Compression of Bayesian Networks with NIN-AND Tree Modeling

Yang Xiang and Qing Liu

School of Computer Science, University of Guelph, Canada

Abstract. We propose to compress Bayesian networks (BNs), reducing the space complexity to being fully linear, in order to widely deploy in low-resource platforms, such as smart mobile devices. We present a novel method that compresses each conditional probability table (CPT) of an arbitrary binary BN into a Non-impeding noisy-And Tree (NAT) model. It achieves the above goal in the binary case. Experiments demonstrate that the accuracy is reasonably high in the general case and higher in tested real BNs. We show advantages of the method over alternatives on expressiveness, generality, space reduction and online efficiency.

Keywords: Bayesian networks, causal models, local distributions, approximation of CPTs, NIN-AND tree models

1 Introduction

The space complexity of a discrete BN of μ variables, each with up to n parents and up to κ possible values, is $O(\mu\ \kappa^n)$. Consider a BN that has at least 25 variables (among others) whose CPTs satisfy $\kappa = 6$ (perhaps due to discretizing continuous variables) and $n = 8$. With 4 bytes per probability, this BN takes a space of about $25 * 6^{8+1} * 4 \approx 1$ gigabytes, more than the full memory of iPhone 5s, making it undeployable in such a device. We propose to compress BNs so that they can be widely deployed in low-resource platforms, such as smart mobile devices and intelligent sensors. Specifically, the reported research has the ultimate goal to reduce the BN space complexity from $O(\mu\ \kappa^n)$ to $O(\mu\ \kappa\ n)$ (fully linear). An important contribution of this paper is the accomplishment of this goal for the case $\kappa = 2$.

A compression method must maintain the expressiveness of the BN reasonably well and enable more efficient inference. The combination of space and time reduction is the key to broader applications of BNs in low-resource platforms. Many existing techniques aimed at reducing parameter complexity in BNs to ease knowledge acquisition can be viewed from the perspective of space reduction. QPNs [1] represent qualitative influences by signs without using numerical parameters. They are limited in expressiveness and cannot resolve parallel influences of opposite signs, although they have been extended to alleviate the limitation, e.g., [2]. Coarsening [3] reduces κ to $\tau < \kappa$, but the space complexity is still exponential on n (i.e., $O(\mu\ \tau^n)$). Divorcing [4] cuts down the value of n,

reducing the space exponentially, but is not always applicable. The noisy-OR [5] and several causal independence models, e.g., [6, 7], reduce space complexity to being linear on n. They capture only reinforcing causal interactions and are limited in expressiveness. A NAT model [8] can also express undermining and is more expressive. Assuming that a BN family (a child plus its parents) satisfies a NAT model, it can be recovered by exhaustively evaluating alternative NATs [9]. The space of the junction tree of a BN is reduced in lazy propagation [10].

We present a novel method, denoted TDPN (Tree-Directed, Pci-based Nat search), to compress an arbitrary binary ($\kappa = 2$) BN CPT into a NAT model. It overcomes limitations of several alternatives discussed above on expressiveness, generality and efficiency. The compression process consists of an offline and an online stage. A novel search tree is constructed offline and is used online to reduce the search space of NAT structures. Numerical search for parameters is performed over the subspace to yield a compressed model.

Section 2 reviews the background on NAT models. The main idea of TDPN is described in Section 3. Its key components are elaborated in Sections 4 through 8. Section 9 reports empirical evaluations. We analyze the advantages of TDPN over alternatives in Section 10.

2 Background on NIN-AND Tree Models

We briefly introduce terminologies on NAT models and more details are found in [9]. A NAT model is defined over an effect variable e and a set $C = \{c_1, ..., c_n\}$ of uncertain causes, where $e \in \{e^-, e^+\}$, e^+ denotes $e = true$, $n \geq 2$, and $c_i \in \{c_i^-, c_i^+\}$. The probability of a causal success, where c_i caused e to occur when other causes are false, is denoted $P(e^+ \leftarrow c_i^+)$ and is referred to as a single-causal (probability). Similarly, $P(e^+ \leftarrow c_1^+, c_2^+)$ is a double-causal or multi-causal. Causal and conditional probabilities are related. If $C = \{c_1, c_2, c_3\}$, then $P(e^+|c_1^+, c_2^+, c_3^-) = P(e^+ \leftarrow c_1^+, c_2^+)$. With $X = \{c_1, c_2\} \subset C$, the multi-causal is also written as $P(e^+ \leftarrow \underline{x}^+)$. The probability of the corresponding causal failure is $P(e^- \leftarrow \underline{x}^+) = 1 - P(e^+ \leftarrow \underline{x}^+)$. When all causes are false, the effect is false, i.e., the leak probability is $P(e^+|c_1^-, c_2^-, c_3^-) = 0$.

Causes reinforce each other if they are collectively at least as effective as when only some act. They undermine each other if collectively they are less effective. A NAT expresses reinforcing and undermining between individual causes as well disjoint subsets. Fig. 1 (a) shows a NAT over $C = \{c_1, ..., c_4\}$, where black nodes are labeled by causal events. Undermining between c_1 and c_2 is encoded by a direct Non-Impeding Noisy-AND (NIN-AND) gate g_1. The gate dictates $P(e^+ \leftarrow c_1^+, c_2^+) = P(e^+ \leftarrow c_1^+)P(e^+ \leftarrow c_2^+)$. Hence, $P(e^+ \leftarrow c_1^+, c_2^+) < P(e^+ \leftarrow c_i^+)$ for $i = 1, 2$ (undermining). The similar holds for c_3 and c_4. Subsets $\{c_1, c_2\}$ and $\{c_3, c_4\}$ reinforce each other, encoded by a dual NIN-AND gate g_3. The left input event to g_3 is causal failure $e^- \leftarrow c_1^+, c_2^+$, where the white oval negates an event. Gate g_3 dictates $P(e^- \leftarrow c_1^+, c_2^+, c_3^+, c_4^+) = P(e^- \leftarrow c_1^+, c_2^+)P(e^- \leftarrow c_3^+, c_4^+)$. Hence, we have $P(e^+ \leftarrow c_1^+, c_2^+, c_3^+, c_4^+) \geq P(e^+ \leftarrow c_1^+, c_2^+)$ (reinforcing).

