Enumerating Unlabeled and Root Labeled Trees for Causal Model Acquisition

Yang Xiang, Zoe Jingyu Zhu, and Yu Li

University of Guelph, Canada

Abstract. To specify a Bayes net (BN), a conditional probability table (CPT), often of an effect conditioned on its n causes, needs to be assessed for each node. It generally has the complexity exponential on n. The non-impeding noisy-AND (NIN-AND) tree is a recently developed causal model that reduces the complexity to linear, while modeling both reinforcing and undermining interactions among causes. Acquisition of an NIN-AND tree model involves elicitation of a linear number of probability parameters and a tree structure. Instead of asking the human expert to describe the structure from scratch, in this work, we develop a two-step menu selection technique that aids structure acquisition.

1 Introduction

To specify a BN, a CPT needs to be assessed for each non-root node. It is often advantageous to construct BNs along the causal direction, in which case a CPT is the distribution of an effect conditioned on its n causes. In general, assessment of a CPT has the complexity exponential on n. Noisy-OR [7] is the most well known causal model that reduces this complexity to linear. A number of extensions have also been proposed such as [4, 3, 5]. However, noisy-OR, as well as related causal models, can only represent reinforcing interactions among causes [9]. The NIN-AND tree [9] is a recently proposed causal model. As noisy-OR, the number of probability parameters to be elicited is linear on n. Furthermore, it allows modeling of both reinforcing and undermining interactions among causes. The structure of causal interactions is encoded as a tree of a linear number of nodes which must be elicited in addition.

An NIN-AND tree can be acquired by asking the expert to describe the tree structure from scratch. When the number of causes is more than 3, describing the target NIN-AND tree accurately may be challenging. In this work, we develop a menu selection technique that aids the structure acquisition. We propose a compact representation through which an NIN-AND tree structure is depicted as a partially labeled tree of multiple roots and a single leaf, called a *root-labeled tree.* As there are too many root-labeled trees for a given number of causes, we divide the menu selection into 2 steps. In the first step, the human expert is presented with an enumeration of unlabeled trees for the given number of causes, and is asked to select one. In the second step, the expert is presented with an enumeration of root-labeled trees that are isomorphic to the selected unlabeled tree. The two-step menu selection reduces significantly the total number of alternatives to be presented. It lowers the overall cognitive load to the expert and is expected to improve the accuracy and efficiency of NIN-AND tree model acquisition.

To implement the two-step menu selection, a proper set of tree structures must be enumerated at each step. For the first step, we draw from a technique from phylogenetics [2] for counting *evolutionary tree shapes*, which are unlabeled trees of a single root and multiple leaves. We extend the technique for counting to an algorithm for enumeration (generation) of unlabeled trees of multiple roots and a single leaf. For the second step, we develop a new algorithm to enumerate root-labeled trees isomorphic to a given unlabeled tree.

2 Background

This section is mostly based on [9]. An *uncertain cause* is a cause that can produce an effect but does not always do so. Denote a set of binary cause variables as $X = \{c_1, ..., c_n\}$ and their effect variable (binary) as e. For each c_i, denote $c_i = true$ by c_i^+ and $c_i = false$ by c_i^-. Similarly, denote $e = true$ by e^+ and $e = false$ by e^-.

A *causal event* refers to an event that a cause c_i caused an effect e to occur successfully. Denote this causal event by $e^+ \leftarrow c_i^+$ and its probability by $P(e^+ \leftarrow c_i^+)$. The causal failure event, where e is false when c_i is true, is denoted as $e^+ \not\leftarrow c_i^+$. Denote the causal event that a set $X = \{c_1, ..., c_n\}$ of causes caused e by $e^+ \leftarrow c_1^+, ..., c_n^+$ or $e^+ \leftarrow \underline{x}^+$. Denote the set of *all causes* of e by C. The CPT $P(e|C)$ relates to probabilities of causal events as follows: If $C = \{c_1, c_2, c_3\}$, then $P(e^+|c_1^+, c_2^-, c_3^+) = P(e^+ \leftarrow c_1^+, c_3^+)$.

Causes reinforce each other if collectively they are at least as effective in causing the effect as some acting by themselves. If collectively they are less effective, then they undermine each other. The following defines the 2 types of causal interactions generally.

