Direct Causal Structure Extraction from Pairwise Interaction Patterns in NAT Modeling Bayesian Networks

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 CPT may be directly expressed as a NAT model or compressed into a NAT model. Once CPTs are NAT-modeled, efficiency of BN inference (both space and time) can be significantly improved. One of the critical operations in NAT modeling CPTs is extracting NAT structures from interaction patterns between causes. The existing method does so through NAT databases coupled with search trees. Although the databases and search trees are compiled offline, the computation is costly and the dependency of NAT extraction on them adds a resource requirement for online computation. We present a novel method for direct NAT structure extraction from full and valid causal interaction patterns, based on bipartitions of causes. We then extend the method to NAT extraction from partial and invalid interaction patterns. The resultant algorithm suite enables direct NAT extraction from all conceivable practical scenarios, with significantly reduced computational complexity, while eliminating dependency on NAT databases and search trees.

Keywords: Graphical models; probabilistic inference; machine learning; Bayesian networks; causal models; non-impeding noisy-AND trees

1. Introduction

Although conditional independence expressed by the structure of BNs avoids combinatorial explosion on the number of variables, BNs are still subject to exponential growth of space and inference time on the number of

causes per effect variable in each CPT [1]. Several local models have been proposed for space-efficient encoding of dependency between an effect and its causes. They include noisy-OR [1], noisy-MAX [2, 3], context-specific independence (CSI) [4], probability trees [5], recursive noisy-OR [6], Non-Impeding Noisy-AND Tree (NIN-AND Tree or NAT) [7], DeMorgan [8], tensor-decomposition [9], and cancellation model [10]. These local models not only reduce the number of parameters needed to acquire CPTs, but also enable significant reduction in inference time, e.g., by exploiting CSI in arithmetic circuits (ACs) and sum-product networks (SPNs) [11, 12, 13], or by exploiting causal independence in NAT models [14].

This work focuses on representing BN CPTs as NAT models [15] or compressing them into NAT models. Merits of NAT models include the following:

1. Linear space on the number of causes per effect.

2. Based on simple causal interactions: reinforcement and undermining.

3. Expressiveness. They support recursive mixture of basic causal interactions. They allow multi-valued variables that are either ordinal or nominal (relaxing the common restriction of being *graded* [16]).

4. Generality. They generalize other local models, including noisy-OR, noisy-MAX [14], and DeMorgan [15]. It has been shown [16] that compression of general CPTs into NAT models has superior accuracy than noisy-MAX (and hence also noisy-OR). Due to the generality of NAT models, such superiority is also expected relative to DeMorgan.

5. Supporting more efficient inference. Two orders of magnitude speedup in lazy propagation is achieved in very sparse NAT-modeled BNs [14].

6. Causal independence in NAT models is orthogonal to CSI. Hence, they provide an alternative to CSI for more efficient inference in BNs.

A NAT model over an effect and n causes consists of a NAT topology (whose size is linear on n) and a set of numerical parameters (whose cardinality is linear on n). It compactly represents a BN CPT. Every pair of causes in a NAT model either undermines each other in causing the effect,

[1] We use causality interchangeably with asymmetric dependency. Each child variable is treated as an effect and its parent variables as the causes.

or reinforcing each other. The interaction can be specified with one bit of value u (undermining) or r (reinforcing). The collection of such bits over all pairs of causes defines a pairwise causal interaction (PCI) pattern [17].

A PCI pattern may be *full* (with one bit for each cause pair) or *partial* (with missing bits). A PCI pattern may also be *valid* (produced by at least one NAT) or *invalid* (cannot be generated by any NAT). A full, valid PCI pattern uniquely identifies a NAT [17, 18]. This property enables PCI patterns to play a critical role in NAT topology acquisition, either when compressing a given BN CPT into a NAT model or when learning a BN CPT from data as a NAT model. The task of topology acquisition takes a PCI pattern as input and returns a NAT topology. This operation is referred to as *NAT structure extraction*.

For instance, a target BN CPT can be compressed into a NAT model as follows [19, 20]: One or more full or partial PCI patterns are first obtained from the target CPT. From the patterns, compatible candidate NATs are extracted from a NAT database through a search tree. All candidate NATs are parameterized into NAT models. The output NAT is selected to minimize a distance measure between target CPT and CPT of the output NAT model.

To support more efficient inference with BNs, NAT modeling can be applied as follows: First, compress each CPT in the BN into a NAT model [19, 20]. Since common BN inference algorithms cannot directly operate on NAT models, the NAT-modeled BN is further converted using one of two techniques. One technique applies *multiplicative factorization* to each NAT model in the BN, converting the NAT-modeled BN into an equivalent Markov network [21]. Another technique applies *de-causalization* [22] to convert the NAT-modeled BN into an equivalent normal BN. After either conversion, the resultant representation is operable by standard inference algorithms. For instance, up to two orders of magnitude speedup in lazy propagation is achieved in a wide range of sparse BNs [22].

The role of the two conversion techniques is the following: A NAT model specifies a CPT of exponential size by a linear number of parameters. Standard BN inference algorithms need to reference values in CPT, and cannot use NAT parameters directly. To determine values in CPT from NAT parameters, traversal of the NAT topology is typically needed [23, 24], which does not integrate efficiently with the control structure of standard BN inference algorithms. On the other hand, if the CPT of a NAT model is pre-computed before inference, the advantage of NAT models (linear complexity) is lost. Multiplicative factorization and de-causalization techniques convert NAT-

modeled BNs to standard graphical models, while retaining the space efficiency of NAT models. Hence, they allow standard BN inference algorithms to take advantage of efficiency of NAT models without any modification.

The NAT structure extraction utilized in the existing compression process requires a NAT database for each n value, that encodes all alternative NATs for n causes, as well as an associated search tree. Since the number of distinct NATs over n causes grows super-exponentially on n, so does the size of NAT database. For instance, it takes 40 hours to generate (offline) the NAT database and search tree for $n = 9$ [19]. Hence, requirement of NAT databases is a source of computational burden, both for offline generation (compute time) and for online extraction (storage space and loading time).

The contribution of this research is a novel framework and associated algorithms that directly extract NATs from PCI patterns without needing the support of NAT databases and associated search trees. First, we present a new approach to analyze a PCI pattern through bipartitions of its causes. We uncover several useful properties of such bipartitions. Second, we develop an algorithm to extract a NAT structure directly from a full, valid PCI pattern, driven by bipartition analysis. Third, we relax the requirement of full patterns so that NAT structures can be extracted from partial, valid PCI patterns. Finally, we go beyond the previous work [19] where only valid PCI patterns were considered, and further extend the direct algorithm to NAT extraction from invalid patterns.

These advancements accomplish NAT structure extraction in all conceivable application scenarios: full valid PCI patterns, partial valid patterns, full invalid patterns, and partial invalid patterns. They all operate on direct extraction, eliminating the burden of NAT databases and search trees.

Section 2 reviews background on NAT models. The task of fault tolerant, direct NAT structure extraction is specified in Section 3. Section 4 formally analyzes properties of cause bipartitions, which leads to the direct extraction algorithm from full valid PCI patterns, covered in Section 5. Additional properties of cause bipartitions are revealed in Section 6 to characterize invalid PCI patterns. They lay the theoretical foundation for extending the algorithm to direct extraction from full invalid PCI patterns, presented in Section 7. Further extension to extraction from partial, possibly invalid patterns is presented in Section 8. Section 9 reports experimental evaluation of the algorithm suite.

2. Background

We briefly review background on NAT models [15, 16]. NAT models are defined over *causal variables* that may be ordinal or nominal, and are more general than *graded* variables [16] as commonly assumed in local models.

Definition 1. *A variable x that can be either inactive or be active possibly in multiple ways, and is involved in a causal relation, is a* **causal variable** *if when all causes of the effect are inactive, the effect is inactive with certainty.*

A NAT model is defined over an effect e and a set of $n \geq 2$ causes $C = \{c_1, ..., c_n\}$, all of which are causal variables. Possible values of effect e are $e^0, ..., e^\eta$ ($\eta \geq 1$), written as $e \in \{e^0, ..., e^\eta\}$, and possible values of cause c_i are $c_i^0, ..., c_i^{m_i}$, where $i = 1, ..., n$ and $m_i \geq 1$. C and e form a single *family* of variables in a BN, where C is the set of parents of e. Values e^0 and c_i^0 are *inactive*. Other values (may be written as e^+ or c_i^+) are *active*. A higher index may signify higher intensity (required by graded variables), but it is not necessary.

Example 1. *Owning a pet (op) has health benefit of fewer doctor visits (fv), and so does regular walking (rw). Here, op and rw are causes of the effect fv, and other causes also exist. Suppose $op \in \{none, dog, cat, fish, bird, other\}$ and $fv \in \{none, 1-4\%, 5-8\%, 9-12\%, 13\%+\}$. Both op and fv are causal variables, whose inactive values are none. When $op = none$ and all other causes of fv are also inactive, the effect takes value $fv = none$ with certainty. Effect fv is ordinal. Cause op is nominal and is not graded. Its values $none, dog, ..., other$ can be denoted $op^0, op^1, ..., op^5$, respectively.*

A causal event is a *success* or *failure* depending on whether e is rendered active at a certain range of values, is *single-* or *multi-causal* depending on the number of active causes, and is *simple* or *congregate* depending on the value range of e. In particular, $P(e^k \leftarrow c_i^j) = P(e^k | c_i^j, c_z^0 : \forall z \neq i)$ $(j > 0)$ is the probability of a *simple single-causal success*, and $P(e \geq e^k \leftarrow c_1^{j_1}, ..., c_q^{j_q}) = P(e \geq e^k | c_1^{j_1}, ..., c_q^{j_q}, c_z^0 : c_z \in C \setminus X)$ is the probability of a *congregate multi-causal success*, where $j_1, ..., j_q > 0$, $X = \{c_1, ..., c_q\}$ $(q > 1)$, possibly denoted as $P(e \geq e^k \leftarrow \underline{x}^+)$. Interactions among causes may be one of the followings:

Definition 2. *Let e^k be an active effect value, $R = \{W_1, ..., W_m\}$ $(m \geq 2)$ be a partition of a set $X \subseteq C$ of causes, $S \subset R$, and $Y = \cup_{W_i \in S} W_i$. Sets of*

causes in R **reinforce** *each other relative to* e^k*, iff* $\forall S\ P(e \geq e^k \leftarrow \underline{y}^+) \leq P(e \geq e^k \leftarrow \underline{x}^+)$*. They* **undermine** *each other iff* $\forall S\ P(e \geq e^k \leftarrow \underline{y}^+) > P(e \geq e^k \leftarrow \underline{x}^+)$*.*

Causal interactions are expressed in a NAT model by NIN-AND gates. A *direct* gate involves disjoint sets of causes $W_1, ..., W_m$. Each input event is a success $e \geq e^k \leftarrow \underline{w}_i^+$ $(i = 1, ..., m)$, e.g., Fig. 1 (a) where each W_i is a singleton. The output event is the success $e \geq e^k \leftarrow \underline{w}_1^+, ..., \underline{w}_m^+$. Its probability 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.

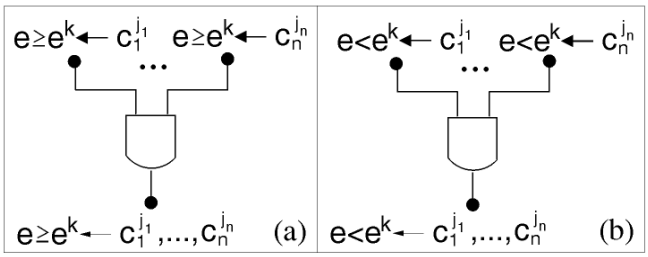

Figure 1: (a) A direct NIN-AND gate. (b) A dual NIN-AND gate.