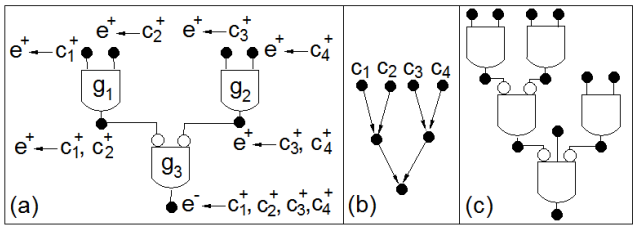

Fig. 1. (a) A NAT of 4 root events, (b) its root-labeled tree of 4 roots, and (c) a NAT of $n = 7$ causes with labels omitted.

A NAT model is a tuple $M = (e, C, T, Sp)$, where $|C| = n$, T is a NAT over e and C, and Sp is a set of n single-causals. M uniquely defines a CPT $P_M(e|C)$. Hence, a NAT modeled-CPT has a space complexity linear on n.

A NAT can be concisely represented as a root-labeled tree by omitting gates and non-root labels, and simplifying root labels, as shown in Fig. 1 (b). From the root-labeled tree and the type of leaf gate g_3, the NAT in (a) is uniquely determined. Hence, each root-labeled tree encodes two distinct NATs.

A NAT defines a pairwise causal interaction (PCI) function from pairs of causes to the set $\{u, r\}$ (undermining or reinforcing). For instance, the NAT in Fig. 1 (a) defines $pci(c_1, c_2) = u$ and $pci(c_1, c_4) = r$. Given an order of cause pairs, denoting $\{u, r\}$ by $\{0, 1\}$, a PCI pattern (bit string) is derived from the function. Using the order $(\langle c_1, c_2\rangle, \langle c_1, c_3\rangle, \langle c_1, c_4\rangle, ...)$, the PCI pattern from Fig. 1 (a) is $(u, r, r, ...)$ or $(0, 1, 1, ...)$. Each NAT has a unique PCI pattern [11].

3 Tree-Directed, PCI-Based NAT Search

We consider how to compress a target CPT P_T into a NAT model M. In other words, we approximate P_T by P_M (the CPT defined by M). The accuracy of approximation is measured by the *average Euclidean distance* (denoted by ED) below, where $K = 2^n$. The range of ED is $[0, 1]$ and matches that of probability.

$$ED(P_T, P_M) = \sqrt{\frac{1}{K}\sum_{i=1}^{K}(P_T(e^+|\underline{c}_i^+) - P_M(e^+|\underline{c}_i^+))^2}$$

When $ED(P_T, P_M) = 0$, the approximation is perfect. This is not possible for every P_T in general. Hence, M^* that minimizes ED is deemed optimal.

A related problem was solved [9] where e and C are assumed to observe an underlying NAT model. Single-causals and double-causals are estimated from observed frequencies, from which a PCI pattern is derived. The pattern is compared with that of every alternative NAT, and a best matching NAT plus the single-causals define the output model. Although the method works well, it does not solve the current problem.

First, P_T does not always yield a well-defined PCI pattern. A bit $pci(c_i, c_j) = u$ in a PCI pattern is well-defined if $P(e^+ \leftarrow c_i^+, c_j^+) < P(e^+ \leftarrow c_k^+)$ $(k = i, j)$, and $pci(c_i, c_j) = r$ is well-defined if $P(e^+ \leftarrow c_i^+, c_j^+) \geq P(e^+ \leftarrow c_k^+)$. The PCI bit $pci(c_i, c_j)$ is not well-defined when P_T yields $P(e^+ \leftarrow c_i^+) < P(e^+ \leftarrow c_i^+, c_j^+) < P(e^+ \leftarrow c_j^+)$, and nor is the corresponding PCI pattern.

Assuming an underlying NAT model, the above case may still occur due to sampling errors. It can be corrected by soft PCI identification [9]. That is, if c_i and c_j are closer to undermining than reinforcing, treat them as undermining. In particular, assign $pci(c_1, c_2) = u$ if the following holds,

$$|P(e^+ \leftarrow c_1^+, c_2^+) - \min(P(e^+ \leftarrow c_1^+), P(e^+ \leftarrow c_2^+))|$$

$$< |P(e^+ \leftarrow c_1^+, c_2^+) - \max(P(e^+ \leftarrow c_1^+), P(e^+ \leftarrow c_2^+))|,$$

and assign $pci(c_1, c_2) = r$ otherwise. This heuristics often recovers correct PCI bits.

Given an arbitrary P_T, interaction between a pair of its causes may be neither undermining nor reinforcing. Hence, softly identified PCI bits do not always lead to an accurate P_M. On the other hand, requiring well-defined PCI bits leads to a partial PCI pattern with missing bits.

Second, single-causals estimated from frequencies are directly used in the output model. This works well when the underlying model is a NAT and the NAT is recovered correctly. For an arbitrary P_T (not a NAT model in general), no matter which NAT is used, its combination with the single-causals directly from P_T is unlikely to yield an accurate P_M. We demonstrate this in Section 9.

In additional to these fundamental limitations, the method evaluates NATs exhaustively. The number of alternative NATs is $O(n!\, 0.386^{-n}\, n^{-3/2})$ [12]. Even though n is not unbounded in BN CPTs, the exhaustive NAT evaluation can be costly for larger n values.