Definition 1 *Let $R = \{W_1, W_2, ...\}$ be a partition of a set X of causes, $R' \subset R$, and $Y = \cup_{W_i \in R'} W_i$. Sets of causes in R reinforce each other, iff $\forall R'$ $P(e^+ \leftarrow \underline{y}^+) \leq P(e^+ \leftarrow \underline{x}^+)$. Sets of causes in R undermine each other, iff $\forall R'$ $P(e^+ \leftarrow \underline{y}^+) > P(e^+ \leftarrow \underline{x}^+)$.*

Disjoint sets of causes $W_1, ..., W_m$ satisfy *failure conjunction* iff

$$(e^+ \not\leftarrow \underline{w}_1^+, ..., \underline{w}_m^+) = (e^+ \not\leftarrow \underline{w}_1^+) \wedge ... \wedge (e^+ \not\leftarrow \underline{w}_m^+).$$

That is, collective failure is attributed to individual failures. They also satisfy *failure independence* iff $P((e^+ \not\leftarrow \underline{w}_1^+) \wedge ... \wedge (e^+ \not\leftarrow \underline{w}_m^+)) = P(e^+ \not\leftarrow \underline{w}_1^+) \; ... \; P(e^+ \not\leftarrow \underline{w}_m^+)$. Disjoint sets of causes $W_1, ..., W_m$ satisfy *success conjunction* iff $e^+ \leftarrow \underline{w}_1^+, ..., \underline{w}_m^+ = (e^+ \leftarrow \underline{w}_1^+) \wedge ... \wedge (e^+ \leftarrow \underline{w}_m^+)$. That is, collective success requires individual effectiveness. They also satisfy *success independence* iff $P((e^+ \leftarrow \underline{w}_1^+) \wedge ... \wedge (e^+ \leftarrow \underline{w}_m^+)) = P(e^+ \leftarrow \underline{w}_1^+) \; ... \; P(e^+ \leftarrow \underline{w}_m^+)$. It has been shown that causes are reinforcing when they satisfy failure conjunction and independence, and they are undermining when they satisfy success conjunction and independence. Hence, undermining can be modeled by a direct NIN-AND gate (Fig. 1, left), and reinforcement by a dual NIN-AND gate (middle).

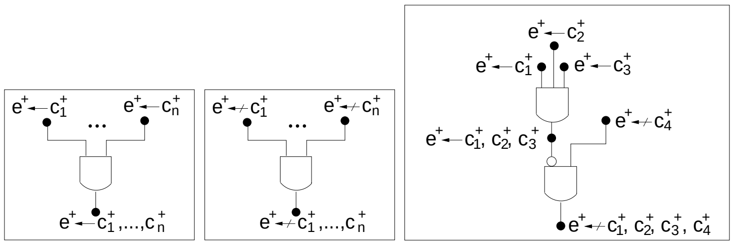
$e^+ \leftarrow c_1^+$
$e^+ \leftarrow c_n^+$
...
$e^+ \leftarrow c_1^+,...,c_n^+$
$e^+ \not\leftarrow c_1^+$
$e^+ \not\leftarrow c_n^+$
...
$e^+ \not\leftarrow c_1^+,...,c_n^+$
$e^+ \leftarrow c_2^+$
$e^+ \leftarrow c_1^+$
$e^+ \leftarrow c_3^+$
$e^+ \not\leftarrow c_4^+$
$e^+ \leftarrow c_1^+, c_2^+, c_3^+$
$e^+ \not\leftarrow c_1^+, c_2^+, c_3^+, c_4^+$

Fig. 1. (Left) Direct NIN-AND gate. (Middle) Dual NIN-AND gate. (Right) The structure of a NIN-AND tree causal model.

As per Def. 1, a set of causes can be reinforcing (undermining), but the set is undermining (reinforcing) with another set. Such causal interaction can be modeled by a NIN-AND tree. As shown in Fig. 1 (right), causes c_1 through c_3 are undermining, and they are collectively reinforcing c_4. The following defines NIN-AND tree models in general:

Definition 2 *The structure of an NIN-AND tree is a directed tree for effect e and a set $X = \{c_1, ..., c_n\}$ of occurring causes.*

1. *There are 2 types of nodes. An* `event` *node (a black oval) has an in-degree ≤ 1 and an out-degree ≤ 1. A* `gate` *node (a NIN-AND gate) has an in-degree ≥ 2 and an out-degree 1.*
2. *There are 2 types of links, each connecting an event and a gate along input-to-output direction of gates. A* `forward` *link (a line) is implicitly directed. A* `negation` *link (with a white oval at one end) is explicitly directed.*
3. *Each terminal node is an event labeled by a causal event $e^+ \leftarrow \underline{y}^+$ or $e^+ \not\leftarrow \underline{y}^+$. There is a single* `leaf` *(no child) with $\underline{y}^+ = \underline{x}^+$, and the gate it connects to is the* `leaf gate`*. For each* `root` *(no parent; indexed by i), $\underline{y}_i^+ \subset \underline{x}^+$, $\underline{y}_j^+ \cap \underline{y}_k^+ = \emptyset$ for $j \neq k$, and $\bigcup_i \underline{y}_i^+ = \underline{x}^+$.*
4. *Inputs to a gate g are in one of 2 cases:*
   (a) *Each is either connected by a forward link to a node labeled $e^+ \leftarrow \underline{y}^+$, or by a negation link to a node labeled $e^+ \not\leftarrow \underline{y}^+$. The output of g is connected by a forward link to a node labeled $e^+ \leftarrow \cup_i \underline{y}_i^+$.*
   (b) *Each is either connected by a forward link to a node labeled $e^+ \not\leftarrow \underline{y}^+$, or by a negation link to a node labeled $e^+ \leftarrow \underline{y}^+$. The output of g is connected by a forward link to a node labeled $e^+ \not\leftarrow \cup_i \underline{y}_i^+$.*