Each input event of a *dual* gate is a failure $e < e^k \leftarrow \underline{w}_i^+$, e.g., Fig. 1 (b). The output event is the failure $e < e^k \leftarrow \underline{w}_1^+, ..., \underline{w}_m^+$. Its probability $P(e < e^k \leftarrow \underline{w}_1^+, ..., \underline{w}_m^+) = \prod_{i=1}^m P(e < e^k \leftarrow \underline{w}_i^+)$ encodes reinforcement.

A NAT consists of multiple NIN-AND gates arranged in a tree topology. Its input events involve disjoint subsets of causes. There is a single output gate (the *leaf* gate), whose output is a causal event that involve all causes.

Example 2. *Consider surface enhancers for furniture and home renovation. Acidic enhancers* h_1 *and* h_2 *are more effective when both are applied. Basic enhancers* b_1 *and* b_2 *work similarly. However, when both types are combined, their effectiveness is reduced. Fig. 2 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. Ovals at the input of* g_1 *negate output events from* g_2 *and* g_3*.*

A *NAT model* consists of a NAT and a set of numerical parameters. Each input event of a NAT typically involves a single cause. In that case, each parameter is a *single-causal probability* such as $P(e^k \leftarrow c_i^j)$, where e^k and c_i^j are active values. The NAT and the set of parameters uniquely define a CPT over e and C.

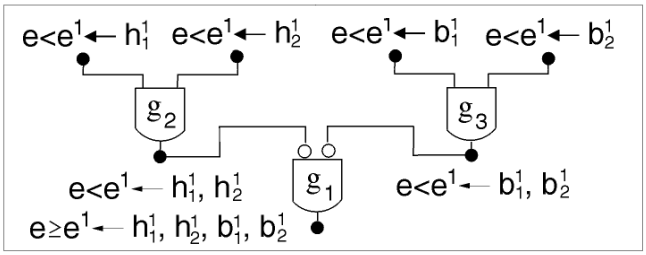

Figure 2: A NAT to model causal interaction among surface enhancers.

Example 3. *Suppose all variables in the NAT of Fig. 2 are ternary (indicated by superscripts 0, 1, and 2). Then a NAT model consists of the NAT and a set of* $2 \times 2 \times 4 = 16$ *single-causal probabilities. The 4 probabilities that involves* h_1 *are in the form* $P(e^k \leftarrow h_1^j)$*, where* $k = 1, 2$ *and* $j = 1, 2$*. From the NAT and the 16 parameters, the CPT* $P(e|h_1, h_2, b_1, b_2)$ *is uniquely defined, whose space has the size of* $3^5 = 243$*.*

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

Definition 3. *Let* T *be a NAT where each root label involves a single cause. The RLT 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.*

The leaf gate of a NAT is at level 1. A gate that feeds into the leaf gate is at level 2, and so on. All gates in the same level of a NAT have the same type (dual or direct), and gates in adjacent levels differ. For instance, g_2 and g_3 in Fig. 2 are both dual, and g_1 is direct.

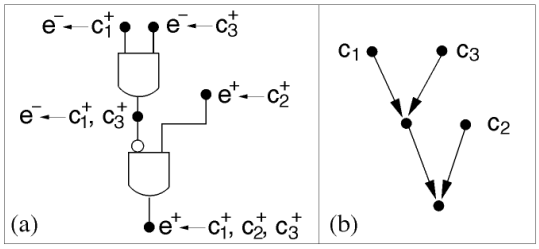

Figure 3: A NAT (a) and its RLT (b).

The RLT of the NAT in Fig. 3 (a) is shown in (b). The leaf of RLT corresponds to the leaf gate of the NAT. When the leaf gate is dual (or

direct), the leaf of RLT is said to be dual (or direct). Levels of nodes of a RLT are specified similarly to levels of gates in the NAT. For example, the leaf in Fig. 3 (b) is direct at level 1, node c_2 is at level 2, and c_1 is at level 3. A RLT and its leaf type uniquely specify a NAT. For instance, from the RLT in Fig. 3 (b) and direct leaf type, the NAT in (a) can be uniquely recovered. Since RLTs are more compact, we often study NATs in terms of their RLTs, and refer to a RLT with a given leaf type as a NAT.

A NAT T has a single leaf z and z has at least two parents. Each parent v of z is the leaf of a subtree *induced* by z. When v is a root, it is referred to as a *root parent* of z, and the induced subtree is trivial. In Fig. 3 (b), there are two subtrees induced by the leaf. One subtree is trivial, where c_2 is a root parent of the leaf, and the *root set* of the subtree is $\{c_2\}$. The root set of the other subtree is $\{c_1, c_3\}$.

Each NAT uniquely defines the pairwise causal interaction between each pair of causes c_i and c_j $(i \neq j)$, denoted by a PCI bit $\pi(c_i, c_j) \in \{u, r\}$. The value $\pi(c_i, c_j)$ is defined by the common gate of c_i and c_j at the highest level [17]. A collection of PCI bits is a PCI pattern π. If π includes one bit for each cause pair, it is a *full* pattern. Otherwise, it is *partial*.

Example 4. *The NAT in Fig. 2 involves 4 causes. A full PCI pattern consists of $C(4, 2) = 6$ PCI bits. For instance, $\pi(h_1, h_2) = r$, since g_2 is dual. PCI bit $\pi(h_1, b_2) = u$, since g_1 is direct.*

3. Direct NAT Structure Extraction

Given a full PCI pattern generated from a NAT, the pattern uniquely identifies the NAT [17, 18]. This enables PCI patterns to play a critical role in NAT topology acquisition, either when compressing a given BN CPT into a NAT model or when learning a BN CPT from data as a NAT model.

For example, to improve inference efficiency, each BN CPT can be compressed into a NAT model [19, 20]. Let $P_T(e|c_1, ..., c_n)$ be a target CPT over a BN family. The output is a NAT model M over $e, c_1, ..., c_n$. The criterion is that CPT $P_M(e|c_1, ..., c_n)$ defined by M minimizes a distance measure between P_M and P_T, such as Kullback-Leibler (KL) divergence. Main steps of compression are the following:

1. Obtain possibly multiple competing PCI patterns from P_T. This is achieved by analyzing probabilities in P_T that are relevant to each PCI bit.

For instance, to determine $\pi(c_i, c_j)$ where e, c_i, c_j are binary, 3 probabilities from P_T are compared: $p = P(e \geq e^1 \leftarrow c_i^1), q = P(e \geq e^1 \leftarrow c_j^1)$, and $s = P(e \geq e^1 \leftarrow c_i^1, c_j^1)$. If $s < min(p, q)$, then $\pi(c_i, c_j) = u$. If $s \geq max(p, q)$, then $\pi(c_i, c_j) = r$. If one of them holds for every pair of c_i, c_j, a full PCI pattern is obtained. If none of them holds, $\pi(c_i, c_j)$ is a *missing* bit, and the final PCI pattern is partial. See [16] for a general treatment of PCI pattern computation. Given c_i, c_j, computing s takes $O(\eta)$ time. Since the number of PCI bits is $C(n, 2)$, the complexity of PCI pattern computation is $O(\eta\ n^2)$.

2. From each PCI pattern, extract one candidate NAT if the pattern is full, and multiple candidate NATs if it is partial.

3. For each candidate NAT, estimate parameters to fully specify a corresponding NAT model. This is achieved by constrained gradient descent.

4. Compute P_M for each NAT model M and its distance from P_T. Return a NAT model with the minimal distance from P_T.

Before illustrating compression with an example, we comment on its experimental computation cost in time and space. The runtime is reported in [16], where the largest target CPT has $n = 6$ and each variable has $k = 4$ possible values. This results in a CPT size of $4^7 = 16,384$. The average runtime of compression is 102 seconds (including all 4 steps above). Step 3 consumes most of the time, and time by other steps are negligible in comparison. The above reports on runtime. To conduct step 2, a NAT database and an associated search tree are constructed offline, which takes 40 hours for $n = 9$ (see below).

For space, the most expensive is step 2, due to the need of a NAT database and the associated search tree, each of whcih has a size super-exponentially on n (see below). For $n = 9$, the database has a size of 108 MByte. The other steps require space not much more than that for storing the target CPT (exponential on n). For the above CPT, the space to store it is 16,384.

In this work, we present a novel approach for step 2, which eliminates the expensive offline computation time associated with step 2, as well as the super-exponential space associated with that step. Next, we illustrate compression with an example.

Example 5. *Consider the well-known Alarm BN [25]. Let the target CPT be* $P(VentLung|VentTube, KinkedTube, Intubation)$*, where variables have domain sizes 4, 4, 2, 3, respectively. The CPT has 72 parameters.*

This CPT has a persistent leaky cause [16], which we denote by c_0*. Hence, the output NAT model is over effect* $e = VentLung$ *and the set of causes* $C = \{c_0, VentTube, KinkedTube, Intubation\}$*. For simplicity, we denote* $C = \{c_0, v, k, i\}$*. Its full PCI pattern has* $C(4, 2) = 6$ *PCI bits. From the target CPT, two full PCI patterns are obtained:*

$$\{\pi_1(c_0, v) = u, \pi_1(c_0, k) = r, \pi_1(c_0, i) = u, \pi_1(v, k) = r, \pi_1(v, i) = u, \pi_1(k, i) = u\},$$
$$\{\pi_2(c_0, v) = u, \pi_2(c_0, k) = u, \pi_2(c_0, i) = u, \pi_2(v, k) = r, \pi_2(v, i) = u, \pi_2(k, i) = u\}.$$

Subsequently, NAT T_1 *in Fig. 4 (a) is extracted from* π_1*, and* T_2 *in (b) is extracted from* π_2*.*

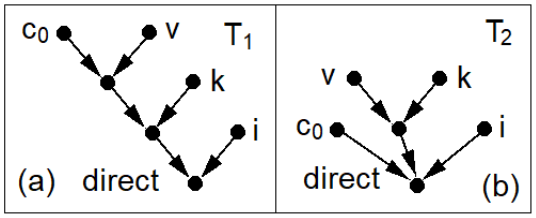

Figure 4: (a) NAT extracted from PCI pattern π_1. (b) NAT extracted from π_2.

Single-causal probabilities for each T_i *are then parameterized by constrained gradient descent, which transforms* T_1 *and* T_2 *into NAT models* M_1 *and* M_2*. The single-causals for* M_2 *are the following:*

$$\begin{array}{lll}
P(e^1 \leftarrow c_0^1) = 0.184 & P(e^2 \leftarrow c_0^1) = 0.01 & P(e^3 \leftarrow c_0^1) = 0.368 \\
P(e^1 \leftarrow v^1) = 0.01 & P(e^2 \leftarrow v^1) = 0.011 & P(e^3 \leftarrow v^1) = 0.026 \\
P(e^1 \leftarrow v^2) = 0.218 & P(e^2 \leftarrow v^2) = 0.361 & P(e^3 \leftarrow v^2) = 0.320 \\
P(e^1 \leftarrow v^3) = 0.01 & P(e^2 \leftarrow v^3) = 0.01 & P(e^3 \leftarrow v^3) = 0.093 \\
P(e^1 \leftarrow k^1) = 0.234 & P(e^2 \leftarrow k^1) = 0.086 & P(e^3 \leftarrow k^1) = 0.578 \\
P(e^1 \leftarrow i^1) = 0.153 & P(e^2 \leftarrow i^1) = 0.587 & P(e^3 \leftarrow i^1) = 0.052 \\
P(e^1 \leftarrow i^2) = 0.097 & P(e^2 \leftarrow i^2) = 0.208 & P(e^3 \leftarrow i^2) = 0.594
\end{array}$$

The KL distance between the CPT of each M_i *and the target CPT is evaluated. As the result,* M_2 *has the smaller distance and is the output NAT model. It reduces the number of parameters from 72 of target CPT to 21 single-causals. Its Euclidean distance is 0.30 and its KL distance is 0.63.*

The focus of this work is on the second operation above. The input of the operation is a full or partial PCI pattern, and the output is one or more corresponding NATs. We refer to the task as *NAT structure extraction*. Although a full PCI pattern uniquely determines a NAT, it is not obvious which NAT matches a given pattern over n causes and the number of candidate NATs is $O(2^{n(n-1)/2})$. The existing method constructs offline a NAT database and an associated search tree for each n value [19]. Since the database stores all alternative NATs over n causes, and each leaf node of the search tree maps to a unique NAT in the database, the space complexity of both the database and the search tree is $O(2^{n(n-1)/2})$. During online compression, for each PCI pattern over n causes, the corresponding search tree retrieves one or more NATs from the database. We refer to the method as *database supported* extraction.