To overcome these limitations, we propose a new method, TDPN, for approximating an arbitrary P_T with a NAT model. The basic idea is to organize seemingly incomparable NATs in a search tree based on their PCI patterns. Through the search tree, a partial pattern retrieves a small number of promising candidate NATs. Only these NATs are evaluated, which greatly improves efficiency. Rather than using the single-causals directly from P_T, they are searched during evaluation of the candidate NATs.

TDPN consists of an offline and an online phase. The offline phase is conducted before P_T is given. It enumerates root-labeled trees of n roots and constructs a PCI pattern based search tree (PST). Each non-root node of PST is assigned a value of a PCI bit. Each path from the root to a leaf is the PCI pattern of a NAT. For each n value, a PST is constructed offline and is reused online to process any P_T of n causes. Sections 4 and 5 elaborate on PST and its construction. The online phase below starts when a P_T is given.

1. Identify a well-defined partial PCI pattern Pat from P_T.
2. Use Pat to retrieve a set of candidate NATs from the PST.

3. For each candidate, search for single-causals so that the resultant NAT model M minimizes $ED(P_T, P_M)$.
4. From the above candidate NAT models, select M^* of the minimum ED as the approximate NAT model of P_T.

Steps 2 and 3 are elaborated in Sections 6 and 7.

4 PCI Pattern Based Search Trees

A PST is used to retrieve candidate NATs according to a given PCI pattern. First, we specify the PST for retrieving a single NAT from a full PCI pattern.

A NAT of n roots defines a PCI pattern of $N = C(n, 2)$ bits. A PST for a given n has $N + 1$ levels, indexed from 0 to N, with the root t placed at level 0. Each level $k > 0$ is associated with a PCI bit, and each node at the level is labeled by a bit value u or r. All leaf nodes occur at level N. A leaf z exists iff the path from t to z forms the PCI pattern of a NAT, and the NAT is assigned to z. For each node y at level $k < N$, y has a child node x iff bit values at y and x are part of a well-defined PCI pattern.

A PST for $n = 3$ is shown in Fig. 2. It has 4 levels ($N = 3$). Levels 1 to 3 are labeled by PCI bit values. The NAT assigned to each leaf is indicated by an arrow. This PST ($n = 3$) is the only balanced one. For instance, when $n = 4$, we have $N = 6$. A balanced binary tree of 7 levels has $2^N = 64$ leaf nodes, but the PST has only 52 leaf nodes (for 52 NATs). From a PCI pattern, e.g., (r, u, r), a path in the PST can be followed to retrieve a NAT, e.g., T_6. It takes $O(n^2)$ time (since N is quadratic in n).

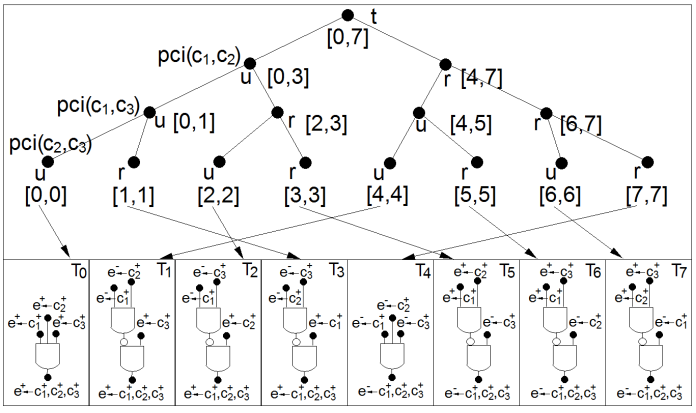

Fig. 2. A PST for $n = 3$ with the NATs at the bottom

If P_T yields a partial PCI pattern, all NATs whose PCI patterns are consistent with the partial pattern need to be retrieved. A partial pattern missing $m > 0$ bits has up to 2^m consistent NATs. Following these paths takes $O(2^m n^2)$ time. We enhance the PST below so that the retrieval from a pattern of m missing bits takes $O(n^2 - m)$ time at best (see below) and $O(2^m n^2)$ time at worst.

After the above PST is formed, index NATs assigned to leaves from left to right in ascending order. Thus, for any node x, the NATs assigned to leaves under x are consecutively indexed. These indexes can be specified by an interval $[i, j]$, which is used to label x. For leaf x, specify $i = j$.

In Fig. 2, NATs assigned to leaves are indexed from left to right at 0 through 7, e.g., T_7 is indexed at 6. If the m missing bits are located at the deepest levels, the search for NATs follows a single path to the level $N-m$ only, and the interval at the last node contains all NATs consistent with the partial pattern. It takes $O(n^2 - m)$ time. For example, the pattern $(r, u, ?)$ leads to the node with the interval $[4, 5]$ and retrieves T_1 and T_6.

5 PST Construction

Let W be the number of distinct NATs of n causes. Before building a PST for n, the W NATs are enumerated [13]. For each NAT, treat its PCI pattern as a base-2 number $R = (a_0, ..., a_j, ..., a_{N-1})$, where $a_j \in \{0, 1\} = \{u, r\}$ and its position has weight 2^{N-1-j}. We refer to PCI patterns and PCI numbers interchangeably.

