On the impact of causal independence

Irina Rish and Rina Dechter
Department of Information and Computer Science
University of California, Irvine
{*irinar,dechter*}*@ics.uci.edu*

October 13, 1998

Abstract

Reasoning in Bayesian networks is exponential in a graph parameter w^* known as *induced width* (also known as tree-width and max-clique size). In this paper, we investigate the potential of *causal independence (CI)* for improving this performance. We consider several tasks, such as belief updating, finding a most probable explanation (MPE), finding a maximum aposteriori hypothesis (MAP), and finding the maximum expected utility (MEU).

We show that exploiting CI in belief updating can significantly reduce the *effective* w^*, sometimes down to the induced width of the unmoralized network's graph. For example, for poly-trees, CI reduces complexity from exponential to linear in the family size. Similar results hold for the MAP and MEU tasks, while the MPE task is less sensitive to CI. These enhancements are incorporated into *bucket-elimination* algorithms based on known approaches of *network transformations* [10, 13] and elimination [18]. We provide an ordering heuristic which guarantees that exploiting CI will never hurt the performance.

Finally, we discuss an efficient way of propagating evidence in CI-networks using *arc-consistency*, and apply this idea to noisy-OR networks. The resulting algorithm generalizes the *Quickscore* algorithm [9] for BN2O networks.

1 Introduction

Bayesian networks is a widely used framework for reasoning under uncertainty. However, probabilistic inference in Bayesian networks is NP-hard [4]. Commonly used structure-exploiting algorithms such as join-tree propagation [12, 11, 16] and variable elimination [18, 7] are time and space exponential in the *induced width* (the *maximal clique size*) of the network's moral graph. The induced width is often large, especially in the networks with large families. Even the input conditional probability tables (CPTs) (which are exponential in the family size) are too large to be specified explicitly in such networks. One way to cope with this problem is to identify the structural properties that simplify the CPT's specification. In this paper, we focus on a property known as *causal independence* [10, 17, 18], where multiple causes contribute independently to a common effect. Examples of causally-independent probabilistic relationships are *noisy-OR* and *noisy-MAX*, that are used as simplifying assumptions in large practical systems such as QMR-DT network for medical diagnosis [15].

Our work is developed along the two main approaches of [10, 13] and [18], which demonstrate some computational benefits of causal independence. The first approach transforms each family into a binary-tree Bayesian network using hidden variables. This results in a probabilistic network whose largest parent set size is bounded by two. The second approach (called VE1 in [18]) applies a network transformation implicitly during the execution of a variable-elimination algorithm. We identify cases in which VE1 may be less efficient than the general-purpose elimination and compensate for this by using an appropriate ordering heuristic.

We demonstrate that the "effective" induced width of algorithms exploiting causal independence can be significantly reduced: it can be as small as the induced width of the unmoralized network's graph, and is guaranteed not to exceed the induced width of the network's moral graph if a proper variable ordering is used. For each given network, the anticipated computational benefits can be evaluated in advance and contrasted with those of the general-purpose algorithm. For example, exploiting causal independence in poly-trees with arbitrary large family size m reduces the effective induced width from m to 2.

Next, the impact of causal independence on finding a most probable explanation (MPE), finding a maximum aposteriori hypothesis (MAP), and finding the maximum expected utility (MEU) is investigated. Surprisingly, while causal independence can significantly reduce the complexity of belief updating and finding MAP and MEU, it does not generally help in finding MPE.

We show that causal independence allows a more efficient way of handling observations. A causally-independent network can be transformed into one that combines probabilistic and deterministic relations. Constraint-propagation techniques can be applied to the transformed network in order to simplify probabilistic inference. In particular, given an observation of a child node, *arc-consistency* can further propagate this evidence through the network. We outline a general belief-updating algorithm for causally-independent networks that uses evidence propagation. We also present a customized version of this algorithm for noisy-OR networks that generalizes the *Quickscore* algorithm proposed by [9] for a particular class of binary-node two-layer noisy-OR networks (BN2O networks).

We will proceed in the following manner. Section 2 supplies some necessary definitions, while Section 3 provides background on causal independence. In Section 4 prior work on network transformations is reviewed and analyzed. A variable elimination approach to causal independence is presented in Section 5, while Section 6 discusses the connection between this approach and previous ones. In Section 7, our analysis is extended to the tasks of finding MPE, MAP and MEU. Evidence propagation in general causally-independent networks is discussed in Section 8, while its application to noisy-OR networks is presented in Section 9. Section 10 concludes the paper.

2 Definitions and preliminaries

Let $X = \{x_1, ..., x_N\}$ be a set of random variables over a domain of size d. A *belief network* is a pair (G, P), where G is a directed acyclic graph on X, and $P = \{P(x_i|pa_i)|i = 1, ..., N\}$ is the set of conditional probabilities defined for each x_i and its *parents* pa_i (a node y is called a *parent* of x if there is a directed edge from y to x). The belief network represents a joint probability distribution over X having the product form

$$P(x_1, ..., x_N) = \prod_{i=1}^{N} P(x_i|pa_i).$$

We call a node and its parents a *family*. The *moral graph* G_M of a belief network (G, P) is obtained by connecting the parents of each node in G and dropping directionality of edges. An example of a belief network is shown in Figure 1a, where the dashed line denotes the edge added by moralization. An *evidence* e is an instantiated subset of variables. The following tasks can be defined on belief networks:

1. *belief updating*, i.e. finding the posterior probability $P(Y|e)$ of *query* nodes $Y \subseteq X$ given evidence e;

2. finding *most probable explanation* (MPE), i.e. finding a maximum probability assignment to unobserved variables given evidence;

3. finding *maximum aposteriory hypothesis* (MAP), i.e. finding a maximum probability assignment to a set of *hypothesis* nodes, given evidence;

4. given a *utility* function, finding the *maximum expected utility* (MEU), namely, finding assignment to a set of *decision* nodes that maximizes the expected *utility* function specified on the network's nodes.

The most common belief-updating algorithms include join-tree propagation [12, 11, 16] and variable elimination [5, 18, 7].

We briefly review the *bucket elimination* algorithm *elim-bel* [7] for belief updating. Assume that the set of query nodes is $Y = \{x_1\}$. By definition of conditional probability,

$$P(x_1|e) = \alpha P(x_1, e) = \alpha \sum_{x_2} \dots \sum_{x_N} \prod_i P(x_i|pa_i), \qquad (1)$$

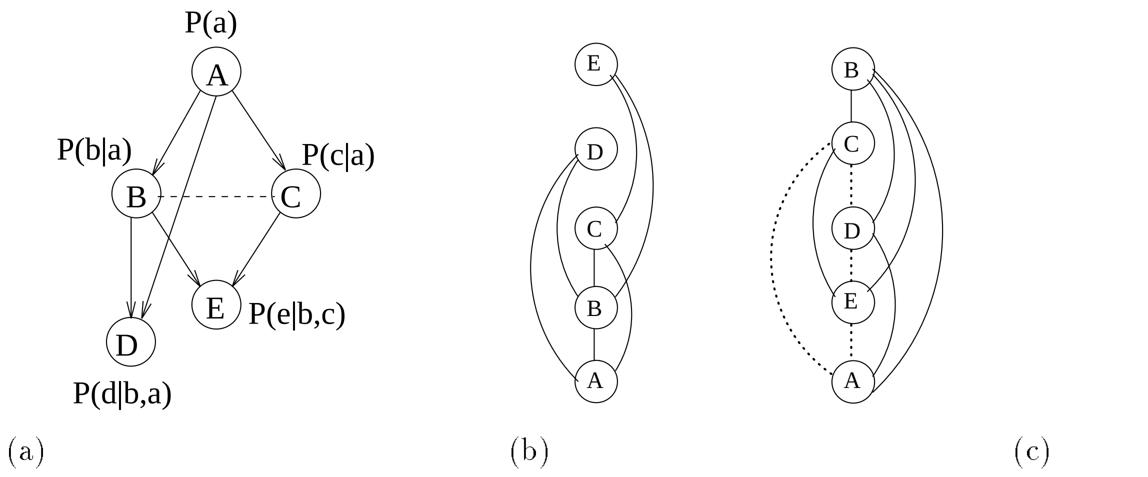


Figure 1: (a) Bayesian network. (b) The induced graph along $o = (a, b, c, d, e)$. (c) The induced graph along $o = (a, e, d, c, b)$

where α is a normalizing constant. By distributivity law,

$$\sum_{x_2} \ldots \sum_{x_N} \prod_i P(x_i|pa_i) = F_1 \sum_{x_2} F_2 \ldots \sum_{x_N} F_N, \tag{2}$$

where each $F_i = \prod_x P(x|pa(x))$ is the product of all probabilistic components such that either $x = x_i$, or $x_i \in pa(x)$. The set of all such components is initially placed in the *bucket* of x_i (denoted $bucket_i$). Algorithm *elim-bel* processes the buckets from N to 1, thus computing the summation 2 from right to left. For each $bucket_i$, it multiplies the bucket's components, sums over x_i, and puts the resulting function in the bucket of its highest-index variable. Similar bucket elimination algorithms can be derived for finding MPE, MAP, and MEU [7].