Table 1 shows the total number K of alternative NATs for $n = 2, ..., 9$ [18, 26]. It illustrates how the sizes of NAT databases and search trees grow

Table 1: Total numbers of distinct NATs for n values 2 through 9.

n	2	3	4	5	6	7	8	9
K	2	8	52	472	5,504	78,416	1,320,064	25,637,824

super-exponentially on n. Although constructed offline, they are a source of computational burden. For $n = 9$, it takes 40 hours to generate the database and search tree [19]. Since NAT models are local models (one BN family per model), n does not grow unboundedly, due to conditional independence encoded in BNs. Nevertheless, it is costly to generate NAT databases and search trees when n grows beyond 9.

To alleviate these costs, we present a novel approach for NAT extraction without requiring support of NAT databases and search trees. We refer to the approach as *direct* extraction.

The input to direct extraction is a PCI pattern that may be full or partial. In addition, the pattern may be valid or invalid as defined below. First, we relate two PCI patterns over the same set of causes.

Definition 4. *Let π and ψ be PCI patterns over a set C of causes. If for each pair of $c_i, c_j \in C$ $(i \neq j)$, we have $\pi(c_i, c_j) = \psi(c_i, c_j)$ whenever both*

$\pi(c_i, c_j)$ and $\psi(c_i, c_j)$ are defined, then π and ψ are **compatible**. *Otherwise, they are* **incompatible**.

Either π or ψ may be partial. Only PCI bits defined under both π and ψ are relevant to compatibility. Next, we relate a PCI pattern to a NAT over the same set of causes.

Definition 5. *Let π be a PCI pattern over a set C of causes. Then π is* **valid** *if there exists a NAT over C whose PCI pattern ψ is compatible with π. Otherwise, π is* **invalid**.

The above π may be partial. A full PCI pattern over n causes has $C(n, 2)$ bits and $2^{C(n,2)}$ variations, not all of which are valid. For $n = 2$, there are 2 NATs (Table 1). A full pattern has $C(2, 2) = 1$ bit and is always valid. For $n = 3$, there are 8 NATs. A full pattern has $C(3, 2) = 3$ bits. Hence, every PCI pattern is valid. For $n = 4$, there are 52 NATs. There are $2^{C(4,2)} = 2^6 = 64$ full patterns, of which 64 - 52 = 12 patterns are invalid. For $n = 7$, there are $2^{21} = 2,097,152$ full patterns and 78,416 NATs. The number of invalid full patterns is 2,018,736. In the remainder of the paper, we present direct extraction of NAT structure from PCI patterns that are either full or partial, and are either valid or invalid.

4. Properties of Cause Bipartitions

Our method of direct extraction is based on bipartitioning causes in C. We introduce cause bipartitions from two alternative perspectives in Section 4.1, the *uniform causal bipartition* and the *subtree-consistent bipartition*, and analyze their properties in Section 4.2.

4.1. Alternative Cause Bipartitions

Causes in C can be divided according to causal interactions.

Definition 6. *Let C ($|C| \geq 2$) be a set of causes, X and Y be non-empty subsets of C where $X \cap Y = \emptyset$ and $X \cup Y = C$, and π be a full PCI pattern over C. Then $\{X, Y\}$ is a* **uniform causal bipartition** *of C under π, if (1) $\forall x \in X, \forall y \in Y,\ \pi(x, y) = r$, or (2) $\forall x \in X, \forall y \in Y,\ \pi(x, y) = u$.*

Note that π may be invalid. When $C = \{x, y\}$, π has one bit, and $\{\{x\}, \{y\}\}$ forms a uniform causal bipartition trivially. For $C = \{x, y, z\}$, every π is valid and identifies a NAT T, as discussed above. At least one cause x is parent of the leaf in T, and $\{\{x\}, \{y, z\}\}$ is a uniform causal bipartition.

A PCI pattern can be alternatively expressed as a *PCI matrix*. Fig. 5 (a) shows the RLT expression of a NAT over 6 causes. Its PCI matrix is shown in (b). Although PCI matrices are less compact than equivalent PCI

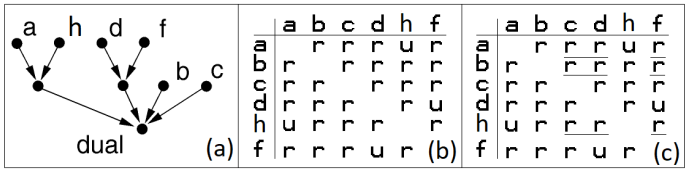

Figure 5: (a) A NAT. (b) Its PCI matrix. (c) Visually checking uniform causal bipartition.

patterns, uniform causal bipartitions can be visually identified more easily through PCI matrices. Let π be the PCI pattern expressed in Fig. 5 (b). Consider cause bipartition $\{X = \{a, b, h\}, Y = \{c, d, f\}\}$ under π. Identify matrix cells at the intersection of rows indexed by X and columns indexed by Y, as underlined in Fig. 5 (c). Since these cells have the same bit value r, $\{X, Y\}$ is a uniform causal bipartition. We often use PCI matrices below in place of PCI patterns, refer to them interchangeably, and denote both by π.

Given a PCI pattern π, there may not exist any uniform causal bipartition under π.

Definition 7. *Let π be a full PCI pattern over a set C ($|C| \geq 2$) of causes. If there exists no uniform causal bipartition under π, then π is a* **PCI core** *and C is the* **domain** *of the PCI core.*

From discussion following Def. 6, there exists no PCI core when $|C| = 2$ and 3. Consider the PCI matrix π in Fig. 6 (a). Since the row indexed by a is not uniform, $\{\{a\}, \{b, c, d\}\}$ is not a uniform causal bipartition. Checking each row indexed by b, c, and d, none of the bipartitions $\{X, Y\}$ is causally uniform when $|X| = 1$. For $X = \{a, d\}$ and $Y = \{b, c\}$, consider cells at the intersection of rows indexed by X and columns indexed by Y, as underlined in Fig. 6 (b). Since underlined values are not uniform, $\{\{a, d\}, \{b, c\}\}$ is not a uniform causal bipartition. It can be verified that none of the bipartitions $\{X, Y\}$ is causally uniform when $|X| = 2$. Hence, π is a PCI core. The smallest PCI core (over the least number of causes) occurs when $|C| = 4$.

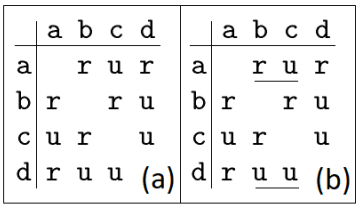

Figure 6: (a) PCI matrix over $C = \{a, b, c, d\}$. (b) Visually checking bipartition.

Given a NAT over C, causes can also be divided based on NAT topology.

Definition 8. *Let T be a NAT over C and $\{X, Y\}$ be a bipartition of C, where $X \neq \emptyset$, $Y \neq \emptyset$, $X \cap Y = \emptyset$, and $X \cup Y = C$. If for each leaf-induced subtree of T and its root set R, either $R \subseteq X$ or $R \subseteq Y$ holds, then $\{X, Y\}$ is a* **subtree-consistent bipartition** *of C with respect to T.*

In Fig. 3, $\{\{c_2\}, \{c_1, c_3\}\}$ is a subtree-consistent bipartition of C, but $\{\{c_1\}, \{c_2, c_3\}\}$ is not. Theorem 1 establishes existence of subtree-consistent bipartitions.

Theorem 1. *Every NAT T over a set C ($|C| \geq 2$) of causes has at least one subtree-consistent bipartition of C.*

Proof: We regard T as RLT of the NAT. Let z be the leaf of T. Since $|C| \geq 2$, z has at least two parents. Let x be a parent of z. If x is a root, then $\{\{x\}, C \setminus \{x\}\}$ is a subtree-consistent bipartition with respect to T. Otherwise, x is the leaf of a subtree. Let X be the root set of the subtree. Then $\{X, C \setminus X\}$ is a subtree-consistent bipartition with respect to T. □

4.2. Relation of Cause Bipartitions

We use RLT representation of a NAT below, as in the above proof. Theorem 2 relates the two types of bipartitions: Whenever a bipartition of C is subtree-consistent, it is also a uniform causal bipartition.

Theorem 2. *Let T be a NAT over C and π be the PCI pattern of T. Every subtree-consistent bipartition of C with respect to T is a uniform causal bipartition under π.*

Proof: By Theorem 1, a subtree-consistent bipartition of C with respect to T exists. Denote the bipartition by $\{X, Y\}$, where $X \neq \emptyset$, $Y \neq \emptyset$, $X \cap Y = \emptyset$, and $X \cup Y = C$.

Let z be the leaf of T. For each pair of $x \in X$ and $y \in Y$, their only common descendent is z, which corresponds to their unique common gate in the NAT. Hence, if z is dual, $\pi(x, y) = r$ and condition (1) of Def. 6 is true. If z is direct, $\pi(x, y) = u$ and condition (2) of Def. 6 is true. □

Correlation of subtree-consistent bipartitions and uniform causal bipartitions established by Theorem 2 raises the question whether subtree-consistent bipartitions can be identified through uniform causal bipartitions. Theorem 3 asserts this positively.

Theorem 3. *Let T be a NAT over C, π be the PCI pattern of T, and $\{X, Y\}$ be a uniform causal bipartition of C under π. Then, $\{X, Y\}$ is a subtree-consistent bipartition with respect to T.*

Proof: We show that if $\{X, Y\}$ is not subtree-consistent, then it is not a uniform causal bipartition. In particular, we show that neither condition (1) nor (2) of Def. 6 holds.

Assume that $\{X, Y\}$ is not subtree-consistent relative to T. Denote the leaf of T by z. There exists a subtree ST induced by z, where one root x of ST satisfies $x \in X$ and another root y of ST satisfies $y \in Y$ (Fig. 7 (a)).

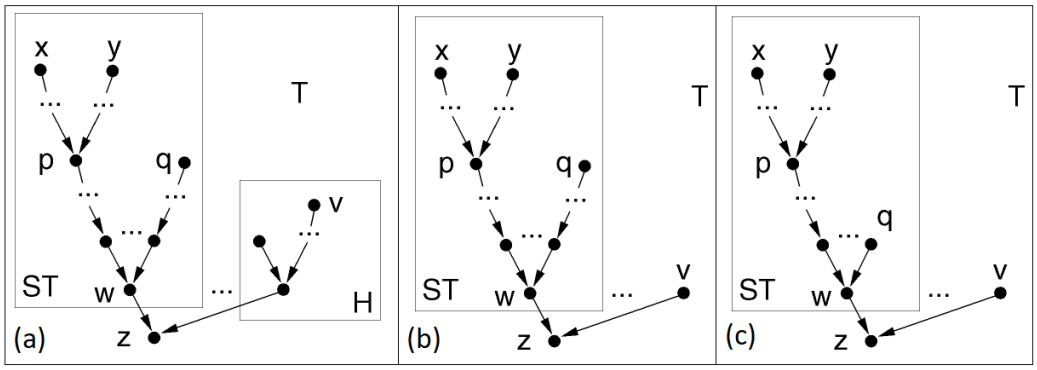

Figure 7: Illustration of proof for Theorem 3.

Leaf z has at least two parents, and subtrees induced by z have disjoint subsets of causes. Hence, there exists a root v such that either v is on a subtree $H \neq ST$ induced by z (Fig. 7 (a)), or v is a root parent of z (b).

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, $v \in X$; (B) dual z, $v \in Y$; (C) direct z, $v \in X$; and (D) direct z, $v \in Y$.