Algorithm 1 *SetPST*
Input: the number n of causes; the set of W NATs;
for each NAT, compute $R = (a_0, ..., a_j, ..., a_{N-1})$;
sort PCI numbers ascending into $(R_0, ..., R_{W-1})$;
sequence corresponding NATs as $(T_0, ..., T_{W-1})$;
initialize PST to contain root t;
label t by interval $[0, W-1]$ and lower bound $B = 0$;
initialize set $Fringe = \{t\}$;
for level $k = 0$ to $N-1$,
  $Leaves = \emptyset$;
  for each $v \in Fringe$ with labels $[i, j]$ and B,
    remove v from $Fringe$;
    $B' = B + 2^{N-(k+1)}$;
    if $R_j < B'$,
      add left child x to v and insert x to Leaves;
      label x by $a_k = 0$, $[i, j]$ and B;
    else if $R_i \geq B'$,
      add right child y to v and insert y to Leaves;
      label y by $a_k = 1$, $[i, j]$ and B';
    else
      add left child x and right child y to v;
      insert x and y to Leaves;
      search index d in $(R_i, ..., R_j)$ such that $R_{d-1} < B'$ and $R_d \geq B'$;
      label x by $a_k = 0$, $[i, d-1]$ and B;
      label y by $a_k = 1$, $[d, j]$ and B';
  $Fringe = Leaves$;
return PST;

To compute the interval $[i, j]$ for each PST node x, associate x with a lower bound of PCI numbers for NATs assigned to leaves below x. The lower bound is discarded once the PST is completed.

Construction of a PST starts at level 0 and proceeds to each deeper level. Current leaf nodes are maintained in a set $Fringe$. Newly generated leaf nodes are recorded in a set $Leaves$. SetPST specifies details of construction.

Lines 12 to 23 expand a current leaf by one child or both, depending on whether each leads to a NAT. Hence, a PST is generally imbalanced. Its size is about $2W$. W has the complexity of $O(n!\ 0.386^{-n}\ n^{-3/2})$ [12] and so does SetPST. It grows faster than $n!$. Fortunately, PSTs are reusable and can be constructed offline once for all.

6 Search for Candidate NATs

At online time, after a well-defined partial PCI pattern Pat is obtained from P_T, candidate NATs whose PCI patterns are consistent with Pat are retrieved using a PST. For a given Pat, some deep PST levels may not be involved, and need not be loaded. Each unloaded deep level reduces the loading time and space by half. Given the $O(n!\ 0.386^{-n}\ n^{-3/2})$ space complexity of PST, such saving is worthwhile.

Denote the set of bits in Pat by $Bits$. The partially loaded PST includes only top levels of the full PST such that all bits in Pat are covered. Denote the PCI bit for level $i > 0$ by b_i. We assume that the PST has $K+1$ levels ($K \leq N$) and $b_K \in Bits$.

The retrieval starts from the root t. Each path consistent with Pat is followed to a node at level K, where the interval specifies candidate NATs. If $b_1 \in Bits$, one child of t is followed, according to the value of b_1 in Pat. Otherwise, both child nodes of t are followed. In general, when the loaded PST includes PCI bits absent from $Bits$, multiple nodes at a given level may be followed. They are maintained in a set $Front$. GetCandidateNAT specifies details of the retrieval.

A Pat extracted from P_T may be invalid (no defining NATs). This condition is captured in line 10, where no PST path is consistent with Pat and hence an empty candidate set is returned. To handle such cases, one option is to continue as if the error causing bit is a missing bit. It will guarantee an non-empty candidate set in the end.

From uniqueness of PCI pattern [11] and construction of PST by SetPST, it can be proven that, given any well-defined Pat, GetCandidateNAT will return exactly the set of NATs whose PCI patterns are consistent with Pat. As shown in Section 4, the time complexity is $O(2^m n^2)$.

A refinement to SetPST and GetCandidateNAT can be devised to improve efficiency for both. As mentioned before, each root-labeled tree corresponds to two NATs T_0 and T_1, differing in the type of leaf gate. Hence, their PCI patterns are bitwise complement of each other. Consider a PST built by SetPST. Let x be the leaf assigned with T_0, and y be the leaf assigned with T_1. On the path from root t to x, the PCI bit value at each level is the complement of the corresponding

bit value on the path from t to y. That is, the path from t to x is the bitwise complement of the path from t to y.

Algorithm 2 *GetCandidateNAT*
Input: a partial pattern Pat over a set Bits of PCI bits; a PST of $K+1$ levels ($K \leq N$) that covers Bits;

```
1  initialize Front = {t}, where t is the root of PST;
2  for level i = 1 to K,
3      Temp = ∅;
4      if b_i ∈ Bits,
5          retrieve value b_i = v_i in Pat;
6          for each x ∈ Front,
7              remove x from Front;
8              if x has child y whose bit value is v_i,
9                  add y to Temp;
10         if Temp = ∅, return ∅;
11     else
12         for each x ∈ Front,
13             remove x from Front;
14             add each child of x to Temp;
15     Front = Temp;
16 initialize Candidates = ∅;
17 for each x ∈ Front with interval [i, j],
18     Candidates = Candidates ∪ {T_i, ..., T_j};
19 return Candidates;
```

This observation allows to refine SetPST and to construct a PST only for NATs of direct leaf gates. The reduced PST relative to that in Fig. 2 is shown in Fig. 3. With the reduced PST, GetCandidateNAT must be run twice and the candidate set is the union of results from both runs. The 2nd run assumes the bit value complement at each node and a dual leaf gate for each NAT.

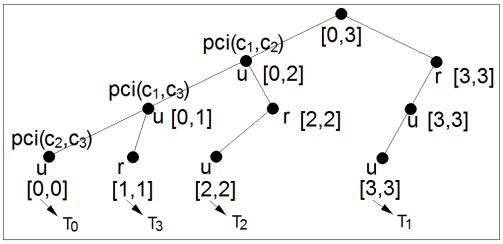

Fig. 3. A PST for NATs of direct leaf gates

This refinement reduces the time of SetPST to half and reduce the space of the PST to half. For GetCandidateNAT, the total search time in both runs is

equivalent to the time used before. Since the half-sized PST needs to be loaded only once, the load time is reduced to half.