The example below illustrates elim-bel on the network in Figure 1a.

Example 1: Given a belief network in Figure 1a, the ordering $o = (a, b, c, d, e)$ and evidence $e = 0$, let us find $P(a|e = 0)$:

$$P(a|e=0) = \sum_{b,c,d,e=0} P(a,b,c,d,e) = \sum_b \sum_c \sum_d \sum_{e=0} P(a)P(c|a)P(e|b,c)P(d|a,b)P(b|a) =$$

$$\sum_b P(b|a) \sum_c P(c|a) \sum_d P(d|a,b) \sum_{e=0} P(e|b,c).$$

The bucket elimination algorithms computes the sum from right to left as follows:
1. bucket e: $h^e(b, c) = P(e = 0|b, c)$
2. bucket d: $h^d(a, b) = \sum_d P(d|a, b)$

3. bucket c: $h^c(a,b) = \sum_c P(c|a)h^e(b,c)$
4. bucket e: $h^b(a) = \sum_b P(b|a)h^c(a,b)$
5. bucket a: $P(a|e=0) = \alpha h^b(a)$,

where α is a normalization constant.

The complexity of the bucket elimination algorithms is exponential in the *induced width* w_o^* [8] of the network's moral graph along the processed ordering o. Given a graph G, the *width* of x_i along o is the number of x_i's earlier neighbors in o. The *width of the graph* along o, w_o, is the largest width along o. The *induced graph* of G along o is obtained by connecting the preceding neighbors of each x_i, proceeding from x_N to x_1. The *induced width* along o, w_o^*, is the width of the induced graph along o, while the *induced width* w^* is the minimum induced width along any ordering. For example, Figures 1b and 1c show the induced graphs of the moral graph in Figure 1a along the orderings $o = (a, b, c, d, e)$ and $o' = (a, e, d\ , c, b)$, respectively. Obviously, $w_o^* = 2$ and $w_{o'}^* = 4$. It can be shown that the induced width of each node x_i is identical to the number of arguments of a function computed in $bucket_i$. Therefore,

Theorem 1: *[7] The complexity of algorithm elim-bel is $O(Nd^{w_o^*+1})$, where w_o^* is the induced width of the moral graph along ordering o.*

Although finding an ordering having smallest induced width is NP-hard [1], good heuristic orderings are available [3, 6, 2].

3 Causal independence

Causal independence assumes that several causes contribute independently to a common effect. Specifically, a probabilistic relation between a set of causes $c_1, ..., c_n$ and an effect e (Figure 2a) can be decomposed into a noisy transformation of each cause c_i into a *hidden* variable u_i, and a deterministic function $e = u_1 * ... * u_n$, where $*$ is a commutative and associative binary operator. A graph depicting this relation (*dependency graph*) is shown in Figure 2b. Formally,

Definition 1: [10, 18] Let $c_1, ..., c_m$ the parents of e in a Bayesian network. The variables $c_1, ..., c_m$ are said to be *causally-independent* w.r.t. e if there exists a set of random variables $u_1^e, ..., u_m^e$, a set of probabilities $P(u_i^e|c_i)$, and a binary commutative and associative operator $*$ such that

1. for each i, u_i^e is independent on any of c_j and u_j^e, $i \neq j$, given c_i, i.e.
$$P(u_i^e|c_1, ..., c_n, u_1^e, ..., u_n^e) = P(u_i^e|c_i), \text{ and}$$
2. $e = u_1^e * \ldots * u_n^e$.

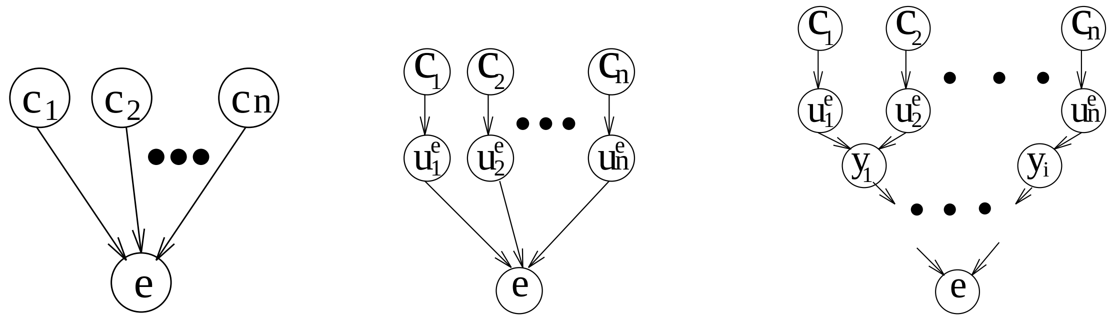


Figure 2: (a) causally-independent belief network; (b) its dependency graph and (c) its decomposition network

A CPT of such a causally-independent family can therefore be specified by m components $P(u_i^e|c_i)$ and expressed as

$$P(e|c_1^e, ...c_m^e) = \sum_{\{u_1^e...u_m^e|e=u_1^e*...*u_m^e\}} \prod_{i=1}^{m} P(u_i^e|c_i^e). \tag{3}$$

This summation can be computed linearly in m using pairwise decomposition, such as

$$\sum_{\{u_1^e,y_1|e=u_1^e*y_1\}} P(u_1^e|c_1^e) \sum_{\{u_2^e,y_2|y_1=u_2^e*y_2\}} P(u_2^e|c_2^e) \ldots \sum_{\{u_{m-1}^e,u_m^e|y_{m-2}=u_{m-1}^e*u_m^e\}} P(u_{m-1}^e|c_{m-1}^e)P(u_m^e|c_m^e),$$

where y_j are *hidden* variables introduced for keeping the intermediate results. Such pairwise summation corresponds to a traversal of a binary computation tree (Figure 2c), where each non-leaf node y represents the result of an operation $y = y_l * y_k$ on its children y_l and y_k.

Given a belief network (G, P), replacing each causally-independent family in G by such computation tree (i.e., replacing Figure 2a by Figure 2c) yields a *decomposition graph* G_D. This transformation introduces *hidden* nodes to G, which include both the nodes u_i^e due to the definition of causal independence, and the nodes y_j of each computation tree. We assume that the probabilities in P are specified as $P = \{P(u_j^x|c_j^x)|c_j^x \in pa(x)\}$. A *decomposition network* of a belief network (G, P) is defined by tuple (G_D, P, C), where G_D is a decomposition graph, and C is the set of constraints $y = y_l * y_k$ introduced by the computation trees. The nodes of the input belief network are called the *input* nodes.

A decomposition network closely resembles the *network transformations* [10, 13] discussed in the next section with a minor variation: instead of compiling a new CPT for each hidden variable y having parents y_l and y_k, a decomposition network keeps the constraint $y = y_l * y_k$.

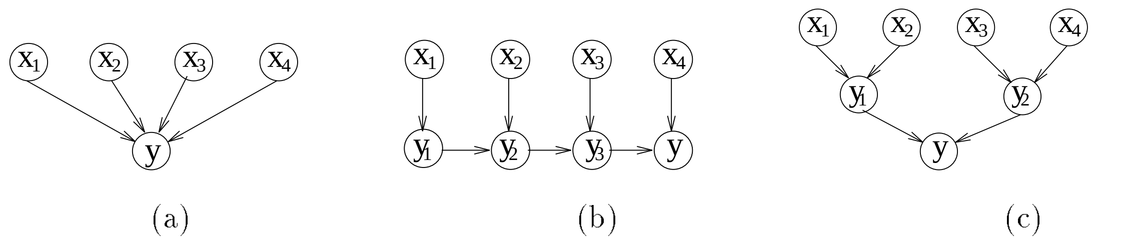


Figure 3: (a) A Bayesian network; (b) temporal transformation (c) parent divorcing.

4 Network transformations

It was shown that a causally independent family can be transformed into a belief network with a smaller parent size. This transformation can be achieved by various means. We will elaborate first on two known methods: *temporal transformation* [10] and *parent divorcing* [13].