An NIN-AND tree model for effect e and its causes C can be obtained by eliciting its structure (with $|C|$ roots) and $|C|$ single-cause probabilities $P(e^+ \leftarrow c_i^+)$ one for each root event in the structure. The CPT $P(e|C)$ can then be derived using the model.

By default, each root event in a NIN-AND tree is a single-cause event, and all causal interactions satisfy failure (or success) conjunction and independence. If a subset of causes do not satisfy these assumptions, suitable multi-cause probabilities $P(e^+ \leftarrow \underline{x}^+)$, where $X \subset C$, can be directly elicited and incorporated into the NIN-AND tree model. The default is assumed in this paper.

Some additional notations used in the paper are introduced below. The number of combinations of n objects taken k at a time without repetition is denoted $C(n, k)$. We assume that the n objects are integers 0 through $n-1$. Each combination is referred to as a k-combination of n objects. We assume $n < 10$ and k-combinations can be stored in an array, say, cb. Thus we refer to the i'th k-combination by $cb[i]$. We denote the number of k-combinations of n objects with repetition by $C'(n, k)$. Note $C'(n, k) = C(n + k - 1, k)$.

A *partition* of a positive integer n is a set of positive integers which sum to n. Each integer in the set is a *part*. A *base* of m *units* is a tuple of m positive integers $s = (s_{m-1}, ..., s_0)$. A *mixed base number* associated with a base s is a tuple $x = (x_{m-1}, ..., x_0)$ where $0 \leq x_i < s_i$. Each x_i $(i > 0)$ has the *weight* $w_i = s_{i-1} * ... * s_0$ and the weight of x_0 is 1. Each integer k in the range 0 through $s_{m-1} * ... * s_0 - 1$ can be represented as a mixed base number x such that $k = \sum_{i=0}^{m-1} x_i * w_i$. For base $b = (3, 2)$, integers 0 through 5 can be represented in that order as $(0, 0), (0, 1), (1, 0), (1, 1), (2, 0), (2, 1)$. We denote an array z of k elements by $z[0..k-1]$.

3 Compact Representation of NIN-AND Tree Structure

NIN-AND tree models allow a CPT of generally exponential complexity to be obtained by eliciting a tree structure and a linear number of probabilities of single-cause events. [9] relies on the human expert to describe the tree topology. When the number of causes is more than 3, accurate description may be cognitively demanding. As we will show, for 4 causes, there are 52 alternative NIN-AND tree structures.

A better alternative is to show the expert a menu of all possible structures from which one can be selected. To construct the menu, we need to enumerate alternative structures. To facilitate the enumeration, we seek a compact representation of NIN-AND tree structure.

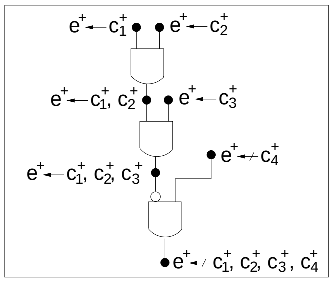
$e^+ \leftarrow c_1^+$
$e^+ \leftarrow c_2^+$
$e^+ \leftarrow c_1^+, c_2^+$
$e^+ \leftarrow c_3^+$
$e^+ \not\leftarrow c_4^+$
$e^+ \leftarrow c_1^+, c_2^+, c_3^+$
$e^+ \not\leftarrow c_1^+, c_2^+, c_3^+, c_4^+$

Fig. 2. An NIN-AND tree structure corresponding to the same causal model as that in Fig. 1 (right).

First, we observe that the NIN-AND tree structure in Fig. 2 and that in Fig. 1 (right) correspond to the same causal model. In both structures, causes c_1 through c_3 are undermining, and they are collectively reinforcing c_4. We regard the structure in Fig. 2 as superfluous and that in Fig. 1 (right) as minimal, according to Def. 3 below. Our first step towards a compact structure representation is to adopt minimal structures.

Definition 3 *Let τ be an NIN-AND tree structure. If τ contains a gate t that outputs to another gate g of the same type (direct or dual), delete t and connect its inputs to g. If such deletion is possible, τ is* `superfluous`*. Apply such deletions until no longer possible. The resultant NIN-AND tree structure is* `minimal`*.*

In a minimal NIN-AND tree structure, if the leaf gate g is a direct gate, then all gates outputting to g are dual, and their inputs are all from direct gates. That is, from the leaf towards root nodes, gates alternate in types.