7 Parameter Search by Steepest Descent

After candidate NATs are obtained, single-causals must be obtained to fully specify corresponding NAT models. It is possible to use the single-causals defined by the target CPT P_T. However, this option, though efficient, does not usually lead to accurate compressed NAT models, as we demonstrate in Section 9. Instead, TPDN searches single-causals for each NAT so that the resultant model M minimizes $ED(P_T, P_M)$. To this end, we apply the method of steepest descent, where a point moves along the surface of a multi-dimensional function, in the direction of the steepest descent, until the gradient is less than a threshold. Alternative techniques, e.g., simulated annealing, may be used instead.

Descent is guided by the function $ED(P_T, P_M)$ between the target CPT P_T and the CPT P_M of a candidate NAT model M. The search space has the dimension n, where each point p is a vector of n single-causals. Each single-causal is bounded in $(0, 1)$. To guard against finding a local extremum, multiple searches are randomly started for each given NAT. If they converge to a single point, the confidence increases that it is the global extremum.

8 Anytime Approximation

The TDPN presented above extracts a well-defined partial PCI pattern Pat from P_T and uses it to search for candidate NATs. As Pat is a heuristic guide to focus evaluation of alternative NATs, there is no guarantee that candidate NATs consistent with Pat include the best NAT. The number of bits in Pat directly affects the number of candidate NATs. The shorter Pat is, the larger the number of candidate NATs and the more accurate the resultant NAT model. On the other hand, evaluating more NATs is more costly, due primarily to time for steepest descent. To balance accuracy and efficiency, we present an anytime enhancement to TDPN that constrains the model approximation computation by a user specified time limit.

Let W be the number of NATs of n causes, rt be the average runtime for steepest descent per NAT, Rt be the user specified runtime, and m be the number of missing bits in Pat. The number of candidate NATs that can be processed in Rt time is about Rt/rt. The number of NATs consistent with a PCI pattern of m missing bits is about $2^{m-C(n,2)}\ W$. Hence, the largest m such that $2^{m-C(n,2)}\ W < Rt/rt$ best balances accuracy and efficiency, given the user time.

What remains is to decide the m missing bits. Suppose P_T yields $C(n, 2)-m'$ well-defined PCI bits. If $m > m'$, then $m - m'$ deepest bits in PST (Section 4) can be dropped from Pat before it is used for NAT retrieval. If $m < m'$, then $m' - m$ soft-identified PCI bits that are shallowest in PST can be added. This ensures a Pat of $C(n, 2) - m$ length, where well-defined PCI bits are used as much as possible.

9 Experimental Evaluation

To evaluate the effectiveness of TDPN, we conducted four groups of experiments. The 1st group compares approximation accuracy of TDPN with the optimal baseline, the 2nd compares TDPN with the well-known noisy-OR, and the 3rd demonstrates effectiveness of the anytime enhancement to TDPN. Five batches of target CPTs (a total of 353 CPTs) were used in these experiments. The 4th group examines the offline PST construction time.

Approximation Accuracy The 1st group mainly evaluates the approximation accuracy of TDPN. The 1st batch of 99 target CPTs were randomly generated (the most general targets), where $n = 4$ and the leak probability is 0 (Section 2).

For comparison, the approximation accuracy by exhaustive search over all NAT models is used as the baseline. We refer to the method by OP. The choice of $n = 4$ (with $W = 52$) was made because running OP for a larger n value is much more costly, e.g., $W = 472$ for $n = 5$.

For each target CPT P_T, OP computes an optimal NAT model as follows. For each distinct NAT, the best set of single-causals is obtained by steepest descent and ED of the resultant NAT model is calculated. The model of the minimum ED is selected by OP. Note that exhaustive search over all NATs by OP is the same as an earlier method [9]. On the other hand, the earlier method uses single-causals from the target CPT, while OP produces them by steepest descent. Hence, the ED resultant from OP signifies the best accuracy that NAT models can achieve.

For each P_T, a NAT model is also obtained by TDPN, as well as its ED. If TDPN selects the same NAT as OP, then difference between the two EDs is zero, and the performance of TDPN for this P_T is deemed optimal.

The 1st box in Fig. 4 (left) depicts EDs obtained by OP, where ends of whiskers represent the minimum and maximum. The sample mean is 0.1787. It shows that, under the most general condition, NAT model approximation can achieve a reasonably high accuracy, while reducing the CPT complexity from being exponential to being linear on n.

The 2nd box in Fig. 4 (left) depicts EDs by TDPN. Out of 99 target CPTs, TDPN selected the same NAT as OP in 69 of them. The sample mean of EDs is 0.1855 and is fairly close to that of OP. Fig. 4 (right) shows the runtime comparison between OP (average 7.7 sec) and TDPN (average 0.8 sec). This result shows that PCI heuristics of TDPN works well. Using 10% of time of OP, TDPN either found the optimal NAT model or one very close to the optimal.

The runtime ratio of OP versus TDPN is expected to grow more than exponentially on n (in favour of TDPN). This is due to super-exponential growth of W on n and the same growth of OP runtime. On the other hand, the number of NATs evaluated by TDPN is determined by the length of the partial PCI pattern and can be well controlled through the anytime enhancement.

Each P_T was also run with a modified version (SOFT) of TDPN. Instead of using a well-defined, partial PCI pattern, SOFT uses soft PCI identification [9]

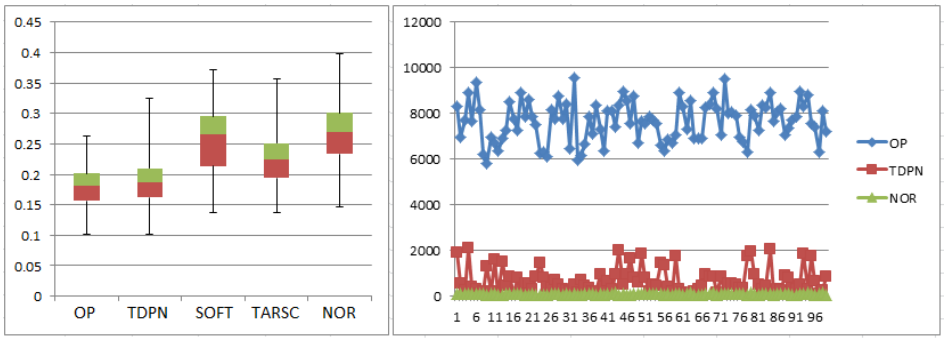

Fig. 4. Left: Boxplot of EDs of NAT models by OP, TDPN, SOFT, TARSC and NOR from the 1st batch of target CPTs. Right: Runtime by OP, TDPN and NOR in msec.