Case (A) involves dual z and $v \in X$. We have either $\pi(x, y) = u$ or $\pi(x, y) = r$. Suppose $\pi(x, y) = u$. Then condition (1) of Def. 6 does not hold. Since the only common descendent of v and y is z (dual), we have $v \in X$, $y \in Y$, and $\pi(v, y) = r$. Hence, condition (2) does not hold either.

Next, suppose $\pi(x, y) = r$. Then condition (2) does not hold. Let w be the leaf of subtree ST (Fig. 7 (a)). Since z is dual, w must be direct. From $\pi(x, y) = r$, node w cannot be the common descendent of x and y at the highest level. That is, there exists an ancestor p of w that is the common descendent of x and y at the highest level, and p is dual. This implies that x and y are contained in a subtree with leaf p (Fig. 7 (a)).

Since w has at least two parents, either there exists another non-trivial 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 (Fig. 7 (b)) or be the root parent (Fig. 7 (c)). Since w is the common descendent of x, y, q at the highest level and w is direct, we have $\pi(x, q) = u$ and $\pi(q, y) = u$. If $q \in X$, $\pi(q, y) = u$ violates condition (1). If $q \in Y$, $\pi(x, q) = u$ violates condition (1). Hence, condition (1) does not hold either.

Since cases (A) through (D) are symmetric, the above proof on (A) can be adapted to (B) through (D). □

5. NAT Extraction by PCI Matrices

The task of direct extraction is to extracts a NAT from an input PCI pattern π. Theorem 3 suggests that uniform causal bipartitions under π correspond to subtree-consistent bipartitions relative to the unknown NAT, and hence provide hints to its topology. Section 5.1 presents an algorithm suite to explore this opportunity for direct extraction. Its behavior is illustrated with example traces in Section 5.2. The completeness of the algorithm suite is analyzed in Section 5.3. Section 5.4 presents a refinement of the algorithm suite for improved efficiency.

5.1. Extraction Algorithm

Algo. 1 takes as input a set C of causes, a PCI matrix π over C, and a proper subset $X \subset C$ from which a bipartition $\{X, Y\}$ (line 1) is defined.

It checks if $\{X, Y\}$ is a uniform causal bipartition. If so, it returns the NIN-AND gate type that encodes the causal interaction, and nil otherwise.

Algorithm 1. $IsUniCausalBipart(C, \pi, X)$

1 $Y = C \setminus X$*;*
2 *if* $\forall x \in X, \forall y \in Y$, $\pi(x, y) = r$ *holds, gatetype = dual;*
3 *else if* $\forall x \in X, \forall y \in Y$, $\pi(x, y) = u$ *holds, gatetype = direct;*
4 *else gatetype = nil;*
5 *return gatetype;*

Algo. SetNatByPciFV below takes as input a set C of causes and a full valid (FV) PCI matrix π over C, and returns a NAT. It calls IsUniCausal-Bipart to evaluate alternative bipartitions. $InNat$ and $InNat'$ are sets of causes added to the current NAT T (in terms of RLT). The $Subsets$ collects subsets X and Y for each uniform causal bipartition $\{X, Y\}$. Matrix reduction (lines 7 and 19) is exemplified in Fig. 8, where the matrix in (a) over $\{a, b, c, d, h\}$ is *reduced* to that in (b) over $\{a, b, c, d\}$.

	a	b	c	d	h
a		r	u	r	u
b	r		r	u	u
c	u	r		u	u
d	r	u	u		u
h	u	u	u	u	

(a)

	a	b	c	d
a		r	u	r
b	r		r	u
c	u	r		u
d	r	u	u	

(b)

Figure 8: PCI pattern in (a) is reduced to that in (b).

SetNatByPciFV is organized into 4 sections. Lines 1 to 6 initialize the NAT T with the leaf z only, and search for root parents of z. Lines 7 to 14 search for alternative root bipartitions, where each subset in a bipartition is the root set of a potential subtree whose leaf is a parent of z. Lines 15 to 22 construct the subtree from each valid root set, and remove invalid candidate root sets. Lines 23 to 25 handle the case where the NAT has only one leaf-induced subtree.

Algorithm 2. $SetNatByPciFV(C, \pi)$

1 init NAT T with leaf z only; type(z) = nil; init set $InNat = \emptyset$;
2 for each $x \in C$, do
3 gatetype = IsUniCausalBipart($C, \pi, \{x\}$);
4 if gatetype $\neq$ nil,
5 type(z) = gatetype; add x to T as parent of z; add x to $InNat$;
6 if $InNat = C$, return T;

7 reduce (C, π) to (C', ψ), where $C' = C \setminus InNat$;
8 $InNat' = \emptyset$; $Subsets = \emptyset$;
9 for $i = 2$ to $|C'|/2$, do
10 for each $X \subset C'$ where $|X| = i$, do
11 gatetype = IsUniCausalBipart(C', ψ, X);
12 if gatetype $\neq$ nil,
13 if type(z) = nil, type(z) = gatetype;
14 if gatetype = type(z), $Subsets = Subsets \cup \{X, C' \setminus X\}$;

15 if $Subsets \neq \emptyset$,
16 for each $X \in Subsets$,
17 if $\exists\ V \in Subsets$ such that $X \supseteq V$, remove X from Subsets;
18 for each $X \in Subsets$, do
19 reduce π to ψ over X; $R = SetNatByPciFV(X, \psi)$;
20 add R to T as a subtree induced by z;
21 $InNat'$ = union of subsets in Subsets;
22 if $InNat' = C'$, return T;

23 R = SetNatByPciFV(C', ψ);
24 add R to T as a subtree induced by z;
25 return T;

5.2. Example Traces

The following examples illustrate operations of SetNatByPciFV. Example 6 illustrates the section in lines 1 to 6. Example 7 demonstrates the section in lines 7 to 14 and the section in lines 15 to 22. Example 8 illustrates why lines 16 and 17 are necessary. Example 9 demonstrates the section in lines 23 to 25.

Example 6. *Consider input pattern π in Fig. 9 (b), whose NAT (unknown) is in (a). The for loop at line 2 iterates for each of a, b, c, d, say, in that*

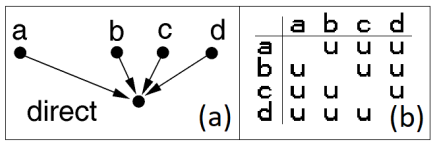

Figure 9: (a) A NAT whose leaf has root parents only. (b) Its PCI matrix.

order. For a, line 3 calls Algo. 1 to check $\pi(a, y)$ where $y \in \{b, c, d\}$. These PCI bits are in the matrix row indexed by a. Hence, Algo. 1 returns direct, leaf z is set to direct, and node a is added as parent of z. Each subsequent iteration adds another parent to z, and the correct NAT is returned in line 6.

Example 7. *Consider input pattern π in Fig. 10 (b), whose NAT (unknown) is in (a). They reproduce Fig. 5 (b) and (a). When the for loop at line 2*

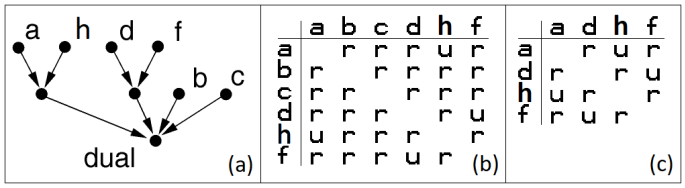

Figure 10: (a) A NAT whose leaf has both root parents and non-root parents. (b) Its PCI matrix. (c) A reduced PCI matrix.

iterates over b and c, Algo. 1 returns dual, since matrix rows indexed by b and c are uniform. Hence, b and c are added as root parents of leaf z. At line 7, C is reduced to $C' = \{a, d, h, f\}$ and π is reduced to ψ in Fig. 10 (c).

Nested for *loops in lines 9 to 14 process alternative root bipartitions of C'. Each bipartition $\{X, C' \setminus X\}$ is tested starting with $|X| = 2$ (line 9), since the root set of each subtree has at least 2 causes. As X and $C' \setminus X$ are symmetric, the upper bound of X size is $|C'|/2$ (integer division).*

Lines 11 and 12 check whether $\{X, C' \setminus X\}$ is causally uniform. Bipartition $\{\{a, d\}, \{h, f\}\}$ does not pass the test, but $\{\{a, h\}, \{d, f\}\}$ does. As the result, root sets $\{a, h\}$ and $\{d, f\}$ are added to Subsets (line 14). Note that $\{a, h\}$ and $\{d, f\}$ are added twice to Subsets in separate iterations of the inner for loop (with one copy retained). This is refined in Section 5.4.

Through the for loop in lines 18 to 20, each of $\{a, h\}$ and $\{d, f\}$ is processed by a recursive call of SetNatByPciFV. The relevant subtree in Fig. 10

(a) is extracted and added to T. The initial activation of SetNatByPciFV terminates in line 22, returning the NAT in Fig. 10 (a).

Example 8. *Consider input π in Fig. 11 (b) with unknown NAT in (a). The for loop in lines 9 to 14 adds root sets $\{a, c\}$, $\{b, d\}$, $\{h, f\}$ as well as sets such as $\{a, b, c, d\}$. Although $\{\{a, b, c, d\}, \{h, f\}\}$ is a uniform causal bipartition, $\{a, b, c, d\}$ does not correspond to a leaf-induced subtree in Fig. 11 (a). Lines 16 and 17 remove root sets such as $\{a, b, c, d\}$ from Subsets before each root set is converted to a subtree.*

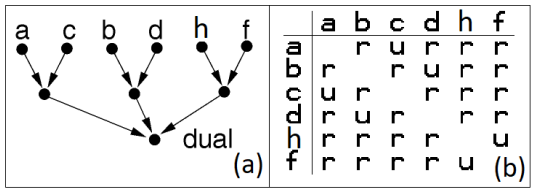

Figure 11: (a) A NAT with 3 leaf-induced subtrees. (b) Its PCI matrix.

Example 9. *Consider input π in Fig. 12 (b) with unknown NAT in (a). The for loop in lines 2 to 5 adds h as a parent of z in T, and sets type of z to dual. At line 7, C is reduced to $C' = \{a, b, c, d\}$ and π is reduced to ψ in Fig. 12 (c).*

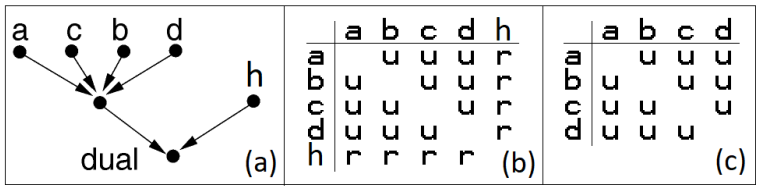

Figure 12: (a) A NAT with one leaf-induced subtree. (b) Its PCI matrix. (c) A reduced PCI matrix.

In the for loop of lines 9 to 14, although $\{\{a, b\}, \{c, d\}\}$ is recognized as a uniform causal bipartition with a direct gatetype, the cause sets are not added to Subsets, since the gatetype does not match that of z (line 14). As the result, the for loop terminates with $Subsets = \emptyset$, and processing continues at line 23. A subtree with root set $\{a, b, c, d\}$ is extracted by the recursive call in line 23, added to T, and the NAT in Fig. 12 (a) is returned.

5.3. Analysis

Completeness of SetNatByPciFV is established in Theorem 4. That is, whenever the input PCI pattern is full and valid, SetNatByPciFV will extract the correct NAT and return it.

Theorem 4. *Let Ψ be a NAT over a set C of causes and π be the PCI matrix of Ψ. Then algorithm $SetNatByPciFV(C, \pi)$ halts and returns $T = \Psi$.*

Proof: Let ρ be the leaf of Ψ. Since $|C| \geq 2$, ρ falls into the following mutually exclusive and exhaustive cases: (1) ρ has 2 or more root parents only. (2) ρ has 2 or more non-root parents and 0 or more root parent. (3) ρ has 1 non-root parent and 1 or more root parent.