Example 2: [temporal transformation]
Consider the causally-independent network in Figure 3(a). By definition of causal independence,

$$P(y|x_1,x_2,x_3,x_4) = \sum_{\{u_1,u_2,u_3,u_4|y=u_1*u_2*u_3*u_4\}} P(u_1|x_1)P(u_2|x_2)P(u_3|x_3)P(u_4|x_4).$$

Since $*$ is commutative and associative, $y = u_1 * u_2 * u_3 * u_4$ can be computed in several different ways. Assume the computation order $y = ((u_1 * u_2) * u_3) * u_4$. Let $y_1 = u_1$, $y_2 = y_1 * u_2$, $y_3 = y_2 * u_3$, then $P(y|x_1,x_2,x_3,x_4) =$

$$= \sum_{\{y_3,u_4|y=y_3*u_4\}} \sum_{\{y_2,u_3|y_3=y_2*u_3\}} \sum_{\{y_1,u_2|y_2=y_1*u_2\}} P(y_1|x_1)P(u_2|x_2)P(u_3|x_3)P(u_4|x_4) =$$

$$= \sum_{y_3} \sum_{\{u_4|y=y_3*u_4\}} P(u_4|x_4) \sum_{y_2} \sum_{\{u_3|y=y_2*u_3\}} P(u_3|x_3) \sum_{y_1} \sum_{\{u_2|y_2=y_1*u_2\}} P(u_2|x_2)P(y_1|x_1).$$

Conditional probabilities for y_1, y_2, y_3, and y can now be defined as: $P(y_1|x_1) = P(u_1|x_1)$, $P(y_2|y_1,x_2) = \sum_{\{u_2|y_2=y_1*u_2\}} P(u_2|x_2)$,
$P(y_3|y_2,x_3) = \sum_{\{u_3|y_3=y_2*u_3\}} P(u_3|x_3)$, $P(y|y_3,x_4) = \sum_{\{u_4|y=y_3*u_4\}} P(u_4|x_4)$, then

$$P(y|x_1,x_2,x_3,x_4) = \sum_{y_3} P(y|y_3,x_4) \sum_{y_2} P(y_3|y_2,x_3) \sum_{y_1} P(y_2|y_1,x_2)P(y_1|x_1).$$

In other words, the original network is transformed into an equivalent one having two additional (*hidden*) variables y_1 and y_2 (Figure 3(b)). Clearly, queries defined on the original network can be answered using an equivalent transformed network. The transformation described above is known as *temporal transformation* proposed by Heckerman and Breese [10].

Example 3: [parent divorcing]
Another network transformation method known as *parent divorcing* [13], is illustrated in Figure 3c. Parent divorcing assumes the computation ordering $y = (x_1 * x_2) * (x_3 * x_4)$, while temporal transformation assumes $y = (x_1 * x_2) * x_3) * x_4$. Then

$$P(y|x_1, x_2, x_3, x_4) = \sum_{\{u_1,u_2,u_3,u_4|y=u_1*u_2*u_3*u_4\}} P(u_1|x_1)P(u_2|x_2)P(u_3|x_3)P(u_4|x_4) =$$

$$= \sum_{\{y_1,y_2|y=y_1*y_2\}} \sum_{\{u_1,u_2|y_1=u_1*u_2\}} \sum_{\{u_3,u_4|y_2=u_3*u_4\}} P(u_1|x_1)P(u_2|x_2)P(u_3|x_3)P(u_4|x_4) =$$

$$= \sum_{\{y_1,y_2|y=y_1*y_2\}} \sum_{\{u_1,u_2|y_1=u_1*u_2\}} P(u_1|x_1)P(u_2|x_2) \sum_{\{u_3,u_4|y_2=u_3*u_4\}} P(u_3|x_3)P(u_4|x_4) =$$

$$= \sum_{\{y_1,y_2|y=y_1*y_2\}} P(y_1|x_1, x_2)P(y_2|x_3, x_4) = \sum_{y_1}\sum_{y_2} P(y|y_1, y_2)P(y_1|x_1, x_2)P(y_2|x_3, x_4),$$

where
$P(y_1|x_1, x_2) = \sum_{\{u_1,u_2|y_1=u_1*u_2\}} P(u_1|x_1)P(u_2|x_2)$,
$P(y_2|x_3, x_4) = \sum_{\{u_3,u_4|y_2=u_3*u_4\}} P(u_3|x_3)P(u_4|x_4)$, and
$P(y|y_1, y_2) = 1$ if $y = y_1 * y_2$.

Introducing hidden variables that cache intermediate results decreases CPT size. Given a causally-independent family having m parents x_1,...,x_m, *temporal transformation* adds $m - 1$ hidden variables y_1,...,y_{m-1} and defines the new CPTs as follows:
$P(y_1|x_1) = P(u_1^y|x_1)$,
$P(y_i|x_i, y_{i-1}) = \sum_{\{u_i^y|y_i=y_{i-1}*u_i\}} P(u_i^y|x_i)$,
$P(y|x_m, y_{m-1}) = \sum_{\{u_m^y|y=y_{m-1}*u_m\}} P(u_m^y|x_m)$.
The *parent divorcing method* builds a binary tree with the leaf nodes x_1,...,x_m and the root y, introducing hidden variables y_i^j level by level, from the parent level $j = 0$ to the child level $j = log(m)$.

There are many possible transformations of a causally independent family into a binary tree. Each transformation represents a particular order of computations.

Definition 2: Given a belief network, we denote the result of its possible *temporal transformation* by TT, and the result of its possible *parent-divorcing transformation* by PD. In general, the result of an arbitrary *binary-tree transformation* is denoted by T.

Once the transformed network is available, a standard junction tree or a variable elimination algorithm can be applied. As noted in [10], the transformation ordering may have a significant impact on the performance of an inference algorithm. However, finding a good transformation ordering may be intricate, especially in large complex networks.

4.1 Complexity analysis

In this section we make a few observations regarding the relative computational gain when exploiting causal independence. We assume a belief network defined on N nodes, each having a domain of size d and no more than m parents.

Since network transformations convert each CPT defined on $m+1$ nodes (m parents and a child) into a belief network where each family has no more than 3 variables, the size of a problem's specification decreases from $O(d^{m+1})$ to $O(md^3)$.

Proposition 1: [transformation complexity]
The complexity of a network transformation and the size of its output T is $O(Nmd^3)$.

Proof: For each causally-independent family having m parents, its transformation introduces no more than m hidden variables (the number of internal nodes in a binary tree having m leaves). Correspondingly, there are no more than m new CPTs, each defined on 3 variables. This yields $O(md^3)$ time and space complexity. Since T has $O(Nm)$ nodes and family sizes not exceeding 3, its specification requires $O(Nmd^3)$ space. $\square$

Since T has $O(Nm)$ nodes, it immediately follows from Theorem 1 that

Proposition 2: [inference complexity]
The complexity of belief updating in a causally-independent network is $O(Nmd^{w_o^*+1})$, where w_o^* is the induced width of the moralized transformed network along an ordering o. $\square$

An ordering having a relatively low induced width can be found in advance using greedy ordering heuristics (see [3, 6] for details) on the transformed network's graph.

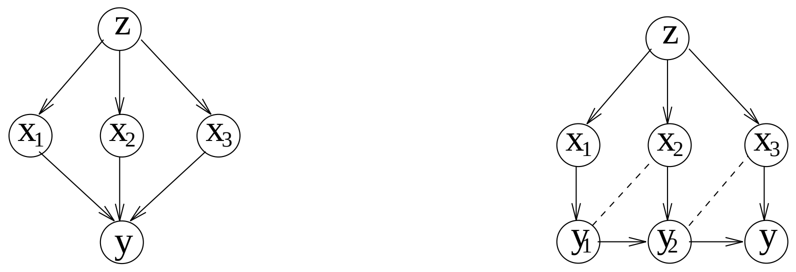
Occasionally network transformations allow a significant decrease in the induced width, which leads to an exponential complexity reduction.

Proposition 3: [poly-trees]
Given a causally-independent poly-tree, the complexity of belief update on the transformed network is $O(Nmd^3)$, while its complexity on the original network is $O(Nd^{m+1})$.

Proof: A family with m parents is associated with a CPT on $m+1$ variables which yields a $O(d^{m+1})$ time and space complexity. Obversely, a transformed network is a poly-tree having families of size 3 or smaller. It allows an ordering having $w^* = 2$, where query nodes appear at the beginning (as required by elim-bel). Therefore, the complexity of elim-bel on the transformed poly-trees is $O(Nmd^3)$. $\square$

Proposition 3 generalizes Pearl's result for noisy-OR poly-trees [14]. Remember, the standard poly-tree algorithm is exponential in size of a largest parent set, i.e. $O(Nd^m)$.

Although the transformations always reduce the family size of the input network, it is not obvious how introducing hidden variables affects w^*. We will now examine the relationship between w^* of the transformed network relative to w^* of the unmoralized input network.



(a) (b)

Figure 4: (a) A Bayesian network and (b) its (moralized) temporal transformation

Proposition 4: [w^* **on the input variables**]
Given an (unmoralized) network BN having induced width w^* along an ordering o, and given a transformed network T, there exists an extension of o to o' such that the induced width of moralized T along o', computed only with regard to the variables of BN, is not larger than w^*. $\Box$

The proof of this proposition can be found in the Appendix.

We will explore some classes of networks in which the induced width of the transformed network is the same as the induced width of the unmoralized input network.

Example 4: Consider the network in Figure 4a and its moralized temporal transformation in Figure 4b. Given the ordering $o = z, y, x_3, x_2, x_1$, the induced width of the original unmoralized network is 2. The induced width of the moralized transformed network along the ordering $o' = z, y, y_1, y_2, x_3, x_2, x_1$ is also 2. Note that the induced width of the moralized original network is 3.

In general,

Proposition 5: *If no two convergent variables in a network BN have more than one common parent (e.g., Figure 4a), then the induced width of T equals the induced width of the unmoralized BN.* $\Box$

See Appendix for proof.