This alternation implies that, for every minimal NIN-AND tree structure τ with a direct leaf gate, there exists a minimal NIN-AND tree τ' replacing each gate in τ with its opposite type, and vice versa. Therefore, if we know how to enumerate NIN-AND tree structures for a given number of causes and with a direct leaf gate, we also know how to enumerate structures with a dual leaf gate. Hence, our second step towards a compact structure representation is to focus only on minimal structures with direct leaf gates.

In a minimal structure with a direct leaf gate, types of all other gates are uniquely determined. If all root events are specified (i.e., root nodes labeled), then the causal event for every non-root node is uniquely determined. Note, however, specification of root events is partially constrained. For example, in Fig. 1 (right), since the leaf gate is dual, every root event connected to the top gate must be a causal success (rather than failure). Hence, our third step towards a compact structure representation is to omit labels for all non-root nodes.

In an NIN-AND tree structure, each gate node is connected to its unique output event. Hence, out final step towards a compact structure representation is to omit each gate node and connect its input event nodes to its output node.

As the result, our compact representation of the structure of each NIN-AND tree model is a minimal tree consisting of event nodes with only root nodes labeled. Its (implicit) leaf gate is a direct gate. Fig. 3 (a) shows the resultant representation for the NIN-AND tree in Fig. 1 (right). Following the convention in Def. 2, all links are implicitly directed (downwards away from labeled nodes). We refer to the graphical representation as *root-labeled tree.* Note that left-right order of parents makes no difference. For instance, Fig. 3 (b) is the same root-labeled tree as (a), whereas (c) is a different root-labeled tree from (a).

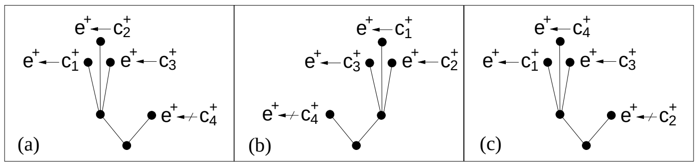

Fig. 3. Compact representations of NIN-AND tree structures.

Theorem 1 establishes the relation between enumeration of root-labeled trees and enumeration of NIN-AND tree model structures.

Theorem 1 *Let Ψ be the collection of NIN-AND tree models for n causes and Ψ' be the collection of root-labeled trees with n roots. The following hold:*

1. $|\Psi| = 2\ |\Psi'|$.
2. *For each NIN-AND tree model in Ψ, a unique tree in Ψ' can be obtained by minimizing the structure of the model, removing its gate nodes, and removing labels of non-root nodes.*
3. *For each tree in Ψ', minimal structures of 2 NIN-AND tree models in Ψ can be obtained by adding gate nodes to the tree, labeling the leaf gate as direct or dual, and labeling other non-root nodes accordingly.*

4 Enumeration of Unlabeled NIN-AND Tree Structures

Due to Theorem 1, we can enumerate NIN-AND tree model structures for n causes by enumerating root-labeled trees with n roots. The list of structures can then be presented to the expert for menu selection. However, when $n > 3$, there are too many root-labeled trees (and twice as many minimal model structures). With 4 roots, there are 26 root-labeled trees corresponding to 52 minimal NIN-AND tree model structures. With 5 roots, the numbers are 236 and 472.

To reduce the cognitive load to the expert, we divide the menu selection into 2 steps. In the first step, only unlabeled trees will be presented. With 4 roots, the menu size is 5. After the expert selects an unlabeled tree, either root-labeled trees or minimal NIN-AND tree structures corresponding to the choice will be presented for the second selection. For instance, if the unlabeled tree corresponding to Fig. 3 (a) is selected in the first step, a total of 4 root-labeled trees (or 8 NIN-AND tree structures) will be presented for second selection: at most 5+8 = 13 items (rather than 52) presented in both steps. The two-step selection can be repeated until the expert is satisfied with the final selection. The advantage is the much reduced total number of menu items presented.

To realize the two-step menu selection, we first need to enumerate unlabeled trees (of a single leaf) given the number of roots. Many methods of tree enumeration in the literature, e.g., [1, 8, 6], do not address this problem. One exception is a technique from phylogenetics [2] for counting *evolutionary tree shapes*. The technique is closely related to our task but needs to be extended before being applicable:

First, [2] considers unlabeled directed trees with a single root and multiple leaves (called *tips*). Those trees of a given number of tips are counted. What we need to consider are unlabeled directed trees with a single leaf and multiple roots. This difference can be easily dealt with, which amounts to reversal of directions for all links. Second, [2] represents these trees with a format incompatible with the standard notion of graph, where some link is connected to a single node instead of two (see, for example, Fig. 3.5 in [2]). Third, [2] considers only counting of these trees, while we need to enumerate (generate) them.