(introduced in Section 3) and a full, softly-defined PCI pattern. The 3rd box in Fig. 4 (left) depicts EDs by SOFT. Out of 99 target CPTs, it selected the same NAT as OP did in 30 of them (less than half compared to TDPN). Its sample mean of EDs is 0.2550 and much worse than TDPN, relative to the OP baseline. This comparison demonstrates the superiority of partial PCI patterns consisting of only well-defined PCI bits.

In addition, each P_T was run with an earlier method [9], referred to as TARSC. It can be viewed as a modified OP, where single-causals of the target CPT are used directly. The 4th box in Fig. 4 (left) depicts EDs by TARSC. It selected the same NAT as OP did in 54 of them and its sample mean of EDs is 0.2234. Even though TARSC searches NATs exhaustively, it performed worse than TPDN, relative to the OP baseline. This comparison demonstrates the benefit of single-causal search by steepest descent.

The results by SOFT and TARSC show that the existing technique [9] for recovering underlying NAT models does not work well for compressing general CPTs. Hence, the innovations in TDPN (well-defined PCI bits, partial PCI patterns, PST guided search, and single-causals by steepest descent) are both effective and necessary.

Comparison with Noisy-OR The 2nd group of experiments compares TDPN with noisy-OR approximation. For each P_T, the latter method (NOR) searches single-causals of a noisy-OR model by steepest descent.

The result of running NOR on the 1st batch of target CPTs is shown by the 5th box in Fig. 4 (left). For two target CPTs, NOR selected the same NAT as OP did, compared with 69 by TDPN. The sample mean of EDs by NOR is 0.2662. In comparison, TDPN is much more accurate, relative to the OP baseline.

Both NOR and TPDN were run on a 2nd batch of 100 randomly generated noisy-OR target CPTs (each CPT follows a noisy-OR model) with $n = 7$. For each P_T, TDPN returned the same NAT as NOR did without exception. The EDs by both methods are about 0.0013, and both ran about 1.5 seconds per

target CPT. Hence, TDPN performs equally well as NOR when the underlying CPT is the noisy-OR. This result confirms the generality of NAT models with noisy-OR as a special case.

NOR and TPDN were run on a 3rd batch of 99 randomly generated NAT CPTs (each CPT follows a NAT model) with $n = 7$. An example NAT is in Fig. 1 (c). The ED performance of both methods are shown in Fig. 5 (left). Since each target CPT is a NAT model, TPDN was able to express the target model accurately, and hence close to zero ED. On the other hand, NOR was unable to express undermining causal interaction between causes, and hence had a much lower approximation accuracy. The runtimes of the two methods are shown in Fig. 5 (right). TPDN not only compresses more accurately, but also runs faster than NOR. This is because TDPN selected the NAT that matched well with the target CPT, making subsequent search for single-causals converging quickly. On the other hand, causal interactions assumed by noisy-OR did not match well with the NAT modeled CPT, slowering down the search for single-causals.

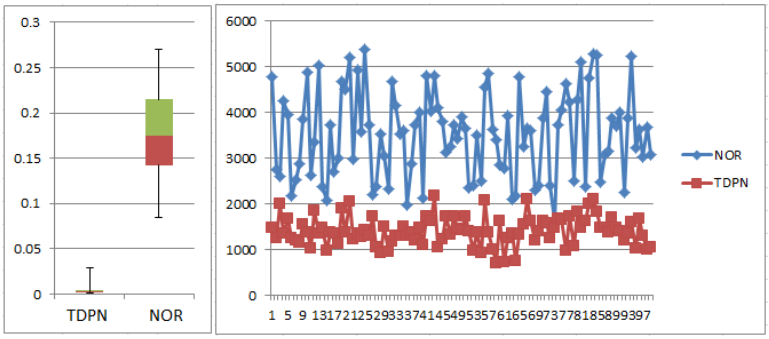

Fig. 5. Left: Euclidean distances obtained by TDPN and NOR from the 3rd batch of target CPTs. Right: Runtime by TDPN and NOR in msec.

The fourth batch was run on target CPTs from three real BNs: Alarm [14], Hailfinder [15], and HeparII [16]. Since all three BNs use multi-valued variables, they were coarsened equivalently into binary, and target CPTs were collected from all variables of 3 or more parents since a non-trivial NAT model has at least 3 causes. A total of 25 target CPTs were collected: 3 from Alarm, 6 from Hailfinder, and 16 from HeparII. Among them, 14 CPTs have $n = 3$, 7 CPTs have $n = 4$, 3 CPTs have $n = 5$, and one CPT has $n = 6$.

The ED performances of TPDN and NOR are shown in Fig. 6. For almost all target CPTs, the data points are above the $X = Y$ line, signifying smaller ED by TPDN. A Friedman test with $k = 2$ [17] resulted in the test statistic 8.8947, which is larger than the critical 0.01 χ^2 value 6.63. Therefore, TDPN compresses these real BN CPTs significantly more accurate than NOR.