Causal independence is exploited whenever parents are eliminated before their children, otherwise eliminating a child will reconstruct the full CPT. However, there are cases when it is better to ignore causal independence.

Example 5: Consider the network in Figure 5a, and its decomposition network in Figure 5b. Eliminating each parent x_i connects u_i^a, u_i^b, and u_i^c to each other, therefore, eliminating next the pair $\{u_1^a, u_2^a\}$ connects a to the rest of the hidden variables, yielding $w^* = 5$.

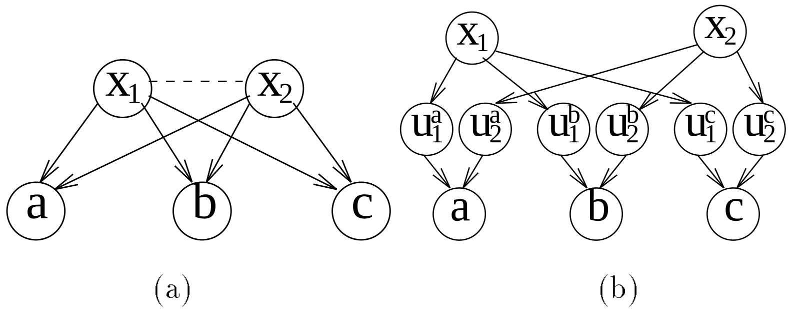


Figure 5: (a) A 2-3 network and (b) its decomposition network

However, the induced width of the original moral graph (Figure 5a) along o =(a, b, c, x_1, x_2) is just 2. The same w^* is attained in the decomposition network when the hidden variables are eliminated first (i.e., the original CPTs are reconstructed).

The network in Figure 5a is an example of a *k-n-network*, which is a two-layer network with k nodes in the top layer, each connected to (some of) n nodes in the bottom layer. Other examples of k-n-networks are *binary-node 2-layer noisy-OR (BN2O)* networks, such as the ones used in QMR-DT knowledge base [15].

The example above suggests that eliminating parents (e.g., x_1 and x_2) before their children and before their hidden nodes in a k-n-network yields $O((k+n)d^{2n})$ complexity, while the complexity of eliminating the hidden nodes first (thus reconstructing the CPTs) is $O((k+n)d^k)$. Therefore,

Proposition 6: [k-n-networks] *The complexity of belief update in a causally-independent k-n-network is* $O(d^{min\{k,2n\}})$. $\Box$

Note that when $k > 2n$, exploiting causal independence can exponentially improve the performance of standard belief update algorithms. However, when $k < 2n$, causal independence should be ignored.

Figure 6 presents a variable ordering procedure which guarantees that exploiting causal independence does not hurt the performance of belief update algorithms. The procedure finds a good heuristic ordering of the transformed network and then, if necessary, restores some CPTs in order to keep w^* of the transformed network lower than or equal to w^* of the original network.

The impact of the network transformations on the performance of elim-bel can be summarized as follows:

1. On poly-trees, exploiting causal independence improves the performance of belief update from $O(Nd^m)$ to $O(Nmd^3)$, where N is the number of nodes in BN, d is the domain size, and m bounds the parent set size.

```
Procedure: Order
Input:  a network BN, and a transformed network $T$.
Output: an ordering $o_T$ of $T$ s.t. $w^*_{o_T} \leq w^*_{o_I}$,
        where $w^*_{o_I}$ is the restriction of $o_T$ to the variables of BN.
1. find a good heuristic ordering $o_T$ of the transformed network.
   Let $o_I$ be the restriction of $o_T$ to the variables of BN.
2. For each variable $x$ in $o_T$, going from last to first,
       If $w^*_{o_T}(x) > w^*_{o_I}$:
          If $x$ is an input variable, move all hidden variables $y_i^x$
          related to $x$ to the top of the ordering $o_T$.
          Else if $x = y_i^z$ is a hidden variable of some convergent variable $z$,
          put all $z$'s hidden variables at the top of the ordering $o_T$.
3. Return $o_T$.
```

Figure 6: Procedure *order*.

2. In many cases, when the w^* of moralized transformed network equals the w^* of the *unmoralized* input network, the performance improves exponentially.

3. Exploiting causal independence in *k-n-networks* yields $O((k+n)d^{min\{k,2n\}})$ complexity, which is better than $O((k+n)d^k)$ complexity of general-purpose algorithms for bounded n and increasing k.

4. Sometimes a bad ordering of the hidden variables may *increase* the inference complexity. We provide the ordering repair procedure *order* that guarantees non-increasing complexity when using network transformations.

5 Variable Elimination

In this section, we present algorithm *ci-elim-bel* for belief updating in causally-independent networks. This algorithm is a re-derivation of elimination algorithm VE1 [18] that avoids certain ordering restrictions of the latter and allows a better performance. Our algorithm works directly on the input specification of causally-independent families, and uses a graph transformation only to guide the ordering.

The algorithm's derivation is demonstrated below.

Example 6: Consider the *noisy-OR* belief network in Figure 7a. Causal independence is explicitly depicted by the graph in Figure 7b. The query $P(x_3|x_1 = 0)$ can be computed

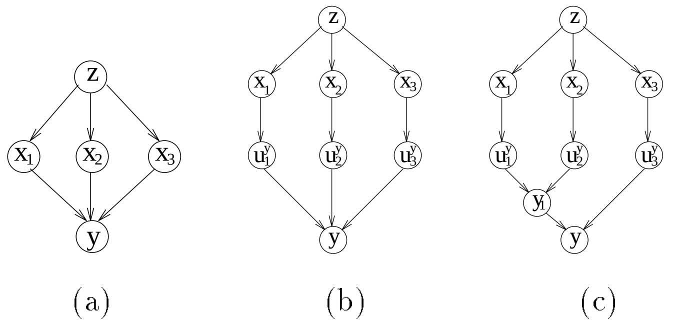


Figure 7: (a) A belief network, (b) its dependency graph, and (c) its decomposition graph

as follows:

$$\sum_{z,y,x_2,x_1=0} P(z)P(x_1|z)P(x_2|z)P(x_3|z)P(y|x_1,x_2,x_3) =$$

$$\sum_{z,y,x_2,x_1=0} P(z)P(x_1|z)P(x_2|z)P(x_3|z) \sum_{\{u_1^y,u_2^y,u_3^y|y=u_1^y\vee u_2^y \vee u_3^y\}} P(u_1^y|x_1)P(u_2^y|x_2)P(u_3^y|x_3).$$

Assuming the decomposition network in Figure 7c, decomposing $\sum_{\{u_1^y,u_2^y,u_3^y|y=u_1^y\vee u_2^y\vee u_3^y\}}$ accordingly, and pushing the summation over x_1 and x_2 as far to the right as possible, yields:

$$\sum_z P(z)P(x_3|z)\sum_y \sum_{\{y_1,u_3^y|y=y_1\vee u_3^y\}} P(u_3^y|x_3) \sum_{\{u_1^y,u_2^y|y_1=u_1^y\vee u_2^y\}} \sum_{x_2} P(x_2|z)P(u_2^y|x_2) \sum_{x_1=0} P(x_1|z)P(u_1^y|x_1).$$

The variables can be summed out from right to left along the ordering $o = (x_3,\ z,\ y,\ \{y_1, u_3^y\},\ \{u_1^y, u_2^y\},\ x_2,\ x_1)$.

The summations can be rearranged in any *legal* order, where each pair of hidden variables $\{y_l, y_k\}$, where $y = y_l * y_k$, is summed out before its parent y in the decomposition tree. The definition of width and induced width can be easily extended to orderings that include subsets of variables.

Algorithm *ci-elim-bel* (Figure 8) computes $P(x_1|e)$. It assumes as an input a decomposition network (G_D, P, C), a legal ordering $o = (Z_1, ..., Z_n)$, where $Z_1 = \{x_1\}$, and evidence e. As previously noted, Z_i may be either a single variable, or a pair of hidden sibling variables in the decomposition network. Each Z_i is associated with its bucket ($bucket_i$) relative to o. First, the components of P, C and e are partitioned into buckets. Each component is placed in the bucket of its highest-index variable. Then the buckets are processed along the reverse ordering, from Z_n to Z_1. If $bucket_i$ contains evidence $x = a$, the algorithm instantiates x in all components and place each in an appropriate lower bucket. Otherwise, it computes the product F of the bucket's components and sum out Z_i's variable(s). If Z_i is a single variable x, we compute $\sum_x F$. Otherwise, if Z_i is a pair (y_l, y_k), then constraint

$y = y_l * y_k$ is in the bucket. We compute $\sum_{\{y_l,y_k|y=y_l*y_k\}} F$ and add the resulting component in the bucket of its highest variable. Finally, $P(x_1|e)$ is computed in $bucket_1$ as the normalized product of all its components.