Nevertheless, the idea in [2] for counting so called *rooted multifurcating tree shapes* is an elegant one. Algorithm EnumerateUnlabeledTree(n) extends it to enumerate unlabeled trees with a single leaf and a given number n of roots.

Algorithm 1 *EnumerateUnlabeledTree(n)*
Input: the number of roots n.

```
1  initialize list $T_1$ to include a single unlabeled tree of one leaf and one root;
2  for $i = 2$ to $n$,
3     enumerate partitions of $i$ with at least 2 parts;
4     for each partition $ptn$ of $t$ distinct parts $(p_0, ..., p_{t-1})$,
5        create arrays $z[0..t-1]$, $s[0..t-1]$ and $cbr[0..t-1][][]$;
6        for $j = 0$ to $t-1$,
7           $z[j]$ = number of occurrences of $p_j$ in $ptn$;
8           $m = |T_{p_j}|$;
9           $s[j] = C'(m, z[j])$;
10          $cbr[j]$ stores $z[j]$-combinations of $m$ objects with repetition;
11       $count = 1$;
12       for $j = 0$ to $t-1$, $count = count * s[j]$;
13       for $q = 0$ to $count - 1$,
14          convert $q$ to a mixed base number $b[0..t-1]$ using base $s[0..t-1]$;
15          subtree set $S = \emptyset$;
16          for $j = 0$ to $t-1$,
17             if $z[j] = 1$, add tree $T_{p_j}[b[j]]$ to $S$;
18             else get combination $cb = cbr[j][b[j]]$;
19                   for each number $x$ in $cb$, add tree $T_{p_j}[x]$ to $S$;
20          $T' = MergeUnlabeledTree(S)$;
21          add $T'$ to $T_i$;
22 return $T_n$;
```

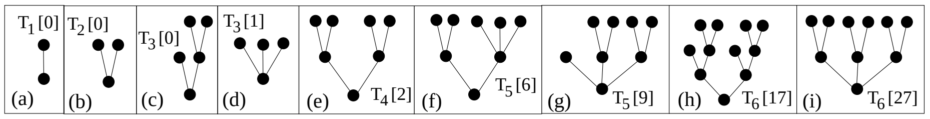

Fig. 4. (a), (b) The only unlabeled tree of one and 2 roots, respectively. (c), (d) The only trees of 3 roots. (e) A tree of 4 roots. (f), (g) Trees of 5 roots. (h), (i) Trees of 6 roots. Links are implicitly directed downwards.

Line 1 creates $T_1 = (T_1[0])$ with $T_1[0]$ shown in Fig. 4 (a). The first iteration ($i = 2$) of *for* loop started at line 2 creates $T_2 = (T_2[0])$ with $T_2[0]$ shown in Fig. 4 (b). The next iteration ($i = 3$) creates $T_3 = (T_3[0], T_3[1])$ shown in (c) and (d). The loop continues to build each tree list of an increasing number of roots, until the list for $i = n$ roots is obtained.

In line 3, the set of partitions of i with 2 or more parts is obtained. For instance, for $i = 4$, the set is $\{\{3, 1\}, \{2, 2\}, \{2, 1, 1\}, \{1, 1, 1, 1\}\}$. For $i = 5$, the set is $\{\{4, 1\}, \{3, 2\}, \{3, 1, 1\}, \{2, 2, 1\}, \{2, 1, 1, 1\}, \{1, 1, 1, 1, 1\}\}$. Each partition signifies how a tree of 4 roots can be assembled from subtrees. For example, $\{3, 2\}$ means that a tree of 5 roots can be assembled by merging a subtree of 3 roots (an element of T_3) with a subtree of 2 root (an element of T_2). The *for* loop started at line 4 iterates through each partition to enumerate the corresponding trees. Lines 5 through 12 count the number of trees from the given partition and specify indexes of subtrees to be merged in $cbr[]$. For each new tree, lines 13 through 21 retrieve relevant subtrees and merge them.

If a part p appears in a partition once (counted by $z[j]$ in line 7), then list T_p contributes one subtree to each new tree. This can be done in $m = |T_p|$ ways (line 8). It is counted by $s[j]$ in line 9, where $C'(m,1) = C(m,1) = m$. The $cbr[j]$ in line 10 will be $(cbr[j][0], ..., cbr[j][m-1])$ where each $cbr[j][k] = (k)$ is a 1-combination that indexes the m elements in T_p. For example, consider the iteration of the *for* loop started at line 4 with $ptn = \{3,2\}$ and hence $(p_0, p_1) = (3,2)$. After line 10, we have occurrence counting $(z[0], z[1]) = (1,1)$. It is used to produce $s[0] = C'(2,1) = 2$, $s[1] = C'(1,1) = 1$, and combinations $(cbr[0][0], cbr[0][1]) = ((0),(1))$ and $cbr[1][0] = (0)$.