In case (1), for each $x \in C$, $\{\{x\}, C \setminus \{x\}\}$ is a subtree-consistent bipartition. By Theorem 2, it is also causally uniform. Hence, each iteration of the for loop in lines 2 to 5 identifies one root parent x of ρ. $SetNatByPciFV(C, \pi)$ halts on line 6 and returns $T = \Psi$.

In case (2), if ρ has any root parent x, it is added to T as above. After line 7, C' contains root nodes in all non-trivial subtrees of Ψ induced by ρ.

By assumption, ρ has at least 2 such induced subtrees. For each subtree and its root set X, either $|X| \leq |C'|/2$ or $|X| > |C'|/2$. If $|X| \leq |C'|/2$, X is evaluated by the inner for loop (lines 10 to 14) when $i = |X|$. If $|X| > |C'|/2$, then $|C' \setminus X| < |C'|/2$, and X is evaluated by the inner for loop when $i = |C' \setminus X|$.

Since $\{X, C' \setminus X\}$ is a subtree-consistent bipartition, by Theorem 2, it is also causally uniform. Hence, X and $C' \setminus X$ are both added to $Subsets$ in line 14. Since root sets of distinct subtrees are disjoint, X cannot be removed from $Subsets$ in line 17, and the subtree over X is added to T in line 20. If $C' \setminus X$ is the root set of another subtree, the subtree is also added to T.

If Ψ has $q \geq 3$ non-trivial subtrees induced by ρ, for each subtree root set X added to $Subsets$, $C' \setminus X$ contains multiple subtree root sets and is also added to $Subsets$. Since each root set contained in $C' \setminus X$ is also added to $Subsets$, $C' \setminus X$ is removed from $Subsets$ in line 17. Hence, the for loop in lines 18 to 20 adds exactly those subtrees of Ψ to T. As the result, $SetNatByPciFV(C, \pi)$ halts on line 22 and returns $T = \Psi$.

In case (3), Ψ has a single non-trivial subtree induced by ρ. All root parents of ρ are identified by the for loop in lines 2 to 5. After line 7, C' is the root set of the subtree and ψ is over C'. Hence, C' has no other root set to pair, cannot be added to $Subsets$, and $Subsets = \emptyset$ after the for

loop in lines 9 to 14. The subtree over C' is added to T in lines 23 and 24, $SetNatByPciFV(C, \pi)$ halts on line 25, and it returns $T = \Psi$. $\square$

5.4. A Refinement of the Algorithm

Before closing the section, we present a refinement of $SetNatByPciFV$ for improved efficiency. The complexity analysis is presented in Section 7.

The for loop at line 9 searches for uniform causal bipartitions $\{X, C' \backslash X\}$, where $|X|$ is bounded at $|C'|/2$ (integer division). When $|C'|$ is odd, each bipartition so processed is unique. However, when $|C'|$ is even, which we consider below, it is no longer the case.

Suppose $C' = \{a, b, c, d\}$. The upper bound of $|X|$ is 2, and there are $C(4, 2) = 6$ distinct Xs, yielding 6 bipartitions. However, 3 of them are redundant, e.g., the bipartition from $X = \{a, b\}$ and that from $X = \{c, d\}$ are identical. This was demonstrated in Example 7. In general, when the for loop at line 9 iterates for $i = |C'|/2$, for each bipartition $\{X, C' \setminus X\}$ processed, an identical mirror bipartition $\{C' \setminus X, X\}$ is redundantly processed.

Since $C(n, n/2) = \max_k C(n, k)$, where $2 \leq k \leq n/2$, the iteration of line 9 is most expensive when $i = |C'|/2$. Hence, avoiding redundant computation over mirror bipartitions when $i = |C'|/2$ is worthwhile. By avoiding mirror bipartitions, exactly $C(n, n/2)/2$ bipartitions are produced, instead of $C(n, n/2)$ ones. For instance, when $n = 20$, $C(n, n/2) = C(20, 10) = 184,756$. Reduction to 92,378 is a nontrivial saving. A noticeable speedup of $SetNatByPciFV$ was observed after we adopt the following modification. We generate candidate bipartitions for $i = |C'|/2$ as follows:

Denote $n = |C'|$, where n is even and $n \geq 4$. Remove one cause x from C' to produce set C^- of size $n - 1$. From C^-, generate the $C(n - 1, (n/2) - 1)$ distinct subsets of size $(n/2) - 1$, and add x to each subset.

For the for loop at line 9, let each X be one subset from the above collection. The result is a collection of $C(n - 1, (n/2) - 1)$ bipartitions of C' in the form of $\{X, C' \setminus X\}$, where $|X| = n/2$. It is straightforward to show the following:

1. Each bipartition in the collection is unique.

2. The collection has cardinality $C(n, n/2)/2$. That is, $C(n, n/2)/2 = C(n - 1, (n/2) - 1)$.

6. Relation between PCI Cores and Invalid Patterns

By Theorem 4, whenever an input PCI pattern is full and valid, SetNatByPciFV extracts the correct NAT. However, when a PCI pattern is obtained from an arbitrary CPT for compression, e.g., those obtained in Example 5, there is no guarantee that it is valid. We refer to NAT structure extraction from invalid PCI patterns as being *fault tolerant*. Existing NAT extraction [19, 20] does not explicitly consider the case when input PCI patterns are invalid. Fault tolerant elicitation of NATs is considered in [24]. However, no algorithm is provided to detect invalid PCI patterns and to generate NATs accordingly. We provide the theoretical foundation of fault tolerant NAT extraction below, and present the algorithm for doing so in Section 7.

Section 4.1 introduced PCI cores that are PCI patterns without uniform causal bipartitions. We show below that PCI cores are fundamental in detecting invalid PCI patterns and in extracting NATs from them.

Definition 9. *Let π be a PCI pattern over C. A PCI pattern ψ over $X \subseteq C$ ($|X| \geq 2$) is a* **sub-pattern** *of π if, for every $x, y \in X$, $\psi(x, y) = \pi(x, y)$.*

The PCI pattern in Fig. 8 (b) is a sub-pattern of that in (a). By Def. 9, π is a trivial sub-pattern of itself. Such inclusion is useful in formulating the theorem below. A sub-pattern differs from a compatible pattern in that it is defined over a subset of causes, whereas compatible patterns are defined over the same cause set.

The PCI pattern π in Fig. 8 (a) has a uniform causal bipartition $\{\{h\}, \{a, b, c, d\}\}$, and hence is not a PCI core. Its sub-pattern ψ in (b) does not have any uniform causal partition, and hence is a PCI core. Theorem 5 below reveals a fundamental condition of invalid PCI patterns, characterized by such PCI cores.

Theorem 5. *A PCI pattern π over C ($|C| \geq 2$) is invalid iff π contains a sub-pattern ψ that is a PCI core.*

Proof: [Sufficiency] Assume that π contains a sub-pattern ψ over $S \subseteq C$ that is a PCI core. Since the smallest PCI core has 4 causes, $|S| \geq 4$. We prove by contradiction that there exists no NAT with PCI pattern π.

Suppose that a NAT T over C has PCI pattern π. By Theorem 1, T has a subtree-consistent bipartition $\{X, Y\}$ of C. Either X and Y split S (possible since $|S| \geq 4$) or they don't. We consider each case below:

(Case 1) If X and Y split S, denote $S_X = X \cap S \neq \emptyset$ and $S_Y = Y \cap S \neq \emptyset$, where $S_X \cup S_Y = S$. By Theorem 2, $\{X, Y\}$ is a uniform casual bipartition of C under π. Hence, $\{S_X, S_Y\}$ is also a uniform casual bipartition of S under ψ: a contradiction to the assumption that ψ is a PCI core.

(Case 2) If X and Y do not split S, then S is contained in one of them, say $S \subseteq X$. From $|S| \geq 4$, we have $|X| \geq 4$. Let z be the leaf of T. Since $\{X, Y\}$ is a subtree-consistent bipartition of C, Y is made of root sets of one or more subtrees induced by z (Fig. 13 (a) or (c)). Remove each such subtree from T, and denote the reduced T by T' (Fig. 13 (b)). If z is left with a single parent z' in T' (Fig. 13 (c)), remove z so that z' becomes the leaf of T' (see (d)). The resultant T' is a well-defined NAT over $X \subset C$ and $|X| \geq 4$.

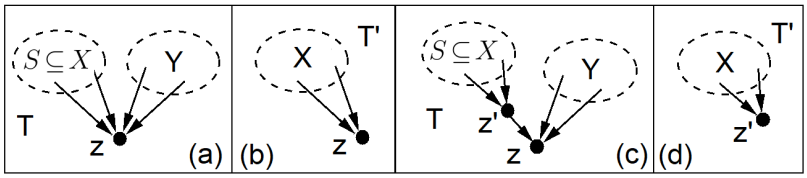

Figure 13: T' in (b) is obtained from (a). T' in (d) is obtained from (c).

Since C is finite, $|S| \geq 4$, and the reduction produces a NAT over a proper subset of causes, by processing a subtree-consistent bipartition in T' recursively, Case 1 must be true eventually, as well as the associated contradiction.

[Necessity] Suppose a PCI pattern π over C does not correspond to any NAT. We prove by contraposition that π contains a sub-pattern ψ that is a PCI core. Assume that π does not contain any PCI core. We show by induction on $|C|$ that a NAT can be constructed with PCI pattern π.

For $|C| = 2$, say, $C = \{x, y\}$, the only bipartition $\{\{x\}, \{y\}\}$ is causally uniform. Hence, a tree T with leaf z and roots x and y is a NAT over C. Assume that for $|C| = k \geq 2$, if PCI pattern π over C does not contain a PCI core, then a NAT can be constructed with pattern π.

Consider $|C| = k + 1$, where pattern π over C does not contain a PCI core. Since π is not a PCI core, there exists a uniform causal bipartition $\{X, Y\}$ of C, where $|X| \leq k$ and $|Y| \leq k$. Since $k + 1 \geq 3$, X and Y cannot both be singletons. Either exactly one of them is a singleton (Case a) or none of them is a singleton (Case b). We construct a NAT with pattern π for each case:

(Case a) Suppose that X is a singleton $\{x\}$. Since π does not contain a PCI core, neither sub-pattern ψ of π over Y does. Since $|Y| = k$, by inductive

assumption, a NAT T_Y can be constructed with PCI pattern ψ. Denote the leaf of T_Y by z.

If z is direct and the uniform causal interaction relative to bipartition $\{\{x\}, Y\}$ is u, add the root parent x to z in T_Y. The resultant tree T is a NAT with PCI pattern π (Fig. 14 (a)). Processing is similar if z is dual and interaction relative to $\{\{x\}, Y\}$ is r.

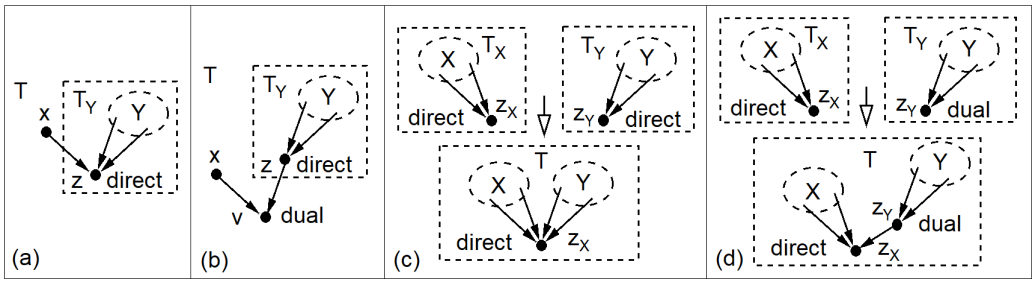

Figure 14: (a) Merge x into T_Y as parent of leaf. (b) Merge x and T_Y with new leaf v. (c) Merge T_X and T_Y at leaf z_X. (d) Merge T_X and T_Y with z_Y being parent of z_X.

If z is direct and the causal interaction relative to bipartition $\{\{x\}, Y\}$ is r, create a tree T with leaf v whose two parents are x and z. The resultant tree T is a NAT with PCI pattern π (Fig. 14 (b)). Processing is similar if z is dual and interaction relative to $\{\{x\}, Y\}$ is u.