The complexity analysis of *ci-elim-bel* can be expressed in terms of the decomposition graph and is identical to that of the transformation networks.

```
Algorithm ci-elim-bel

Input: A decomposition network, a legal ordering $o = (Z_1, ..., Z_n)$,
       and evidence $e$.
Output: $P(x_1|e)$.
1. For  $i = n$ to $i = 1$, /* create buckets */
       For each  $x \in Z_i$, put in $bucket_i$ all network's components
           whose highest ordered variable is $x$.
2. For $i = n$ downto 1 do /* process buckets */
      /* $\lambda_1, ..., \lambda_m$ are probabilistic components in $bucket_i$. */
• If $x = a$ is in the bucket, replace $x$ by $a$ in each $\lambda_i$
    and put the result in appropriate lower bucket.
• else
    if $Z_i = \{x\}$, /* input variable */
      $\lambda^{Z_i} \leftarrow \sum_x \Pi_j \lambda_j$.
    else /* $Z_i = \{y_l, y_k\}$, $y = y_l * y_k$. */
      $\lambda^{Z_i} \leftarrow \sum_{\{y_l,y_k|y=y_l*y_k\}} \Pi_j \lambda_j$.
    Put $\lambda^{Z_i}$ in the highest bucket that mentions $\lambda^{Z_i}$'s variable.
3. Return $Bel(x_1) = \alpha\lambda^{x_1}$, where $\alpha$ is a normalizing constant.
```

Figure 8: Algorithm ci-elim-bel

6 Connection between network transformations, VE1, and ci-elim-bel

Algorithm *ci-elim-bel* is similar to network transformations. Both methods assume a decomposition of the input network into its decomposition graph. However, rather than computing the CPTs of the transformed network in advance, *ci-elim-bel* postpones this computation until the bucket processing.

On the other hand, *ci-elim-bel* is very similar to the variable elimination algorithm VE1 proposed in [18]. Like *ci-elim-bel*, VE1 exploits the causal independence within the elimination process and uses a "fine-grain" representation of causally-independent CPTs

$P(x|pa(x))$ via their *contributing* factors $P(u_i^x|y_i)$, $y_i \in pa(x)$. This algorithm is applied to the input network, and does not explicitly mention the hidden variables u_i^x introduced by causal independence. These variables are eliminated implicitly, using a new operator $\otimes$. This makes the derivation of the algorithm less transparent and assumes some restrictions on the elimination ordering of the hidden variables. In certain cases, such as k-n-networks, those restrictions can make VE1 exponentially worse than the original algorithm it tries to improve, similar to what we observed with certain orderings of the transformed networks. The flexibility of hidden variable ordering in the network transformation approach and in the *ci-elim-bel* approach allows for ordering repair (Figure 6). Algorithm *ci-elim-bel* bridges the gap between VE1 and the network transformations. On one hand, it is a variable elimination variant of the network transformation approach. On the other hand, it is identical to VE1 for certain restricted orderings. VE1 handles hidden variables u_i^e in *contributing factors* $f^i(e, c_i) = P(u_i^e|c_i)$ (see definition 1) using a new operator $\otimes$ defined as follows:

$$f \otimes g(e_1 = \alpha_1, ..., e_k = \alpha_k, Y) =$$
$$\sum_{\alpha_{11} * \alpha_{12} = \alpha_1} \cdots \sum_{\alpha_{k1} * \alpha_{k2} = \alpha_k} f(e_1 = \alpha_{11}, ..., e_k = \alpha_{k1}, Y_1) \cdot g(e_1 = \alpha_{12}, ..., e_k = \alpha_{k2}, Y_2),$$

where $e_1, ..., e_k$ are all convergent variables that appear both in f and g, and $Y = Y_1 \cup Y_2$ is the set of all the other variables involved in f and g.

Using the operator $\otimes$ and denoting $f^i(e, c_i) = P(u_i^e, c_i)$ shortens the notation. Consider the network in Figure 5. An expression for $P(c)$

$$\sum_b \sum_a \sum_{\{u^a_{x_1}, u^a_{x_2} | a = u^a_{x_1} \vee u^a_{x_2}\}} \sum_{\{u^b_{x_1}, u^b_{x_2} | b = u^b_{x_1} \vee u^b_{x_2}\}} \sum_{\{u^c_{x_1}, u^c_{x_2} | c = u^c_{x_1} \vee u^c_{x_2}\}} \sum_{x_2} P(u^a_{x_2}|x_2) P(u^b_{x_2}|x_2) P(u^c_{x_2}|x_2) \times$$
$$\times \sum_{x_1} P(u^a_{x_1}|x_1) P(u^b_{x_2}|x_2) P(u^c_{x_1}|x_1) = \tag{4}$$

can be written as follows:

$$P(c) = \sum_b \sum_a (\sum_{x_2} f_a^2(a, x_2) f_b^2(b, x_2) f_c^2(c, x_2)) \otimes (\sum_{x_1} f_a^1(a, x_1) f_b^1(b, x_1) f_c^1(c, x_1)).$$

Algorithm VE1 is equivalent to *ci-elim-bel* along any ordering of the input variables where hidden variables are eliminated by $\otimes$ inside the buckets of input variables. In each bucket, VE1 applies first $\otimes$ to the appropriate components, and then sums over the bucket's variable. For example, VE1 computes the sum 6 as:

1. **Bucket** x_1 : $\{f_a^1(a, x_1), f_b^1(b, x_1), f_c^1(c, x_1)\} \rightarrow$
   $h^{x_1}(a, b, c) = \sum_{x_1=0} f_a^1(a, x_1) f_b^1(b, x_2) f_c^1(c, x_1) \rightarrow$ put in bucket of a.

2. **Bucket** x_2 : $\{f_a^1(a, x_2), f_b^2(b, x_2), f_c^2(c, x_2)\} \rightarrow$
   $h^{x_2}(a, b, c) = \sum_{x_1=0} f_a^2(a, x_2) f_b^2(b, x_2) f_c^2(c, x_2) \rightarrow$ put in bucket of a.

3. **Bucket** a : $\{h^{x_1}(a,b,c), h^{x_2}(a,b,c)\} \rightarrow$
   3.1. $h(a,b,c) = h^{x_1}(a,b,c) \otimes h^{x_2}(a,b,c)$
   3.2. $h^a(b,c) = \sum_a h(a,b,c) \rightarrow$
   put in bucket of b.

4. **Bucket** b : $\{h(a,b,c)\} \rightarrow h^b(c) = \sum_b h^a(b,c) \rightarrow$
   put in bucket of c.

5. **Bucket** c : $\{h^b(c)\} \rightarrow P(c) = \alpha h^b(c)$, where α is a normalizing constant.

The elimination ordering (c, b, a, x_2, x_1) used by VE1 corresponds to *ci-elim-bel* ordering $c, b, a, \{u^c_{x_1}, u^c_{x_2}\}, \{u^b_{x_1}, u^b_{x_2}\}, \{u^a_{x_1}, u^a_{x_2}\}, x_1, x_2$. Note, that since the input ordering for VE1 mentions only the input variables, hidden variable elimination ordering is restricted. For example, the hidden variables $u^a_{x_1}$, $u^b_{x_1}$, $u^c_{x_1}$ cannot be eliminated before both x_1 and x_2. Therefore, the complexity of VE1 on this network along the ordering b, a, x_1, x_2 is $O(d^6)$, while *ci-elim-bel* along the ordering $b, a, x_1, x_2, \{u^c_{x_1}, u^c_{x_2}\}, \{u^b_{x_1}, u^b_{x_2}\}, \{u^a_{x_1}, u^a_{x_2}\}$ effectively restores the original CPTs and yields $O(d^5)$ complexity, as discussed in the previous section.

To summarize,

Proposition 7: *Given a belief network BN and an ordering o of its variables, there exists a decomposition graph D and its ordering o' which coincides with the computation order of VE1. Therefore, the complexity of VE1 along o coincides with the complexity of ci-elim-bel along o'.* □

7 Optimization tasks: finding MPE, MAP, and MEU

Next, we investigate the impact of causal independence on the tasks of finding MPE, MAP, and MEU. It is not always possible to take advantage of causal independence in optimization problems since permuting maximization and summation is not allowed, i.e.,

$$\max_x \sum_y f(x,y) \neq \sum_y \max_x f(x,y).$$

This restricts the order of computations so that a CPT decomposition may not be exploited.

Consider the task of finding MPE.

$$MPE = \max_{x_1,\ldots,x_N} \prod_i P(x_i|pa_i) = \max_{x_1} F_1 \ldots \max_{x_N} F_N,$$

where $F_i = \prod_x P(x|pa(x))$ is the product of all probabilistic components such that either $x = x_i$, or $x_i \in pa(x)$. The bucket elimination algorithm *elim-mpe* sequentially eliminates x_i from right to left. However, the decomposition introduced by the causally-independent

CPTs within each F_j cannot be exploited. Given the causally-independent family in Figure 2a, having 3 parents c_1, c_2 and c_3,

$$MPE = \max_{c_1,c_2,c_3,e} P(c_1)P(c_2)P(c_3)P(e|c_1,c_2,c_3) =$$

$$= \max_{c_1} P(c_1) \max_{c_2} P(c_2) \max_{c_3} P(c_3) \max_{e}$$

$$\sum_{\{y_1,u_1^e|e=y_1*u_1^e\}} P(u_1^e|c_1) \sum_{\{u_2^e,u_3^e|y_1=u_2^e*u_3^e\}} P(u_2^e|c_2)P(u_3^e|c_3).$$

Maximization and summation cannot be interchanged here, so the hidden variables must be summed out before maximizing over c_1, c_2, c_3. This reconstructs the CPT on the whole family, i.e., causal independence has no effect. We will see later that causal independence can be exploited for families whose child nodes are observed.