The total number of distinct trees due to the partition is counted by the product of $s[j]$. For instance, line 12 produces $count = 2$, which says that there are 2 unlabeled trees in T_5 due to partition $\{3,2\}$.

If a part p appears in a partition $z[j] > 1$ times, then list T_p contributes $z[j]$ subtrees to each new tree. This can be done in $C'(m, z[j])$ ways (line 8). It is counted by $s[j]$ in line 9. For example, consider the iteration of the *for* loop started at line 4 with $ptn = \{3,3\}$ and hence $p_0 = 3$ and $z[0] = 2$. Now $s[0] = C'(2,2) = 3$ and $(cbr[0][0], cbr[0][1], cbr[0][2]) = ((0,0),(1,0),(1,1))$.

Each iteration of the *for* loop started at line 13 obtains a new tree in T_i. The relevant subtrees from earlier tree lists are retrieved and then merged into a new tree. Consider $ptn = \{3,2\}$, $(s[0], s[1]) = (2,1)$, $(cbr[0][0], cbr[0][1]) = ((0),(1))$ and $cbr[1][0] = (0)$ mentioned above. For $q = 1$, line 14 produces $(b[0], b[1]) = (1,0)$. Each iteration of the *for* loop started at line 16 adds one or more subtrees to S based on the mixed base number $b[]$. Since $(z[0], z[1]) = (1,1)$, the first iteration adds $T_3[1]$ to S and the next iteration adds $T_2[0]$. They are merged into unlabeled tree $T_5[6]$ shown in Fig. 4 (f).

Next, consider $ptn = \{3,3\}$, $s[0] = 3$, and $(cbr[0][0], cbr[0][1], cbr[0][2]) = ((0,0),(1,0),(1,1))$. The loop started at line 13 iterates 3 times. For $q = 2$, we have $b[0] = 2$. At line 18, $cb = cbr[0][2] = (1,1)$. At line 19, 2 copies of $T_3[1]$ (see Fig. 4) are added to S.

Next, consider MergeUnlabeledTree(). Two unlabeled trees t and t' of i and $j \geq i$ roots respectively may be merged in 3 ways to produce a tree of $i+j$ roots:

- M_1 Their leaf nodes are merged, which is how $T_2[0]$ is obtained from 2 copies of $T_1[0]$ (see Fig. 4).
- M_2 The leaf of t' become the parent of the leaf of t, which is how $T_3[0]$ is obtained from $T_1[0]$ and $T_2[0]$. Note that roles of t and t' cannot be switched.
- M_3 Both leaf nodes may become the parents of a new leaf node, which is how $T_2[0]$ and $T_3[1]$ are merged to produce $T_5[6]$.

When $k > 2$ subtrees are merged into a new tree, 2 subtrees are merged first and the remaining subtrees are merged into the intermediate tree one by one. Which of the 3 ways of merging is used at each step is critical. Incorrect choice produces some trees multiple times while omitting others. The resultant T_i will not be an enumeration. MergeUnlabeledTree() is detailed below:

Algorithm 2 *MergeUnlabeledTree(S)*
Input: a set S of $k \geq 2$ unlabeled (sub)trees.

```
1 sort trees in $S$ in ascending order of number of roots as $(t_0, ..., t_{k-1})$;
2 if $t_0$ has one root and $t_1$ has one root, merge them to $t$ by $M_1$;
3 else if $t_0$ has one root and $t_1$ has 2 or more roots, merge them to $t$ by $M_2$;
4 else merge them to $t$ by $M_3$;
5 for $i = 2$ to $k-1$,
6     if $t_i$ has one root, merge $t$ and $t_i$ to $t'$ by $M_1$;
7     else merge $t$ and $t_i$ to $t'$ by $M_2$;
8     $t = t'$;
9 return $t$;
```

EnumerateUnlabeledTree(n) correctly enumerates unlabeled trees of n roots. A formal proof of correctness is beyond the space limit. Our implementation generates list T_n whose cardinality is shown in Table 1 for $n \leq 10$:

Table 1. Cardinality of T_n for $n \leq 10$.

n	1	2	3	4	5	6	7	8	9	10
$\lvert T_n \rvert$	1	1	2	5	12	33	90	261	766	2312

5 Enumerate Root-Labeled Trees Given Unlabeled Tree

To realize the second step of menu selection, we enumerate root-labeled trees for a given unlabeled tree of n roots. Since left-right order of causal events into the same NIN-AND gate does not matter, the number of root-labeled trees is less than $n!$. We propose the following algorithm based on the idea of assigning labels to each group of roots with mirror subtree (defined below) handling. It enumerates root-labeled trees correctly, although a formal proof is beyond space limit. The root-labeled trees are plotted as they are enumerated.

