parent.
3. Leaf ρ has 1 non-root parent and 1 or more root parent.

In case 1, for each $x \in X$, $\{x\}$ and $X \setminus \{x\}$ form a subtree-consistent root bipartition. By Theorem 1, one of the conditions holds. Hence, each iteration of for loop in lines 3 to 9 identifies one root parent x of ρ. $SetNatByPci(X, A)$ halts on line 10 and returns $T = \Psi$.

In case 2, if ρ has any root parent x, it is correctly identified and added to T as argued above. After line 11 is executed, X is reduced to be the set of root nodes in all subtrees.

By assumption, ρ has at least 2 induced subtrees. For each subtree and its root set S, either $|S| \leq |X|/2$ or $|S| > |X|/2$. If $|S| \leq |X|/2$, S is evaluated when the inner for loop (lines 14 to 19) is run relative to $i = |S|$. If $|S| > |X|/2$, then $|X \setminus S| < |X|/2$ and S is evaluated when the inner for loop is run relative to $i = |X \setminus S|$.

Since S and $X \setminus S$ form a subtree-consistent root bipartition, by Theorem 1, one of the conditions holds. As the result, ibs (line 15) equals the type of ρ, and S and $X \setminus S$ are added to W. Since root sets of distinct subtrees are disjoint, S cannot be removed from W in line 20. The subtree over S is added to T. If $X \setminus S$ is the root set of another subtree, the subtree is also added to T.

Suppose Ψ has $q \geq 3$ subtrees. For each subtree root set S that is added to W, $X \setminus S$ contains multiple subtree root sets and is also added to W. Since each root set contained in $X \setminus S$ is added to W, $X \setminus S$ will be removed from W in line 20. Hence, the for loop in lines 22 to 25 adds exactly the subtrees of Ψ to T. Therefore, $SetNatByPci(X, A)$ halts on line 26 and returns $T = \Psi$.

In case 3, Ψ has a single subtree induced by ρ. All root parents of ρ are identified by the for loop in lines 3 to 9. After line 11, X is the root set of the subtree and A is the PCI matrix over the root set. Hence, X has no other root set to pair, it cannot be added to W, and $W = \emptyset$ after the for loop in lines 13 to 19. Through a recursive argument, the subtree over X is added to T in lines 27 and 28. Therefore, $SetNatByPci(X, A)$ halts on line 29 and returns $T = \Psi$. □

Experiment

To evaluate effectiveness of the algorithm suite, the experimental setup in Fig. 9 is used. The case for $n = 9$ is illustrated and cases for other n values are similar. Note that NAT models are local models in BNs. Due to conditional independence encoded in BNs, the number of causes per BN CPT is not unbounded.

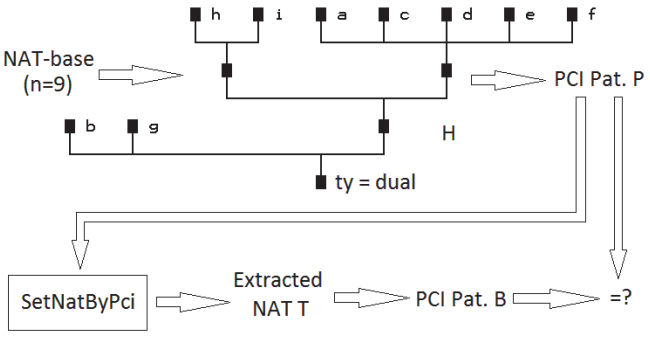

Figure 9: Experimental setup.

For each n value between 4 and 9, the NAT database is used to retrieve all NATs (dual leaf) of n causes exhaustively. For the two NATs of the same RLT, since they differ by the type of leaf gate only and their processing are symmetric, only the NAT of the dual leaf is experimentally processed. For each NAT H, its PCI pattern P is derived. P is used as the input to $SetNatByPci$, implemented in Java and executed in a 2.9 GHz ThinkPad X230 laptop. After the NAT T is extracted by $SetNatByPci$, its PCI pattern B is also derived. If $B = P$, the NAT extraction is verified.

The verification is sound since a PCI pattern uniquely identifies a NAT (Xiang and Truong 2014). Hence, comparison between B and P is equivalent to comparison between T and H. On the other hand, comparison of B and P can be conducted much more easily than testing isomorphism between T and H.

Table 1 summarizes the experimental results. For all NATs with n value in the range of 4 to 9, the PCI pattern of the extracted NAT matches exactly that of the retrieved NAT.