Nevertheless, two other optimization tasks of computing MAP and MEU allow exploiting causal independence, because they also involve summation over a subset of the variables. By definition,

$$MAP = \max_{x_1,...,x_m} \sum_{x_{m+1},...,x_N} \prod_i P(x_i|pa_i),$$

where $x_1,...,x_k$ are the *hypothesis* variables. Given the same network in Figure 2a and hypothesis e,

$$MAP = \max_e \sum_{c_1,c_2,c_3} P(c_1)P(c_2)P(c_3)P(e|c_1,c_2,c_3).$$

Decomposing the causally-independent $P(e|c_1,c_2,c_3)$ and rearranging the summation order gives:

$$MPE = \max_e \sum_{c_1} P(c_1) \sum_{c_2} P(c_2) \sum_{c_3} P(c_3)$$

$$\sum_{\{y_1,u_1^e|e=y_1*u_1^e\}} P(u_1^e|c_1) \sum_{\{u_2^e,u_3^e|y_1=u_2^e*u_3^e\}} P(u_2^e|c_2)P(u_3^e|c_3) =$$

$$= \max_e \sum_{\{y_1,u_1^e|e=y_1*u_1^e\}} \sum_{\{u_2^e,u_3^e|y_1=u_2^e*u_3^e\}}$$

$$\sum_{c_1} P(c_1)P(u_1^e|c_1) \sum_{c_2} P(c_2)P(u_2^e|c_2) \sum_{c_3} P(c_3)P(u_3^e|c_3).$$

The summations over the parents c_i and the corresponding hidden variables can be rearranged in this example. Generally speaking, a decomposition of CPTs due to causal independence can be exploited when (at least some) parents in the causally-independent family are not included in the hypothesis.

Algorithm *ci-elim-map* for computing MAP in causally-independent networks is shown in Figure 9. It is similar to *ci-elim-bel*, except that a decomposition network does not

```
Algorithm ci-elim-map

Input: A decomposition network, a hypothesis $H \in X$, an evidence $e$, and
        a legal ordering $o = (Z_1, ..., Z_n)$, where $Z_1, ..., Z_m$ are hypothesis variables.
Output: An assignment $h = \arg\max_a P(H = a|e)$.
1. For  $i = n$ to $i = 1$, /* make buckets */
        For each  $x \in Z_i$, put in $bucket_i$ all network's components
            whose highest ordered variable is $x$.
2. /* Elimination */
   /* $\lambda_1, ..., \lambda_k$ are probabilistic components in $bucket_i$. */
2.1.For $i = n$ downto $m + 1$ do /* sum-out */
    • If $x = a$ is in the bucket, /* observation */
        replace $x$ by $a$ in each $\lambda_i$ and put
        the result in appropriate lower bucket.
    • else
        if $Z_i = \{x\}$ /* input variable */
          $\lambda^{Z_i} \leftarrow \sum_x \prod_j \lambda_j$.
        else /* $Z_i = \{y_l, y_k\}$, $y = y_l * y_k$. */
          $\lambda^{Z_i} \leftarrow \sum_{\{y_l, y_k | y = y_l * y_k\}} \prod_j \lambda_j$.
        Put $\lambda^{Z_i}$ in the highest bucket that mentions $\lambda^{Z_i}$'s variable.
2.2. For $i = m$ downto 1 do
     $\lambda^{Z_i} \leftarrow \max_x \prod_j \lambda_j$.
3. For  $i = 1$ to $i = m$, /* find assignment h */
        $h_i \leftarrow \arg\max_a \prod_j \lambda_j(x = a, ...)$
4. Return assignment h.
```

Figure 9: Algorithm ci-elim-map

transform the families of child nodes which are included in the hypothesis. Those families will be moralized in the decomposition graph. The hypothesis variables are placed first in the ordering and eliminated by maximiziation over $\prod_j \lambda_j$, where λ_j are the probabilistic components in the corresponding bucket.

Similarly to MAP, MEU is computed by summing over a subset of the variables and maximizing over the rest. By definition,

$$MEU = \max_{x_1,...,x_m} \sum_{x_{m+1},...,x_N} \prod_i P(x_i|pa_i)U(x_1,...,x_N),$$

where $x_1,...,x_m$ are *decision* variables (usually denoted $d_1,...,d_m$), $x_{m+1},...,x_N$ are *chance* variables, and $U(x_1,...,x_N)$ is a *utility function.* The utility is often assumed to be *decomposable*: $U(x_1,...,x_N) = \sum r(x_i)$, where $r(x_i)$ are individual utilities, or *rewards.* A belief network with decision nodes and a utility function is also called an *influence diagram.*

A bucket elimination algorithm *elim-meu* for computing the MEU was presented in [7]. It assumes that *the decision variables have no parents*, and puts them first in the ordering.

Assume that c_1 in example of Figure 2a is a decision variable, and that the utility is decomposable, $U(d,e,c_2,c_3) = r(d) + r(e) + r(c_2) + r(c_3)$. Then

$$MEU = \max_d \sum_{e,c_2,c_3} P(c_2)P(c_3)P(e|d,c_2,c_3)U(d,e,c_2,c_3).$$

Decomposition of $P(e|d,c_2,c_3)$ into pairwise sums yields:

$$\max_d \sum_e \sum_{e,c_2,c_3} P(c_2)P(c_3) \sum_{\{y_1,u_1^e|e=y_1*u_1^e\}} P(u_1^e|d)\times$$

$$\times \sum_{\{u_2^e,u_3^e|y_1=u_2^e*u_3^e\}} P(u_2^e|c_2)P(u_3^e|c_3)U(d,e,c_2,c_3).$$

Pushing the summations over c_2 and c_3 to the right produces

$$\max_d \sum_e \sum_{\{y_1,u_1^e|e=y_1*u_1^e\}} P(u_1^e|d) \sum_{\{u_2^e,u_3^e|y_1=u_2^e*u_3^e\}}$$

$$\sum_{c_2} P(c_2)P(u_2^e|c_2) \sum_{c_3} P(c_3)P(u_3^e|c_3)U(d,e,c_2,c_3).$$

Using decomposability of the utility, the last summation (over c_3) can be expressed as

$$\sum_{c_3} P(c_3)P(u_3^e|c_3)[r(d)+r(e)+r(c_2)+r(c_3)] =$$

$$= [\sum_{c_3} P(c_3)P(u_3^e|c_3)](r(d)+r(e)+r(c_2))+$$

```
**Algorithm ci-elim-meu**

Input: A decomposition network, a legal ordering $o = (Z_1, ..., Z_n)$,
       $Z_1 = \{d_1\}$,..., $Z_m = \{d_m\}$, an evidence $e$.
Output: An assignment $d^1, ..., d^m$ to decision variables that
maximizes the expected utility.
1. Initialization:
   For $i = n$ to $i = 1$ /* make buckets */
       For each $x \in Z_i$, put in $bucket_i$ all network's components
       whose highest ordered variable is $x$.
2. /* Elimination */
/* $\lambda_j$ and $\mu_k$ denote probabilistic and utility
components, correspondingly, in $bucket_i$. */
2.1. For $i = n$ down to $m+1$ do /* chance nodes */
     • If $x = a$ is in the bucket, replace $x$ by $a$ in each $\lambda_i$
         and put the result in appropriate lower bucket.
     • Else if $Z_i = \{x\}$,
         $\lambda_i \leftarrow \sum_x \Pi_j \lambda_j$.
         $\theta_i \leftarrow \sum_x \Pi_j \lambda_j \sum_k \mu_k$.
     • Else if $Z_i = \{y_l, y_k\}$ and $y = y_l * y_k$,
         $\lambda_i \leftarrow \sum_{\{y_l,y_k|y=y_l*y_k\}} \Pi_j \lambda_j$
         $\theta_i \leftarrow \sum_{\{y_l,y_k|y=y_l*y_k\}} \Pi_j \lambda_j \sum_k \mu_k$.
         Put $\lambda_i$ and $\mu_i = \theta_i / \lambda_i$ in the buckets
         of their highest-index variables.
2.2. For $i = m$ to 1 do
     /* decision nodes $Z_i = \{d\}$ */
         $\theta_i \leftarrow \Pi_j \lambda_j \sum_k \mu_k$
         $\gamma_i \leftarrow \max_d \theta_i$
         $d^i = \arg\max_d \theta_i$
     Put $\gamma_i$ in the bucket of its highest variable.
3. Return an optimal set of functions $(d_1, ...d_m)$ recorded in the decision buckets,
and the maximum expected utility $V_e$.
```