Algorithm 3 *EnumerateRootLabeledTree(t)*
Input: an unlabeled tree t of n roots.

```
1 if all root nodes have the same child;
2     label roots using $n$ labels in arbitrary order;
3     plot the root-labeled tree;
4 grp = grouping of roots with the same child nodes;
5 search for mirror subtrees using grp;
6 if no mirror subtrees are found, EnumerateNoMirrorRLT(t, grp);
7 else EnumerateMirrorRLT(t, grp);
```

Lines 1 to 3 handle cases such as $T_3[1]$ (Fig. 4). Otherwise, roots of the same child nodes are grouped in line 4. For $T_5[6]$ (Fig. 4), we have 2 groups of sizes 2 and 3. Two labels out of 5 can be assigned to the left group of size 2 in $C(5,2) = 10$ ways and the right group can be labeled using remaining labels. Hence, $T_5[6]$ has 10 root-labeled trees. It contains no mirror subtrees (which we will explain later) and line 6 is executed, as detailed below:

Algorithm 4 *EnumerateNoMirrorRLT(t, grp)*
Input: an unlabeled tree t of r roots without mirror subtree; grouping grp of roots with common child nodes.

```
1  $n = r$;
2  $g$ = number of groups in $grp$;
3  for $i = 0$ to $g - 2$,
4    $k$ = number of roots in group $i$;
5    $s[i] = C(n, k)$;
6    $cb[i]$ stores $k$-combinations of $n$ objects without repetition;
7    $n = n - k$;
8  $count = 1$;
9  for $i = 0$ to $g - 2$, $count = count * s[i]$;
10 for $i = 0$ to $count - 1$,
11   initialize $lab[0..r - 1]$ to $r$ root labels;
12   convert $i$ to a mixed base number $b[0..g - 2]$ using base $s[0..g - 2]$;
13   for $j = 0$ to $g - 2$,
14     get combination $gcb = cb[j][b[j]]$;
15     for each number $x$ in $gcb$, label a root in group $j$ by $lab[gcb[x]]$;
16     remove labels indexed by $gcb$ from $lab[]$;
17   label roots in group $g - 1$ using labels in $lab[]$;
18   plot the root labeled tree;
```

In EnumerateNoMirrorRLT(), lines 3 to 7 process roots group by group. For each group (except the last one) of size k, the number of ways that k labels can be selected from n is recorded in $s[i]$. Here, n is initialized to the number of roots (line 1), and is reduced by k after each group of size k is processed (line 7). The labels in each selection are indexed by $cb[i]$. Lines 8 and 9 count the total number of root labeled trees isomorphic to t.

Lines 10 and onwards enumerate and plot each root-labeled tree. Tree index i is converted to a mixed base number $b[]$. Each $b[j]$ is then used to retrieve the label indexes in $cb[i]$ (line 14). The label list $lab[]$ is initialized in line 11, whose elements are used to label a root group in line 15, and the list is updated in line 16. The first $g - 1$ groups are labeled in the *for* loop of lines 13 to 16. The last group is labeled in line 17.

Next, we consider $T_4[2]$ in Fig. 4 (e). It has 2 root groups of size 2 and the left group can be assigned 2 labels in $C(4, 2) = 6$ ways. However, half of them switch the labeling between left and right group in the other half. Hence, the number of root-labeled trees for $T_4[2]$ is 3, not 6. Applying EnumerateNoMirrorRLT() to $T_4[2]$ would be incorrect. We define *mirror subtrees* for such cases.

Definition 4 *A* `subtree` *s in an unlabeled tree t is a subgraph consisting of a non-root, non-leaf node of t (as the leaf of s) and all its ancestors in t.*

Two subtrees s and s' are `mirror subtrees` *if they are isomorphic, each has more than one root node, and the leaf of s and the leaf of s' have the same path length from the leaf of t in t.*

$T_4[2]$ in Fig. 4 (e) has 2 mirror subtrees, each of which is a copy of $T_2[0]$, and so does $T_5[9]$ in (g). $T_6[17]$ in (h) has 2 mirror subtrees, each of which is a copy of $T_3[0]$. $T_6[27]$ in (i) has 3 mirror subtrees, each of which is a copy of $T_2[0]$. None of the other trees in Fig. 4 has mirror subtrees. In general, a tree of $n \geq 4$ roots may have mirror subtrees from $T_{[n/2]}$, where [.] denotes the floor function. Table 2 shows the possible number and type of mirror subtrees for $n \leq 7$.

Table 2. Number of mirror subtrees that may be present in a tree of n roots.

n	1	2	3	4	5	6	7
$T_2[0]$	0	0	0	2	2	2 or 3	2 or 3
$T_3[0]$	0	0	0	0	0	2	2
$T_3[1]$	0	0	0	0	0	2	2

At line 5 of EnumerateRootLabeledTree(), mirror subtrees are searched. Most trees are rejected based on their *grp*. Otherwise, if the *grp* of a tree is compatible with a pattern in Table 2, its leaf is removed, splitting it into subtrees recursively, and possible mirror subtrees are detected. If mirror subtrees are found, at line 7, EnumerateMirrorRLT() is performed. We present its idea but not pseudocode.