(Case b) Suppose that none of X and Y is singleton. Let π_X (π_Y) be the sub-pattern of π over X (Y). Since π does not contain a PCI core, neither π_X nor π_Y does. Since $|X| \leq k$ ($|Y| \leq k$), by inductive assumption, a NAT T_X (T_Y) can be constructed with PCI pattern π_X (π_Y). Denote the leaf of T_X (T_Y) by z_X (z_Y).

If z_X and z_Y are both direct and the uniform causal interaction relative to bipartition $\{X, Y\}$ is u, merge T_X and T_Y by adding all parents of z_Y as parents of z_X and deleting z_Y. The resultant tree T is a NAT with pattern π (Fig. 14 (c)). Processing is similar if z_X and z_Y are both dual and interaction relative to $\{X, Y\}$ is r.

If z_X is direct, z_Y is dual, and the uniform causal interaction relative to bipartition $\{X, Y\}$ is u, merge T_X and T_Y by making z_Y a parent of z_X. The resultant tree T is a NAT with pattern π (Fig. 14 (d)). Processing is similar for other cases where types of z_X and z_Y differ. □

Theorem 5 establishes that the necessary and sufficient condition of an invalid PCI pattern π is that either π is a PCI core or a sub-pattern of π is.

7. NAT Extraction with Invalid PCI Pattern Detection

Based on Theorem 5, we extend SetNatByPciFV to detect invalid PCI patterns. As input, the extended algorithm takes a set C of causes and a full PCI matrix π over C that may be invalid. We denote it by SetNatByPciFI, where FI refers to full invalid patterns.

The main difference of SetNatByPciFI from SetNatByPciFV is additional lines 20, 24, and 26. If π is valid, SetNatByPciFI returns the respective NAT, similarly as SetNatByPciFV. However, when π is invalid, it is detected through lines 20, 24, and 26, where a PCI core is returned.

Algorithm 3. *$SetNatByPciFI(C, \pi)$*

1 init NAT T with leaf z only; type(z) = nil; init set $InNat = \emptyset$;
2 for each $x \in C$, do
3 gatetype = IsUniCausalBipart($C, \pi, \{x\}$);
4 if gatetype $\neq$ nil,
5 type(z) = gatetype; add x to T as parent of z; add x to $InNat$;
6 if $InNat = C$, return T;

7 reduce (C, π) to (C', ψ) relative to $InNat$;
8 $InNat' = \emptyset$; $Subsets = \emptyset$;
9 for $i = 2$ to $|C'|/2$, do
10 for each $X \subset C'$ where $|X| = i$, do
11 gatetype = IsUniCausalBipart(C', ψ, X);
12 if gatetype $\neq$ nil,
13 if type(z) = nil, type(z) = gatetype;
14 if gatetype = type(z), $Subsets = Subsets \cup \{X, C' \setminus X\}$;

15 if $Subsets \neq \emptyset$,
16 for each $X \in Subsets$,
17 if $\exists\ V \in Subsets$ such that $X \supseteq V$, remove X from Subsets;
18 for each $X \in Subsets$, do
19 reduce π to ψ over X; $R = SetNatByPciFI(X, \psi)$;
20 if $R = X$, return X;
21 add R to T as a subtree induced by z;
22 $InNat'$ = union of subsets in Subsets;
23 if $InNat' = C'$, return T;
24 if $InNat \cup InNat' = \emptyset$, return C;

25 R = SetNatByPciFI(C', ψ);
26 if $R = C'$, return C';
27 add R to T as a subtree induced by z;
28 return T;

We analyze important properties of the algorithm. SetNatByPciFI is *complete*, if whenever π is the PCI pattern of a NAT Ψ over C, it returns NAT $T = \Psi$. SetNatByPciFI is *sound*, if whenever π is invalid, it returns the domain of a PCI core.

Theorem 6 below establishes completeness. From this theorem, it follows that SetNatByPciFI produces the same output as SetNatByPciFV for examples in Section 5.2. After presenting Theorem 6, we illustrate, with examples, the behavior of SetNatByPciFI when the input pattern is invalid, before proving its soundness.

Theorem 6. *Let Ψ be a NAT over a set C of causes and π be the PCI matrix of Ψ. Then algorithm $SetNatByPciFI(C, \pi)$ halts and returns $T = \Psi$.*

Proof: SetNatByPciFI differs from SetNatByPciFV in recursive calls at lines 19 and 25, and in extra statements at lines 20, 24, and 26. We show that SetNatByPciFI cannot exit from lines 20, 24, and 26. Hence, assuming that π is a valid PCI pattern, the algorithm is equivalent to $SetNatByPciFV(C, \pi)$, and its completeness follows from Theorem 4.

Let ρ be the leaf of Ψ. Since $|C| \geq 2$, ρ falls into three mutually exclusive and exhaustive cases: (1) ρ has 2 or more root parents only. (2) ρ has 2 or more non-root parents and 0 or more root parent. (3) ρ has 1 non-root parent and 1 or more root parent.

In case (1), SetNatByPciFI halts in line 6, as shown in the proof of Theorem 4. Hence, lines 20, 24, and 26 are irrelevant.

In case (2), let T_X be any subtree of Ψ with root set X, that corresponds to a non-root parent of ρ. As shown in the proof of Theorem 4, after line 17, $Subsets$ contains each and every root set such as X. The for loop at line 18 iterates once for each root set, and the corresponding subtree is returned in line 19. Hence, SetNatByPciFI cannot exit from line 20. It will exit from line 23, as shown in the proof of Theorem 4, rendering lines 24 and 26 irrelevant.

In case (3), since ρ has root parents, they are added to $InNat$ in line 5. Hence, $InNat \neq \emptyset$ after line 6. As shown in the proof of Theorem 4,

$Subsets = \emptyset$ after line 14. Hence, SetNatByPciFI cannot exit from line 20. Since $InNat \neq \emptyset$ at line 24, SetNatByPciFI cannot exit from line 24.

Let T_X be the only subtree of Ψ with root set X, that corresponds to the single non-root parent of ρ. T_X will be returned in line 25, and hence SetNatByPciFI cannot exit from line 26. Instead, it halts in line 28. □

Examples below illustrate behavior of SetNatByPciFI when its input pattern is invalid.

Example 10. *Consider input pattern π in Fig. 6 (a). Since π is a PCI core, no uniform causal bipartition in the form $\{\{x\}, C \setminus \{x\}\}$ exists. After line 7, $InNat = \emptyset$, $C' = C$, and $\psi = \pi$. For the same reason, after line 14, $InNat' = \emptyset$ and $Subsets = \emptyset$. As the result, C is returned in line 24.*

Example 11. *Consider input pattern π in Fig. 8 (a). Recall that π is not a PCI core, but its sub-pattern over $\{a, b, c, d\}$ is. Since $\{\{h\}, \{a, b, c, d\}\}$ is a uniform causal bipartition, the for loop at line 2 adds h to T as parent of z, and $InNat = \{h\}$. After line 7, $C' = \{a, b, c, d\}$. Since the sub-pattern over C' is a core, after line 14, $Subsets = \emptyset$. The processing continues at line 25 and C' is returned in line 26.*

Theorem 7 below establishes soundness of SetNatByPciFI.

Theorem 7. *Let π be an invalid PCI pattern over a set C of causes. Then algorithm $SetNatByPciFI(C, \pi)$ returns $S \subseteq C$ which is the domain of a PCI core under π.*

Proof: Assume that π over C is invalid. Either π is a PCI core, or it is not. Suppose π is a PCI core. No uniform causal bipartition in the form $\{\{x\}, C \setminus \{x\}\}$ exists. Hence, $InNat = \emptyset$ at line 24. No uniform causal bipartition in the form $\{X, C \setminus X\}$ exists, where $|X| > 1$. Hence, $InNat' = \emptyset$ at line 24. It follows that SetNatByPciFI returns C at line 24.

Next, suppose π is not a PCI core. By Theorem 5, π has a sub-pattern ψ that is a PCI core, whose domain $S \subset C$ is the largest among PCI cores under π. Pattern ψ falls into the following mutually exclusive and exhaustive cases: (a) $C = S \cup X$, where S and X are disjoint, and $X \neq \emptyset$. For each $x \in X$, $\{\{x\}, C \setminus \{x\}\}$ is a uniform causal bipartition. (b) $C = S \cup X \cup Y$, where S, X, Y are disjoint, and $Y \neq \emptyset$. For each $x \in X$, $\{\{x\}, C \setminus \{x\}\}$ is a uniform causal bipartition, and X (possibly empty) includes all such x. In addition, there exists $Q \subseteq Y$ such that $\{S, Q\}$ is a uniform causal bipartition.

In case (a), each $x \in X$ is added to $InNat$ in line 5. Hence, after line 7, we have $InNat = X$ and $C' = S$. Since ψ over S is a core, the for loop in lines 9 to 14 ends up with $Subsets = \emptyset$. Subsequently, line 25 is run, which renders S to be returned in line 26.

In case (b), after line 7, we have $C' = S \cup Y$. After line 14, $Subsets$ contains S and Q. Since ψ over S is a core, S cannot be removed from $Subsets$ by line 17. S will be processed by the for loop at line 18, which renders S to be returned in line 20. Note that the PCI core ψ over S that qualifies for case (b) may not be unique. The first such S processed by the for loop at line 18 will be returned. $\square$

Next, we consider time complexity of SetNatByPciFI. When π is valid, the complexity depends on the respective NAT T. Let z be leaf of T and $|C| = n$. If every cause in C is a parent of z, only lines 1 to 6 are run, and the complexity is $O(n^2)$. If no cause is a parent of z, lines 1 to 6 are followed by lines 7 to 14. The number of alternative X (line 10) where $2 \leq |X| \leq n/2$ is $O(2^{n-1} - n - 1)$, and evaluation of each X takes $O(n^2/4)$ time. The complexity is $O(n^2\ 2^{n-1})$. This is also the complexity when π is a PCI core.

If π is valid, some causes are the parents of z, the computation time is between $O(n^2)$ and $O(n^2\ 2^{n-1})$. The same holds if π is invalid and contains a PCI core over a proper subset of C. In summary, the time complexity of SetNatByPciFI is a function of π and is between $O(n^2)$ and $O(n^2\ 2^{n-1})$. Note that since a NAT model is over a single BN family, n is not unbounded. In comparison, the complexity of SetNatByPciFI significantly reduces $O(2^{n(n-1)/2})$ by the database supported extraction.

8. NAT Extraction from Partial PCI Patterns

The input to SetNatByPciFI is a full PCI pattern. We extend SetNatByPciFI to SetNat below to allow partial input patterns. In particular, input of SetNat includes a set C of causes, a PCI pattern π over C, and a set B (possibly empty) of missing PCI bits. Set B is so defined that if $\pi(x, y)$ is a missing bit, then the unordered pair $\langle x, y \rangle \in B$. The output of SetNat is a NAT over C.

SetNat consists of 3 sections. Lines 1 to 4 collect full PCI patterns compatible with π into the set Π. In particular, at line 4, the full PCI pattern ψ is obtained from the partial pattern π by adding the missing bits according to θ. Lines 5 to 7 try to extract a NAT from the full PCI patterns. If unsuccessful, lines 8 to 14 switch some PCI bits in the PCI patterns and extract

a NAT from them. Variable $SBCount$ counts the number of PCI Bits to be switched. When π is invalid, it controls the number of bits in π to be switched.

Algorithm 4. $SetNat(C, \pi, B)$

1 *Def = set of defined bits in* π; $\Pi = \emptyset$; *SBCount = 1;*
2 *if* $B = \emptyset$, $\Pi = \{\pi\}$;
3 *else for each instantiation* θ *of missing bits in* B,
4 *complete* π *by* θ *into* ψ; $\Pi = \Pi \cup \{\psi\}$;

5 *for each* $\psi \in \Pi$, *do*
6 R = *SetNatByPciFI*(C, ψ);
7 *if* R *is a NAT, return* R;

8 *do*
9 *for each* $\psi \in \Pi$, *do*
10 *for each combination of SBCount bits in Def, do*
11 *get* τ *from* ψ *by switching these bits;*
12 R = *SetNatByPciFI*(C, τ);
13 *if* R *is a NAT, return* R;
14 *SBCount++;*

We illustrate execution of SetNat with an example.