Figure 10: Algorithm ci-elim-meu

$$+\sum_{c_3} P(c_3)P(u_3^e|c_3)r(c_3) =$$

$$= [\sum_{c_3} P(c_3)P(u_3^e|c_3)][r(d)+r(e)+r(c_2)+$$

$$+\frac{\sum_{c_3} P(c_3)P(u_3^e|c_3)r(c_3)}{\sum_{c_3} P(c_3)P(u_3^e|c_3)}] =$$

$$= \lambda^{c_3}(u_3^e)[r(d)+r(e)+r(c_2)+\mu^{c_3}(u_3^e)],$$

where

$$\lambda^{c_3}(u_3^e) = \sum_{c_3} P(c_3)P(u_3^e|c_3),$$

$$\mu^{c_3}(u_3^e) = \frac{\theta_{c_3}(u_3^e)}{\lambda^{c_3}(u_3^e)}, \text{and}$$

$$\theta^{c_3} = \sum_{c_3} P(c_3)P(u_3^e|c_3)r(c_3).$$

In each bucket we will have probability components λ_i and utility components μ_j. Initially, the probabilistic components are those specified in the belief network, and the utility components are the individual rewards. We compute a new pair of λ and μ in each bucket, as demonstrated above, and place them in the appropriate lower buckets. For example, both new components $\lambda^{c_3}(u_3^e)$ and $\mu^{c_3}(u_3^e)$ will be placed in bucket $\{u_2^e, u_3^e\}$.

Algorithm *ci-elim-meu* is shown in Figure 10. Since causal independence is not defined for decision nodes, the decomposition network transforms only the families of chance nodes. As with *ci-elim-bel* and *ci-elim-map*, *ci-elim-meu* partitions into buckets all the network components (including the utility components) into buckets. Then it processes chance nodes, from last to first, computing the new λ and μ (see step 2.1 of the algorithm). The only difference between *ci-elim-meu* and *elim-meu* [7] is that the summation operation in buckets of hidden variables is different. Finally, the buckets of the decision nodes are processed by maximization and an optimal assignment to those nodes is generated.

Comparably to *ci-elim-bel*,

Theorem 2: *The complexity of ci-elim-map and ci-elim-meu is $O(Nmd^{w_o^*+1})$, where w_o^* is the induced width of a decomposition network along its ordering o.* $\Box$

The decomposition graph can be inspected to determine the benefits of causal independence before running the algorithm. As for belief updating, the induced width of the decomposition network can be as small as the induced width of the original unmoralized graph, and no larger than the induced width of the network's moral graph.

```
Algorithm hybrid-elim

Input: A Hybrid network $CBN = \{X, P, C\}$, an ordered partition
       of $X$, $o = (Z_1, ..., Z_n)$, $Z_1 = \{x_1\}$, and evidence $e$.
Output: $P(x_1|e)$.
       1. Propagate_evidence($CBN, e$).
       2. Buckets = Make_buckets($CBN, e, o$).
       3. For $i = n$ downto 1 do
              Eliminate($Z_i$) .
```

Figure 11: Algorithm **hybrid-elim**

```
Make_buckets($N, e, o$)

Input: A network $N = \{X, P, C\}$, and
       an ordering $o = (Z_1, ..., Z_n)$ of $N$.
Input: A list of $n$ buckets.
For  $i = n$ to $i = 1$, /* Partition $P$, $C$ and $e$ into buckets for each $Z_i$ */
    For each  $x \in Z_i$, put in $bucket_i$ all components in $P$, $C$, and $e$
              whose highest ordered variable is $x$.
Return buckets.
```

Figure 12: Procedure **Make_buckets**

8 Evidence propagation

Causal independence allows an efficient way of handling evidence via *evidence propagation*. For particular classes of belief networks, such as noisy-OR, this may result in exponential complexity decrease.

Note that causal independence allows transforming a belief network into one that has both probabilities and constraints (we will call such networks *hybrid*). A decomposition network contains constraints on hidden variables having the form $y = y_i * y_j$ (in transformation networks constraints are represented by deterministic CPTs).

Applying constraint propagation techniques such as *arc-consistency* to hybrid networks can significantly reduce the complexity of inference.

A general elimination scheme for hybrid networks is shown in Figure 11. An input hybrid network CBN is a triplet $\{X, P, C\}$, where X is the set of nodes, P is the set of conditional probability distributions, and C is the set of constraints. First, the procedure *Make_buckets* creates buckets in a standard way. Then the procedure *Propagate_evidence*

```
Propagate_evidence($N, e$)

Input: A hybrid network $N = \{X, P, C\}$, and evidence list $e = \{(x_i = a_i)\}$.
/* Assume $D$ is the domain of the variables, $|D| > 1$ */
1. $C = C \cup e$ /* add singleton constraints */
2. Perform arc-consistency on $(X, C)$ /* shrink domains*/
   /* $D_x$ will denote a new domain of a variable $x$. */
3. /* Restrict the probabilities to new domains */
For each probabilistic component $p(Y) \in P$, $Y \subset X$
        For each variable $x \in Y$
                If $|D_x| < |D|$ /* the domain of $x$ has shrunk */
                   restrict $p(Y)$ on $x \in D_x$.
```

Figure 13: Procedure **Propagate_evidence**

(Figure 13) performs *arc-consistency* on a subset of variables that participate in constrains. Arc-consistency may reduce the variables' domains so that the CPTs involving such variables need to be restricted to the new domains. The algorithm will assign zero probabilities to the tuples involving inconsistent domain values. Note, that arc-consistency in belief networks can never produce an empty domain, since an empty domain of a variable implies no non-zero-probability tuple exists, which contradicts the definition of probabilistic distribution. If all observations are made on regular variables, *Propagate_evidence* simply replaces observed variables by their values in the corresponding probabilistic components. However, an observation on a convergent variable may cause domain reduction of other variables (for example, given that $x = y \vee z$ in a noisy-OR network, evidence $x = 0$ implies $y = 0$ and $z = 0$). Finally, after propagating evidence, an elimination procedure will be invoked that solves a particular task such as belief update, finding MPE, MAP, or MEU.

9 Noisy-OR networks

In this section we investigate how specific properties of the operator $*$ can be exploited for particular classes of causally-independent networks, such as noisy-OR.

Consider a commonly used class of noisy-OR networks, called BN2O (Binary Node 2-layer Noisy-OR) networks. BN2O networks are used in medical database called QMR-DT [15]. A fully-connected BN2O network with k diseases and n findings is a k-n-network. The nodes d_i in the top layer represent diseases, while the bottom-layer nodes f_j correspond to findings (see Figure 14a). The assignment $f_i = 0$ is called a *negative* finding, while the assignment $f_i = 1$ is called a *positive* finding. Assuming that all finding nodes are observed

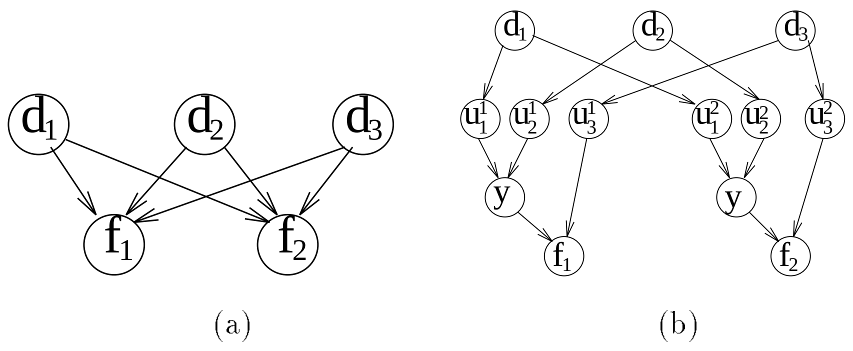


Figure 14: (a) A BN2O network; (b) its decomposition network.

(evidence is denoted by e), the algorithm *Quickscore*, proposed by [9], finds $P(d_i|e)$ in $O(exp(p))$ time, where p is the number of positive findings. Note that for $k < p$ it is faster to run a standard inference algorithm, such as $elim - bel$, which takes $O(exp(k))$ time. Therefore, the state-of-the art algorithm for BN2O networks can be written as follows:
1. if $k < p$, run elim-bel;
2. otherwise, run Quickscore.
Then the complexity of finding $P(d_i|e)$ in BN2O networks is $O(exp(min\{k,p\}))$. Let us compare it to the complexity of *ci-elim-bel*. Since fully-connected BN2O network is a k-n-network, the complexity of *ci-elim-bel* is $O(exp(min\{k,2n\}))$ (proposition 6). This complexity can be decreased by evidence propagation.

Theorem 3: *Given a BN2O network with k diseases, n findings, and evidence e that consists of p positive and $n-p$ negative findings, ci-elim-bel takes $O(exp(min\{k,2p\})$ time and space to compute $P(d_i|e)$, if the network is preprocessed by the procedure $Propagate\_evidence$.*

Proof: Propagating negative evidence causes instantiation of all hidden variables associated with negative findings in a decomposition network. Eliminating these variables cannot add induced arcs to the network, so that they can be ignored for the purpose of complexity analysis. What is left is a k-p-network, which has the complexity of $O(exp(min\{k,2p\})$. □

Example 7 shows the effect of evidence propagation in the case of negative evidence.