The sample mean of EDs by TDPN over all 25 target CPTs is 0.1497. Compared with 0.1855 from the random target CPTs, it suggests that TDPN approx-

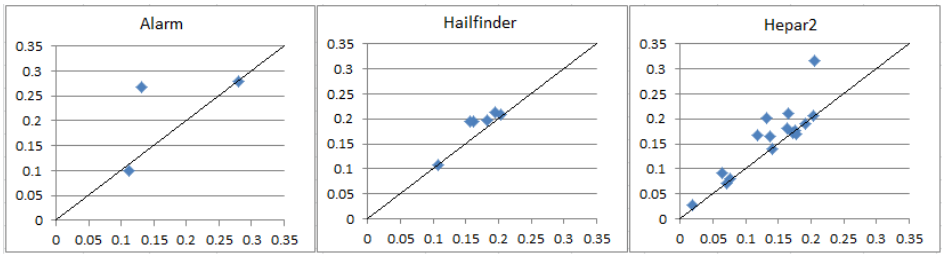

Fig. 6. Euclidean distances obtained by TDPN (X-axis) and NOR (Y-axis) from the 4th batch of target CPTs.

imates real BN CPTs more accurately than the random CPTs. In other words, real BN CPTs are closer to NAT models than random CPTs.

The runtimes for the 4th batch are shown in Fig. 7. Because a single model structure was evaluated by NOR, while multiple NATs were evaluated by TDPN, NOR runs faster. The three most time consuming CPTs in HeparII (at top of chart) took TDPN between 16 to 50 seconds due to the need to evaluate a large number of alternative NATs.

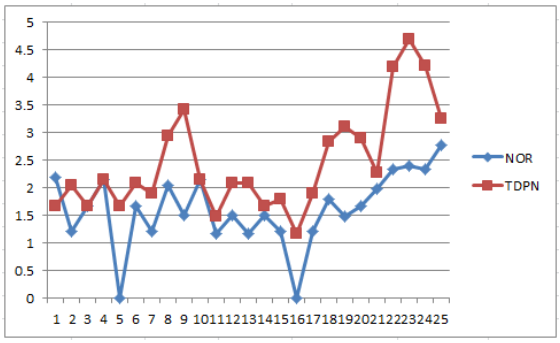

Fig. 7. Log10 runtimes from 4th batch in order of Alarm, Hailfinder and HeparII

By comparing TDPN runtimes between the 3rd batch (Fig. 5, mostly < 2 sec) and 4th (up to 50 sec), we see that TDPN is able to adapt its amount of computation according to the target CPT. When the target CPT is fairly close to a NAT model, very few alternative NATs are evaluated by steepest descent. More NATs are evaluated only when it is necessary.

Effectiveness of Anytime Enhancement The 3rd group of experiment examines effectiveness of anytime enhancement to TDPN, where the length of a PCI pattern is controlled to influence the runtime and accuracy. A 5th batch

of 30 target CPTs with $n = 5$ were randomly generated. For $n = 5$, we have $N = 10$. We chose to run TDPN for each P_T using the following m (number of missing bits in Pat) values: 2, 3, 4, 5, and 6. Fig. 8 summarizes the results.

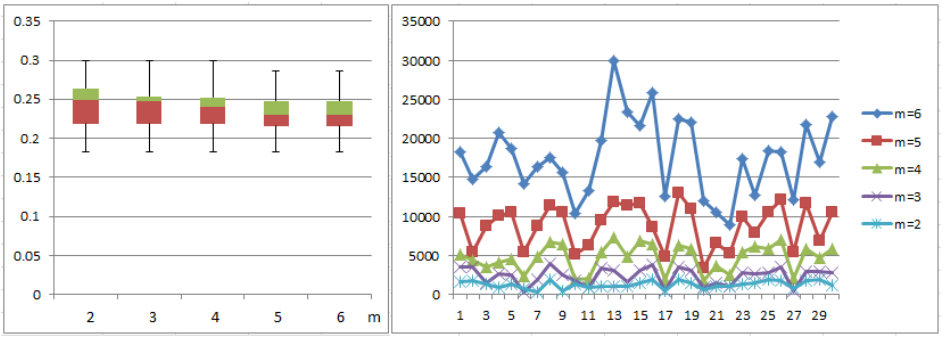

Fig. 8. Left: Euclidean distances obtained by TDPN from the 5th batch of target CPTs. Right: Runtimes in msec.

As m was increased from 2 to 6, the number of candidates NATs and runtime was increased by more than 10 times, though still less than 30 sec. For some P_T, the resultant ED was reduced by as much as 0.0609, and for some as little as 0. The result shows that, with a small m value, TPDN runs fast with a reasonably high accuracy and, with larger m values, the accuracy improves moderately with the controlled extra time.

PST Construction Time The 4th group of experiments examines the offline time for NAT enumeration and PST construction. Table 1 reports the runtime using a 2.9 GHz laptop.

Table 1. Number of distinct NATs, runtime of NAT enumeration and PST construction, in relation to n

n	6	7	8	9
# NATs	5,504	78,416	1.32×10^6	2.56×10^7
NAT enu.	110ms	0.3s	4.6s	134.5s
PST con.	828ms	16.8s	1.8h	42.9h

Table 1 shows that for $n \leq 9$, construction of PSTs are practical, since it is performed offline once for all and the resultant PSTs are reusable. Due to conditional independence encoded in BN structures, the n value for BN CPTs are not unbounded and are unlikely to be much larger than 9. Furthermore, for target CPTs with $n \geq 10$, it is possible to decompose the BN family into two or

more subsets where $n < 10$ and to apply TDPN to each. Constructing PSTs for $n \geq 10$ will then be unnecessary. We leave this to a sequel of the current work.