Example 12. *Consider PCI pattern* π *over* $C = \{a, b, c, d, h\}$ *in Fig. 15 (a). It is partial since PCI bit* $\pi(c, h)$ *is missing, as indicated by underline. It is invalid since its sub-pattern over domain* $\{a, b, c, d\}$ *is a PCI core.*

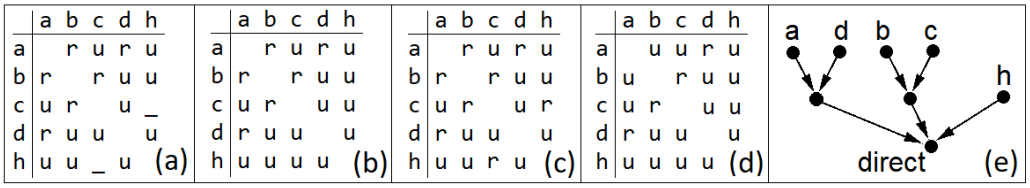

Figure 15: (a) Input PCI pattern. (b) Missing bit is filled with u. (c) Missing bit is filled with r. (d) $\pi(a, b)$ in (b) is switched. (e) Extracted NAT.

When $SetNat(C, \pi, B = \{\langle c, h\rangle\})$ starts, Def consists of 9 defined bits. After the for loop in lines 3 and 4, Π consists of ψ_0 and ψ_1, shown in Fig. 15 (b) and (c), respectively. When the for loop in lines 5 to 7 iterates for ψ_0, at line 6, R stores PCI core $\{a, b, c, d\}$. For iteration on ψ_1, at line 6, R stores PCI core $\{a, b, c, d, h\}$. Hence, computation continues to the do loop in line 8.

Suppose the for loop in line 9 starts with ψ_0 in Fig. 15 (b), and the for loop in line 10 starts with $\pi(a, b)$. This results in τ in Fig. 15 (d). Subsequently, line 12 obtains the NAT in Fig. 15 (e), it is returned in line 13, and SetNat halts.

Theorem 8 establishes the fault tolerant behavior of $SetNat$ algorithm.

Theorem 8. *Let π be a PCI pattern over C, and B be the set of missing bits in π. Then $SetNat(C, \pi, B)$ returns a NAT T with full PCI pattern ψ that satisfies the following:*

1. *If π is full and valid, then $\psi = \pi$.*
2. *If π is partial and valid, then ψ is compatible with π.*
3. *If π is invalid (full or partial), then ψ is least incompatible with π among all full valid PCI patterns over C.*

Proof: We prove for each case above. In case 1, $B = \emptyset$ and line 2 is run. The for loop in line 5 iterates exactly once on π. By Theorem 6, T is returned in line 7 and $\psi = \pi$ holds.

In case 2, $B \neq \emptyset$ and the for loop in lines 3 and 4 is run. After the loop is complete, Π contains each full PCI pattern compatible with π. Since π is valid, at least one of them is valid. The for loop in lines 5 to 7 iterates for each of them. By Theorem 6, a NAT is returned in line 7 when the first valid pattern in Π is processed, whose PCI pattern is compatible with π.

In case 3, when the for loop at line 5 starts, either $|\Pi| = 1$ (if π is full) or $|\Pi| > 1$ (if π is partial). Since π is invalid, all patterns in Π are invalid. By Theorem 7, the for loop in lines 5 to 7 cannot terminate in line 7, and the *do* loop at line 8 is executed.

Since $SBCount = 1$ in the first iteration of the *do* loop, the for loop in lines 9 to 13 processes all patterns in Π with exactly 1 switched bit. If one of the resultant pattern is valid, by Theorem 6, its generating NAT T is return

in line 13. It is incompatible with π by exactly 1 bit, which is the least. If none of patterns with 1 switched bit is valid, the first iteration of *do* loop ends with $SBCount = 2$.

Assume that the PCI pattern that is least incompatible with π differs from π (on defined bits in Def) by $k \geq 1$ bits. By Theorem 7, none of the first $k-1$ iterations of *do* loop halts successfully in line 13. By Theorem 6, the kth iteration does, and returns a NAT T in line 13, whose PCI pattern is least incompatible with π. $\square$

It is straightforward to extend SetNat and compute all NATs whose PCI patterns are least incompatible with π. We omit such extension, but analyze time complexity of SetNat, as well as the version so extended.

If π is valid, we have $|\Pi| = 2^{|B|}$. Hence, the number of executions of SetNatByPciFI (line 6) is $O(2^{|B|})$. Combining the complexity result for SetNatByPciFI (Section 7), the time complexity is $O(n^2\ 2^{|B|+n-1})$, where $n = |C|$. Since $O(2^{|B|})$ amounts to evaluating every $\psi \in \Pi$, this is also the complexity if we extend SetNat to obtain all NATs whose PCI patterns are compatible with π.

If π is invalid, let the PCI pattern that is least incompatible with π differ from π by $k \geq 1$ bits. Let β be the $SBCount$ value in a *do* iteration (line 8). The loop has k iterations for $\beta = 1, ..., k$. The for loop at line 9 has $2^{|B|}$ iterations one for each $\psi \in \Pi$. Denote $\gamma = |Def| = (n(n-1)/2) - |B|$. The for loop at line 10 has $C(\gamma, \beta)$ iterations one for each τ.

Since $C(\gamma, \beta)$ grows with β (typically β is much less than γ), the *do* iteration with $\beta = k$ dominates the computation. During that iteration, the number of executions of SetNatByPciFI is $O(2^{|B|}C(\gamma, k))$. For example, when $n = 10, k = 3, |B| = 10$, $2^{|B|}C(\gamma, k) = 6,702,080$. Hence, the complexity when π is invalid is $O(2^{|B|+n-1}\ n^2\ C(\gamma, k))$. Since every τ that is potentially valid and differs from π by k bits is counted, this is also the complexity if we extend SetNat to obtain all NATs whose PCI patterns are least incompatible with π. The above complexity significantly reduces $O(2^{n(n-1)/2})$ by the database supported extraction. For instance, with $n = 10$, we have $2^{n(n-1)/2} \approx 3.5 \times 10^{13}$.

9. Experimental Evaluation

The algorithm SetNat is empirically evaluated using randomly generated PCI patterns, ranging from valid to invalid and from full to partial, and

covering all four types of combinations. We report our experimental study on valid PCI patterns first, followed by that on invalid patterns.

9.1. NAT Extraction from Valid PCI patterns

We conducted two groups (Group 1 and 2) of experiments using valid input PCI patterns, whose number of causes has alternative values $n = 9, 15, 21$. We chose the lower value $n = 9$ for two reasons: One is because direction extraction for $n < 9$ takes less than 1 msec. Another is because $n = 9$ is almost the practical limit of database supported extraction, where building the NAT database and associated search tree took 40 hours. We choose the upper value $n = 21$ as it exceeds n values of CPTs for most real world BNs.

In Group 1, each input pattern is a randomly generated full PCI pattern. For each input pattern of a given n value, a random NAT with n causes is first generated, and its PCI pattern is derived. This guarantees that the pattern is full and valid. The group consists of 3 batches, where input patterns in each batch has the identical n value. Each batch consists of 50 PCI patterns, and hence Group 1 has a total of 150 PCI patterns.

In Group 2, each input pattern is a partial PCI pattern. Its number of missing PCI bits has alternative values $m = 1, 3$. For each input pattern of a given (n, m) value pair, a random NAT with n causes is first generated, its PCI pattern is derived, and then m PCI bits are randomly chosen and deleted. This guarantees that the pattern is partial and valid. The group consists of 6 batches, where input patterns in each batch has the identical (n, m) value pair. Each batch consists of 50 PCI patterns, and hence Group 2 has a total of 300 PCI patterns.

SetNat is run for each of the 450 PCI patterns in a ThinkPad X230. For each input pattern π, after SetNat extracts a NAT T, the PCI pattern ψ of T is derived, and identity between ψ and π is tested. For all 450 PCI patterns, identity tests are successful.

Table 2 (left) summarizes runtime (msec) of SetNat for Group 1, where 0 is shown if the runtime is less than 1 msec. Table 2 (right) summarizes that for Group 2. The relative scales of runtime in different batches are also shown in Fig. 16, where the runtime is in Log10.

Under database supported extraction, building the database and associated search tree takes 40 hours for $n = 9$, as the number of NATs for $n = 9$ is 25,637,824 [26]. The number of NATs for $n = 10$ is 564,275,648. It would take at least 880 hours to generate the NAT database and search tree.

Table 2: Summary of runtime for Group 1 (left) and Group 2 (right).

Batch	n	m	$\hat{\mu}$ msec	$\hat{\sigma}$ msec
1	9	0	0	0
2	15	0	7.2	9.0
3	21	0	711.2	692.7

Batch	n	m	$\hat{\mu}$ msec	$\hat{\sigma}$ msec
4	9	1	0	0
5	9	3	0	0
6	15	1	10.5	10.0
7	15	3	33.8	40.5
8	21	1	1260.0	1366.7
9	21	3	4995.6	5169.7

Runtime from Batch 1 shows that SetNat completes extraction for $n = 9$ instantly. For $n = 15$ and 21 (much beyond $n = 10$), it takes SetNat about 7 msec (Batch 2) and 711 msec (Batch 3), respectively. This demonstrates significant practical superiority of direct extraction.

Comparing Batch 3 with Batches 8 and 9, runtime increases are close to twice and eight times, as expected from complexity analysis. Finally, standard deviation is comparable to mean on all batches, reflecting significant fluctuation in runtime due to variation in NAT topology, which is also anticipated by complexity analysis at the end of Section 7.

9.2. NAT Extraction from Invalid PCI patterns

We conducted another two groups (Group 3 and 4) of experiments using invalid input PCI patterns, whose number of causes has alternative values $n = 9, 12, 15$. In Group 3, each input pattern is full. We require that the pattern to be invalid and to differ from any least incompatible valid pattern by r bits. Alternative r values for a given pattern are $r = 1, 2$.

For each input pattern of a given (n, r) pair, we obtain it as follows: A random NAT of n causes is first generated, its PCI pattern is derived, and r PCI bits are randomly chosen and switched. A pattern so generated appears to satisfy the requirement, but it is not always so. After changing r bits, the switched pattern may turn into another valid one. The switched pattern may also be invalid, but differs from any least incompatible valid pattern by less than r bits. To ensure uniform input patterns, after a NAT is extracted by SetNat from an input pattern of a given (n, r) pair, its PCI pattern is derived and compared with the input. If they differ by less than r bits, the input pattern is deemed to fail the requirement. In such a case, the input

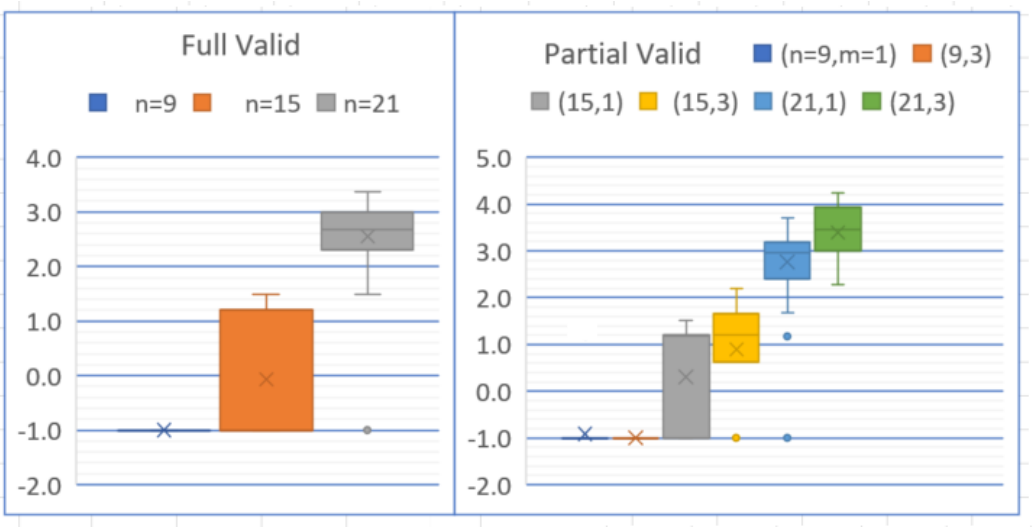

Figure 16: Summary of runtime (in Log10) for Group 1 (left) and Group 2 (right).

is replaced by another, on which SetNat is re-run. The process is repeated, until outcome of SetNat passes the test.