Suppose an unlabeled tree t with mirror subtrees and n roots have $k \leq n$ roots in mirror subtrees. If $n - k > 0$, remaining roots are labeled first in the same way as EnumerateNoMirrorRLT(). Then, for each partially root-labeled tree, mirror subtrees are root-labeled. For example, for $T_5[9]$ in Fig. 4 (g), the left-most root can be labeled in 5 ways as usual, using up one label. The first mirror subtree is labeled in $C(4,2)/2 = 3$ ways, using up another 2 labels. The second mirror subtree is then labeled using the remaining labels. For $T_6[27]$ in (i), the first mirror subtree is labeled in $C(6,2)/3 = 5$ ways, using up 2 labels. The second mirror subtree is labeled in $C(4,2)/2 = 3$ ways, using up another 2 labels. The third mirror subtree is then labeled using the remaining labels. Table 3 shows the number of root-labeled trees given some unlabeled trees in Fig. 4 as enumerated by our implementation of EnumerateRootLabeledTree().

Table 3. Number of root-labeled trees for some given unlabeled trees.

Unlabeled tree	$T_4[2]$	$T_5[6]$	$T_5[9]$	$T_6[17]$	$T_6[27]$
No. root-labeled trees	3	10	15	90	15

Table 4 shows the total number of root-labeled trees with n roots for $n \leq 7$. Since T_7 contains 90 unlabeled trees (Table 1), each has on average $39208/90 \approx 435$ root-labeled trees. Its implication is discussed in the next section.

Table 4. Toal number of root-labeled trees with n roots.

n	1	2	3	4	5	6	7
No. root-labeled trees	1	1	4	26	236	2752	39208

6 Remarks

As learning from data is limited by missing values, small samples, and cost in data collection, elicitation of CPT remains an alternative in constructing BNs when the expert is available. Due to conditional independence encoded in BNs, a CPT that involves more than 10 causes is normally not expected. Even so, the task of eliciting up to 2^{10} parameters is daunting. NIN-AND trees provide a causal model that reduces the number of parameters to be elicited to linear (10 for 10 binary causes), while capturing both reinforcing and undermining interactions among causes. A tree-shaped causal structure of a linear number of nodes (less than 20 for 10 causes), however, must be elicited in addition.

This contribution proposes the two-step menu selection for causal structure elicitation. The technique reduces the cognitive load on the expert, compared to structure elicitation from scratch or the single step menu selection. For the first step, we extend an idea of counting from phylogenetics into an algorithm to enumerate NIN-AND tree structures with unlabeled root nodes. For the second step, we develop an algorithm to enumerate completely labeled NIN-AND tree structures given a structure selected by the expert from the first step. Compared to off-line enumeration, our online enumeration is interactive. Even though the choice from the first step may be inaccurate, the two-step selection can be repeated easily (in seconds) until the expert's satisfaction.

As the average number of root-labeled trees is beyond 400 when the number of causes is beyond 7, we believe that the two-step menu selection is practical for elicitation of NIN-AND tree (and thus CPT) with up to 7 causes. For CPTs with 8 causes or beyond, we have developed an alternative technique to be presented elsewhere. Empirical evaluation of our proposed elicitation techniques with human experts is underway.

Acknowledgements

Financial support from NSERC, Canada through Discovery Grant to the first author is acknowledged. We thank reviewers for their comments.

References

1. A. Cayley. A theorem on trees. *Quarterly J. Mathematics*, pages 376–378, 1889.
2. J. Felsenstein. *Inferring Phylogenies.* Sinauer Associates, Sunderland, Mass., 2004.
3. S.F. Galan and F.J. Diez. Modeling dynamic causal interaction with Bayesian networks: temporal noisy gates. In *Proc. 2nd Inter. Workshop on Causal Networks*, pages 1–5, 2000.
4. D. Heckerman and J.S. Breese. Causal independence for probabilistic assessment and inference using Bayesian networks. *IEEE Trans. on System, Man and Cybernetics*, 26(6):826–831, 1996.
5. J.F. Lemmer and D.E. Gossink. Recursive noisy OR - a rule for estimating complex probabilistic interactions. *IEEE Trans. on System, Man and Cybernetics, Part B*, 34(6):2252–2261, 2004.
6. J.W Moon. *Counting Labeled Trees.* William Clowes and Sons, London, 1970.
7. J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, 1988.
8. J. Riordan. The enumeration of trees by height and diameter. *IBM J.*, pages 473–478, Nov. 1960.
9. Y. Xiang and N. Jia. Modeling causal reinforcement and undermining for efficient cpt elicitation. *IEEE Trans. Knowledge and Data Engineering*, 19(12):1708–1718, 2007.