10 Conclusion

This paper presents three key contributions in order to compress BNs for deployment in low-resource platforms. First, we defined PCI pattern based search trees (PSTs) and developed an algorithm for their construction. PSTs organize seemingly incomparable NAT structures into a uniform searchable representation. PSTs enable exponential online complexity [9] to be shifted to offline and to be incurred once for all. Second, we presented an algorithm for searching highly promising NAT structures based on partial PCI patterns and proposed its combination with steepest descent. This combination enables efficient online compression of general target CPTs with reasonable accuracy. Third, our experimental study demonstrated several key results: (a) many general target CPTs can be approximated fairly well by NAT models, (b) NAT models lead to more accurate approximations than noisy-OR models, and (c) TDPN is superior in approximation accuracy than alternatives where either soft PCI identification or single-causals from target CPT are used.

TDPN overcomes limitations of several existing techniques. Relative to QPNs, a NAT-modeled BN retains the full range of probability. Relative to the noisy-OR, NAT-modeled BNs encode both reinforcing and undermining. Hence, TDPN leads to more expressive compression. Earlier method [9] can only recover a target CPT accurately if it is truly a NAT model. TDPN does not assume an underlying NAT model and still achieves a reasonably high accuracy. Hence, TDPN provides a general method to reduce the space complexity of BNs from $O(\mu\ \kappa^n)$ to $O(\mu\ \kappa\ n)$ for $\kappa = 2$. It can be viewed as complementing techniques such as divorcing by providing yet another alternative. Relative to coarsening whose space complexity is still exponential on n, complexity of NAT-modeled BNs is linear on n. Hence, TDPN has better space efficiency. The earlier method [9] takes an exponential online time, while online computation of TDPN is efficient ($O(2^m n^2)$ where m is a user-controllable, small integer). BNs compressed by TDPN support more efficient inference: lazy propagation in NAT-modeled BNs can be one-order of magnitude faster [18].

Extension of TDPN to multi-valued NAT models where $\kappa \geq 2$ remains the most important for further research. We have shown that TDPN can practically compress binary CPTs of up to at least $n = 9$. Although complexity of PST construction is exponential, the computation is offline, it is incurred once for all, and n is not unbounded in BNs. For target CPTs with $n \geq 10$, promising directions include decomposition and parallel PST construction. Relaxation to target CPTs of positive leak probabilities will also extend the generality of TPDN.

Acknowledgement

We thank anonymous reviewers for their helpful comments. Financial support through Discovery Grant from NSERC, Canada is acknowledged.

References

1. Wellman, M.: Graphical inference in qualitative probabilistic networks. Networks **20**(5) (1990) 687–701
2. Renooij, S., Parsons, S., van der Gaag, L.: Context-specific sign-propagation in qualitative probabilistic networks. Artificial Intelligence **140** (2002) 207–230
3. Chang, K., Fung, R.: Refinement and coarsening of Bayesian networks. In: Proc. Conf. on Uncertainty in Artificial Intelligence. (1990) 475–482
4. Jensen, F., Nielsen, T.: Bayesian Networks and Decision Graphs (2nd Ed.). Springer, New York (2007)
5. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann (1988)
6. Galan, S., Diez, F.: Modeling dynamic causal interaction with Bayesian networks: temporal noisy gates. In: Proc. 2nd Inter. Workshop on Causal Networks. (2000) 1–5
7. Lemmer, J., Gossink, D.: Recursive noisy OR - a rule for estimating complex probabilistic interactions. IEEE Trans. on System, Man and Cybernetics, Part B **34**(6) (Dec 2004) 2252–2261
8. Xiang, Y., Jia, N.: Modeling causal reinforcement and undermining for efficient CPT elicitation. IEEE Trans. Knowledge and Data Engineering **19**(12) (Dec 2007) 1708–1718
9. Xiang, Y., Truong, M., Zhu, J., Stanley, D., Nonnecke, B.: Indirect elicitation of NIN-AND trees in causal model acquisition. In Benferhat, S., Grant, J., eds.: Inter. Conf. on Scalable Uncertainty Management (SUM 2011), LNCS 6929. Springer-Verlag Berlin Heidelberg (2011) 261–274
10. Madsen, A., Jensen, F.: Lazy propagation: A junction tree inference algorithm based on lazy evaluation. Artificial Intelligence **113**(1-2) (1999) 203–245
11. Xiang, Y., Li, Y., Zhu, J.: Towards effective elicitation of NIN-AND tree causal models. In Godo, L., Pugliese, A., eds.: Inter. Conf. on Scalable Uncertainty Management (SUM 2009), LNCS 5785. Springer-Verlag Berlin Heidelberg (2009) 282–296
12. Schroeder, E.: Vier combinatorische probleme. Z. f. Math. Phys. **15** (1870) 361–376
13. Xiang, Y., Zhu, J., Li, Y.: Enumerating unlabeled and root labeled trees for causal model acquisition. In Gao, Y., Japkowicz, N., eds.: Advances in Artificial Intelligence, LNAI 5549. Springer (2009) 158–170
14. Beinlich, I., Suermondt, H., Chavez, R., Cooper, G.: 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) 247–256
15. Abramson, B., Brown, J., Edwards, W., Murphy, A., Winkler, R.L.: Hailfinder: A Bayesian system for forecasting severe weather. Inter. J. Forecasting **12** (1996) 57–71
16. Onisko, A.: Probabilistic Causal Models in Medicine: Application to Diagnosis of Liver Disorders. PhD thesis, Institute of Biocybernetics and Biomedical Engineering, Polish Academy of Science (2003)
17. Sheskin, D.: Handbook of Parametric and Nonparametric Statistical Procedures (3rd ed.). Chapman & Hall/CRC (2004)
18. Xiang, Y.: Bayesian network inference with NIN-AND tree models. In Cano, A., Gomez-Olmedo, M., Nielsen, T., eds.: Proc. 6th European Workshop on Probabilistic Graphical Models, Granada (2012) 363–370