The group consists of 6 batches, where input patterns in each batch has the identical (n, r) pair. Each batch consists of 50 patterns, and hence Group 3 has 300 PCI patterns.

In Group 4, each input pattern is partial. Its number of missing PCI bits has alternative values $m = 1, 2$. In addition, we require that the pattern to be invalid and to differ from any least incompatible valid pattern by $r = m$ bits. For each input pattern of a given (n, m, r) pair, it is obtained as follows: A random NAT of n causes is generated with its PCI pattern derived, m PCI bits are randomly chosen and deleted, and finally r PCI bits are randomly chosen and switched. To ensure satisfaction of the above requirement, the post-extraction test and repetition used for the first group are also applied.

The group consists of 6 batches, where input patterns in each batch has the identical (n, m, r) pair. Each batch consists of 50 patterns, and hence Group 4 has 300 PCI patterns.

SetNat is run for each of the 600 PCI patterns. For each input pattern π characterized by a (n, r) pair, after SetNat extracts a NAT T, the PCI pattern ψ of T is derived, and compatibility between ψ and π is tested. For all 300 such PCI patterns, ψ and π differ by exactly r bits.

For each input pattern π characterized by a (n, m, r) pair, after SetNat

extracts a NAT T, the PCI pattern ψ of T is derived, and compatibility between ψ and π is tested, with the m missing bits in π ignored. For all 300 such PCI patterns, ψ and π differ by exactly r bits. Hence, SetNat successfully extract NATs in all possible types of scenarios.

Table 3 summarizes runtimes (msec) of SetNat for Group 3 (left) and Group 4 (right). The relative scales of runtime in different batches are also shown in Fig. 17.

Table 3: Summary of runtime for Group 3 (left) and Group 4 (right).

Bat.	n	r	$\hat{\mu}$ msec	$\hat{\sigma}$ msec
10	9	1	1.9	5.1
11	9	2	25.8	17.8
12	12	1	23.0	22.1
13	12	2	1094.6	788.5
14	15	1	499.8	441.0
15	15	2	29182.6	20639.8

Bat.	n	m	r	$\hat{\mu}$ msec	$\hat{\sigma}$ msec
16	9	1	1	2.2	5.5
17	9	2	2	66.9	55.9
18	12	1	1	44.1	41.1
19	12	2	2	3215.9	2483.6
20	15	1	1	722.3	742.9
21	15	2	2	111150.3	101381.6

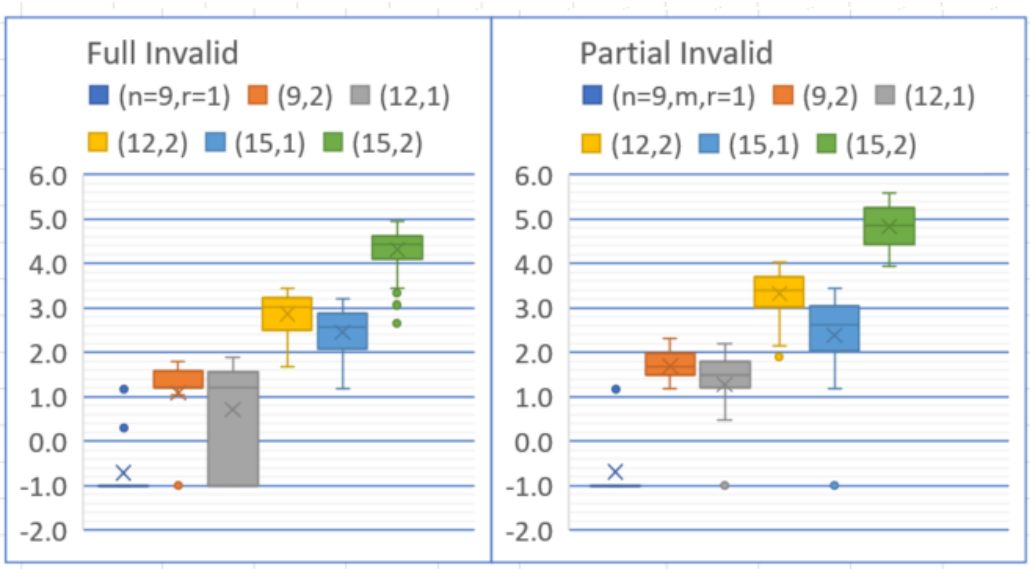

Figure 17: Summary of runtime (in Log10) for Group 3 (left) and Group 4 (right).

Comparing runtimes from Batches 16, 18, and 20, where $m = 1$, with their counterparts in Batches 10, 12, and 14, the increase of runtime is bounded

by a factor of $2^m = 2$. Similarly, comparing Batches 17, 19, and 21, where $m = 2$, with Batches 11, 13, and 15, the increase is bounded by a factor of $2^m = 4$.

On the other hand, comparing runtime where $r = 1$ with that where $r = 2$, e.g., Batches 18 versus 19, and Batches 20 versus 21, we see super-exponential growth of runtimes. This demonstrates that invalid patterns incur a much more significant computation cost than patterns with missing bits.

For $(n = 15, m = 2, r = 2)$, the mean runtime is about 2 minutes. It is practically tolerable, but further super-exponential growth would not. Hence, this result provides a practical guidance to learning NAT-modelled Bayesian networks, when tradeoff between global structure (the DAG) and local structure (the NAT) is involved.

10. Conclusion

The main contribution of this work is an algorithm suite for direct NAT extraction from possibly partial and invalid PCI patterns. The algorithm suite is founded on formal analysis of the properties of cause bipartitions. They allow NAT structure extraction in all conceivable scenarios, and therefore enable NAT modeling to be applied more effectively in compressing BN CPTs and in learning compact BN CPTs from data. Integrating these algorithms with the existing CPT compression algorithms is an immediate future work.

Our experiments showed that an incorrect PCI bit in the input pattern is much more costly than a missing PCI bit in NAT extraction. Further research will be devoted to improve efficiency of extraction from invalid PCI patterns.

Acknowledgement

Financial support from the NSERC Discovery Grant is acknowledged.

References

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

[2] M. Henrion, Some practical issues in constructing belief networks, in: L. Kanal, T. Levitt, J. Lemmer (Eds.), Uncertainty in Artificial Intelligence 3, Elsevier Science Publishers, 1989, pp. 161–173.

[3] F. Diez, Parameter adjustment in Bayes networks: The generalized noisy OR-gate, in: D. Heckerman, A. Mamdani (Eds.), Proc. 9th Conf. on Uncertainty in Artificial Intelligence, Morgan Kaufmann, 1993, pp. 99–105.

[4] C. Boutilier, N. Friedman, M. Goldszmidt, D. Koller, Context-specific independence in Bayesian networks, in: Proc. 12th Conf. on Uncertainty in Artificial Intelligence, 1996, pp. 115–123.

[5] A. Salmeron, A. Cano, S. Moral, Importance sampling in Bayesian networks using probability trees, Computational Statistics and Data Analysis 34 (2000) 387–413.

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

[7] Y. Xiang, N. Jia, Modeling causal reinforcement and undermining with noisy-AND trees, in: L. Lamontagne, M. Marchand (Eds.), Advances in Artificial Intelligence, LNAI 4013, Springer-Verlag, 2006, pp. 171–182.

[8] P. Maaskant, M. Druzdzel, An independence of causal interactions model for opposing influences, in: M. Jaeger, T. Nielsen (Eds.), Proc. 4th European Workshop on Probabilistic Graphical Models, Hirtshals, Denmark, 2008, pp. 185–192.

[9] J. Vomlel, P. Tichavsky, An approximate tensor-based inference method applied to the game of Minesweeper, in: Proc. 7th European Workshop on Probabilistic Graphical Models, Springer LNAI 8745, 2012, pp. 535–550.

[10] S. Woudenberg, L. van der Gaag, C. Rademaker, An intercausal cancellation model for Bayesian-network engineering, Inter. J. Approximate Reasoning 63 (2015) 32–47.

[11] A. Darwiche, A differential approach to inference in Bayesian networks, J. ACM 50 (3) (2003) 280–305.

[12] H. Poon, P. Domingos, Sum-product networks: A new deep architecture, in: Proc. 12th Conf. on Uncertainty in Artificial Intelligence, 2011, pp. 2551–2558.

[13] H. Zhao, M. Melibari, P. Poupart, On the relationship between sum-product networks and Bayesian networks, in: Proc. 32nd Inter. Conf. Machine Learning, 2015, pp. 116–124.

[14] Y. Xiang, Y. Jin, Multiplicative factorization of multi-valued NIN-AND tree models, in: Z. Markov, I. Russell (Eds.), Proc. 29th Inter. Florida Artificial Intelligence Research Society Conf., AAAI Press, 2016, pp. 680–685.

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

[16] Y. Xiang, Q. Jiang, NAT model based compression of Bayesian network CPTs over multi-valued variables, Computational Intelligence 34 (1) (2018) 219–240.

[17] Y. Xiang, Y. Li, J. Zhu, Towards effective elicitation of NIN-AND tree causal models, in: L. Godo, A. Pugliese (Eds.), Inter. Conf. on Scalable Uncertainty Management (SUM 2009), LNCS 5785, Springer-Verlag Berlin Heidelberg, 2009, pp. 282–296.

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

[19] Y. Xiang, Q. Liu, Compression of Bayesian networks with NIN-AND tree modeling, in: L. vander Gaag, A. Feelders (Eds.), Probabilistic Graphical Models, LNAI 8754, Springer, 2014, pp. 551–566.

[20] Y. Xiang, Q. Jiang, Compression of general Bayesian net CPTs, in: R. Khoury, C. Drummond (Eds.), Advances in Artificial Intelligence, LNAI 9673, Springer, 2016, pp. 285–297.

[21] Y. Xiang, Y. Jin, Efficient probabilistic inference in Bayesian networks with multi-valued NIN-AND tree local models, Int. J. Approximate Reasoning 87 (2017) 67–89.

[22] Y. Xiang, D. Loker, De-causalizing NAT-modeled Bayesian networks for inference efficiency, in: E. Bagheri, J. Cheung (Eds.), Canadian AI 2018, LNAI 10832, Springer, 2018, pp. 17–30.

[23] Y. Xiang, N. Jia, Modeling causal reinforcement and undermining for efficient CPT elicitation, IEEE Trans. Knowledge and Data Engineering 19 (12) (2007) 1708–1718.

[24] Y. Xiang, Acquisition and computation issues with NIN-AND tree models, in: P. Myllymaki, T. Roos, T. Jaakkola (Eds.), Proc. 5th European Workshop on Probabilistic Graphical Models, Finland, 2010, pp. 281–289.

[25] I. Beinlich, H. Suermondt, R. Chavez, G. Cooper, The alarm monitoring system: A case study with two probabilistic inference techniques for belief networks, in: Proc. 2nd European Conf. Artificial Intelligence in Medicine, 1989, pp. 247–256.

[26] Y. Xiang, J. Zhu, Y. Li, Enumerating unlabeled and root labeled trees for causal model acquisition, in: Y. Gao, N. Japkowicz (Eds.), Advances in Artificial Intelligence, LNAI 5549, Springer, 2009, pp. 158–170.