Table 1: Runtime for NAT recovery

n	No. NATs	Time (sec)	Time/1000 NATs (sec)
4	26	0.016	0.62
5	236	0.093	0.39
6	2,752	0.500	0.18
7	39,208	7.13	0.18
8	660,032	136.87	0.21
9	12,818,912	3,638.62	0.28

The 1st column indicates the number of causes n. The 2nd column shows the total number of NATs of a dual leaf for a given n. The 3rd column indicates the total runtime to process all NATs of a given n. The last column is the runtime per 1,000 NATs. Note that the processing for each NAT includes all steps illustrated in Fig. 9. Hence, the runtime consumed by $SetNatByPci$ is only a fraction of the time reported.

Remarks

Expressing BN CPTs as or compressing them into NAT models can significantly improve efficiency in BN inference. The most important operation in encoding or compressing CPTs into NAT models is extraction of NAT structures from PCI patterns. The contribution of this work is a novel method for NAT extraction based on bipartition of causes. In comparison with the existing method, it does not require the support of a NAT database and the associated search tree, making NAT extraction more efficient and flexible. The soundness of the method is formally established and its effectiveness is experimentally evaluated.

A number of future work can be identified. Although the efficiency of the proposed method is empirically evaluated, a formal analysis of its complexity is desirable. Its efficiency relative to the existing method (based on a NAT database and a search tree) is yet to be experimentally compared.

The proposed extraction method assumes the input of a full PCI pattern from an unknown NAT. To express or compress a target CPT, often only partial PCI patterns (with missing PCI bits) are available. PCI patterns that do not correspond to any NAT can also occur (due to error or noise in PCI pattern acquisition). The result presented in this paper provides a solid foundation for further research in order to tackle such cases.

Acknowledgement

We thank anonymous reviewers for constructive criticism. Financial support from the Discovery Grant, NSERC, Canada is acknowledged.

References

Boutilier, C.; Friedman, N.; Goldszmidt, M.; and Koller, D. 1996. Context-specific independence in Bayesian networks. In *Proc. 12th Conf. on Uncertainty in Artificial Intelligence*, 115–123.

Diez, F. 1993. Parameter adjustment in Bayes networks: The generalized noisy OR-gate. In Heckerman, D., and Mamdani, A., eds., *Proc. 9th Conf. on Uncertainty in Artificial Intelligence*, 99–105. Morgan Kaufmann.

Henrion, M. 1989. Some practical issues in constructing belief networks. In Kanal, L.; Levitt, T.; and Lemmer, J., eds., *Uncertainty in Artificial Intelligence 3*. Elsevier Science Publishers. 161–173.

Lemmer, J., and Gossink, D. 2004. Recursive noisy OR - a rule for estimating complex probabilistic interactions. *IEEE Trans. on System, Man and Cybernetics, Part B* 34(6):2252–2261.

Maaskant, P., and Druzdzel, M. 2008. An independence of causal interactions model for opposing influences. In Jaeger, M., and Nielsen, T., eds., *Proc. 4th European Workshop on Probabilistic Graphical Models*, 185–192.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann.

Vomlel, J., and Tichavsky, P. 2012. An approximate tensor-based inference method applied to the game of Minesweeper. In *Proc. 7th European Workshop on Probabilistic Graphical Models, Springer LNAI 8745*, 535–550.

Woudenberg, S.; van der Gaag, L.; and Rademaker, C. 2015. An intercausal cancellation model for Bayesian-network engineering. *Inter. J. Approximate Reasoning* 63:3247.

Xiang, Y., and Jiang, Q. 2016. Compression of general Bayesian net CPTs. In Khoury, R., and Drummond, C., eds., *Advances in Artificial Intelligence, LNAI 9673*. Springer. 285–297.

Xiang, Y., and Jin, Y. 2016. Multiplicative factorization of multi-valued NIN-AND tree models. In Markov, Z., and Russell, I., eds., *Proc. 29th Inter. Florida Artificial Intelligence Research Society Conf.*, 680–685. AAAI Press.

Xiang, Y., and Liu, Q. 2014. Compression of Bayesian networks with NIN-AND tree modeling. In vander Gaag, L., and Feelders, A., eds., *Probabilistic Graphical Models, LNAI 8754*. Springer. 551–566.

Xiang, Y., and Truong, M. 2014. Acquisition of causal models for local distributions in Bayesian networks. *IEEE Trans. Cybernetics* 44(9):1591–1604.

Xiang, Y. 2012a. Bayesian network inference with NIN-AND tree models. In Cano, A.; Gomez-Olmedo, M.; and Nielsen, T., eds., *Proc. 6th European Workshop on Probabilistic Graphical Models*, 363–370.

Xiang, Y. 2012b. Non-impeding noisy-AND tree causal models over multi-valued variables. *International J. Approximate Reasoning* 53(7):988–1002.