Example 7: Consider a BN2O network in Figure 14a. Its decomposition network is shown in Figure 14b. Let the query be $P(d_1|e)$, and let $e = \{f_1 = 0, f_2 = 0\}$. This evidence will be propagated by *Propagate_evidence* procedure in the initialization step of *ci-elim-bel*, first setting y_1, u_3^1, y_2, u_3^2 to 0, and next setting u_1^1, u_2^1, u_1^2, and u_2^2 to 0. The bucket of the query variable d_1 has to be first in the ordering since it is the query variable. Assume that all instantiated variables are processed first, then the rest of the variables are processed in the following manner:

1. **Bucket** d_3:$\{P(u_1^3=0|d_3), P(u_2^3=0|d_3)\} \rightarrow$
   $h^{d_3} = \sum_{d_3} P(u_1^3=0|d_3)P(u_2^3=0|d_3)$ is a constant.
2. **Bucket** d_2:$\{P(u_1^2=0|d_2), P(u_2^2=0|d_2)\} \rightarrow$
   $h^{d_2} = \sum_{d_2} P(u_1^2=0|d_2)P(u_2^2=0|d_3)$ is a constant.
3. **Bucket** d_1: $\{P(u_1^2=0|d_2), P(u_2^2=0|d_2)\} \rightarrow$
   $P(d_1|e) = \alpha P(u_1^2=0|d_2), P(u_2^2=0|d_2)$, where α is a normalization constant.

It is easy to show that given a BN2O network with all-negative findings, elimination is linear in the total number of findings, n.

As stated by theorem 3, the complexity of *ci-elim-bel* after evidence propagation is $O(exp(min\{k,2p\}))$. This is worse than the $O(exp(p))$ complexity of *Quickscore* for $k > p$. Let us look more closely at how *Quickscore* works. In general, any (exact) inference algorithm corresponds to a particular decomposition of the sum 1 into simpler expressions. Variable elimination decomposes this sum into a product of linear number of (possible exponential) functions. In contrast, *Quickscore* uses a decomposition into exponential number of summands, each computed in linear time. Following [9], we derive the algorithm *Quickscore* and analyze its complexity.

Consider the network in Figure 14(a), assuming positive evidence $f_1 = 1$ and $f_2 = 1$. Since

$$P(f_i=1|d_1,d_2,d_3) = 1 - P(f_i=0|d_1,d_2,d_3) = 1 - P(u_1^i=0|d_1)P(u_2^i=0|d_2)P(u_3^i=0|d_3),$$

then

$$P(d_1|f_1=1,f_2=1) = \alpha \sum_{d_2}\sum_{d_3} P(d_1)P(d_2)P(d_3)P(f_1=1|d_1,d_2,d_3)P(f_2=1|d_1,d_2,d_3) =$$

$$= \alpha \sum_{d_2}\sum_{d_3} P(d_2)P(d_3)(1-\prod_{j=1}^{3} P(u_j^1=0|d_j))(1-\prod_{j=1}^{3} P(u_j^2=0|d_j)) =$$

$$= \alpha \sum_{d_2} P(d_2) \sum_{d_3} P(d_3)[1-\prod_{j=1}^{3} P(u_j^1=0|d_j)) - \prod_{j=1}^{3} P(u_j^2=0|d_j) + \prod_{j=1}^{3} P(u_j^1=0|d_j))P(u_j^2=0|d_j)] =$$

$$= \alpha[1 - P(u_1^1=0|d_1)\sum_{d_2} P(d_2)P(u_2^1=0|d_2)\sum_{d_3} P(d_3)P(u_3^1=0|d_3) -$$

$$-P(u_1^2=0|d_1)\sum_{d_2} P(d_2)P(u_2^2=0|d_2)\sum_{d_3} P(d_3)P(u_3^2=0|d_3) +$$

$$+P(u_1^1=0|d_1)P(u_1^2=0|d_1)\sum_{d_2} P(d_2)P(u_2^1=0|d_2)P(u_2^2=0|d_2)\sum_{d_3} P(d_3)P(u_3^1=0|d_3)P(u_3^2=0|d_3),$$

where α is a normalizing constant. The number of summands is exponential in the number of positive evidence $p = 2$, while each summand is a function of a single variable.

Let us further consider a general network BN2O having k diseases and n findings, where p findings are positive and the rest of them are negative. If F^+ denotes the set of positive findings and F^- denotes the set of negative findings, then

$$P(d_1|F^+, F^-) = \alpha \sum_{d_2,...,d_k} \prod_{i=1}^{k} P(d_i) \prod_{f_j \in F^-} P(f_j = 0|d_1, ..., d_k) \prod_{f_l \in F^+} P(f_l = 1|d_1, ..., d_k) =$$

$$= \alpha \sum_{d_2,...,d_k} \prod_{i=1}^{k} P(d_i) P^- P^+,$$

where

$$P^- = \prod_{f_j \in F^-} P(f_j = 0|d_1, ..., d_k) = \prod_{f_j \in F^-} \prod_{i=1}^{k} P(u_i^j = 0|d_i),$$

and

$$P^+ = \prod_{f_l \in F^+} P(f_l = 1|d_1, ..., d_k) = \prod_{f_l \in F^+} (1 - \prod_{i=1}^{k} P(u_i^l = 0|d_i)) =$$

$$= \sum_{F' \in 2^{F^+}} (-1)^{|F'|} \prod_{i=1}^{k} \prod_{f_l \in F'} P(u_i^l = 0|d_i).$$

Therefore,

$$P^- \cdot P^+ = \sum_{F' \in 2^{F^+}} (-1)^{|F'|} \prod_{i=1}^{k} \prod_{f_j \in F' \cup F^-} P(u_i^j = 0|d_i),$$

and

$$P(d_1|F^+, F^-) = \alpha \sum_{d_2,...,d_k} \prod_{i=1}^{k} P(d_i) P^- P^+ =$$

$$\alpha \sum_{F' \in 2^{F^+}} (-1)^{|F'|} \sum_{d_2,...,d_k} \prod_{i=1}^{k} [P(d_i) \prod_{f_j \in F' \cup F^-} P(u_i^j = 0|d_i)] =$$

$$= \alpha \sum_{F' \in 2^{F^+}} (-1)^{|F'|} \prod_{i=1}^{k} \sum_{d_i} [P(d_i) \prod_{f_j \in F' \cup F^-} P(u_i^j = 0|d_i)].$$

Computing each factor

$$P(d_i) \prod_{f_j \in F' \cup F^-} P(u_i^j = 0|d_i)$$

is linear in $|F' \cup F^-| \leq n$, so that computing each

$$\prod_{i=1}^{k} \sum_{d_i} [P(d_i) \prod_{f_j \in F' \cup F^-} P(u_i^j = 0|d_i)]$$

is $O(kn)$, and the total complexity of computing $P(d_1|F^+, F^-)$ is $O(kn2^p)$, since the outer sum $\sum_{F' \in 2^{F^+}}$ has $2^{|F^+|} = 2^p$ summands.

In the next section, we will show how the approach used by *Quickscore* can improve the performance of *ci-elim-bel*.

9.1 NOR-elim-bel algorithm for noisy-OR networks

Algorithm *NOR-elim-bel*, presented below, combines both variable elimination and *Quickscore* approaches and applies them to arbitrary noisy-OR networks.

Assume a noisy-OR network defined on n nodes x_1,...,x_n, where the first k nodes are not observed. The next p nodes are observed positive findings $F^+ = \{x_{k+1}, ..., x_{k+p}\}$, and the rest of the nodes are observed negative findings $F^- = \{x_{k+p+1}, ..., x_n\}$. Then $P(x_1|e)$ can be computed as

$$P(x_1|e) = \alpha \sum_{x_2 ... x_k} \prod_{i=1}^{k} P(x_i|pa(x_i)) \prod_{x_j \in F^-} P(x_j = 0|pa(x_j)) \prod_{x_l \in F^+} P(x_l = 1|pa(x_l)) =$$

$$= \alpha \sum_{x_2 ... x_k} \prod_{i=1}^{k} P(x_i|pa(x_i)) \prod_{x_j \in F^-} \prod_{x_m \in pa(x_j)} P(u_m^j = 0|x_m) \prod_{x_l \in F^+} (1 - \prod_{x_q \in pa(x_l)} P(u_q^l = 0|x_q)) =$$

$$= \alpha \sum_{x_2 ... x_k} \prod_{i=1}^{k} P(x_i|pa(x_i)) \prod_{x_j \in F^-} \prod_{x_m \in pa(x_j)} P(u_m^j = 0|x_m) \sum_{F' \in 2^{F^+}} (-1)^{|F'|} \prod_{x_l \in F'} \prod_{x_q \in pa(x_l)} P(u_q^l = 0|x_q) =$$

$$= \alpha \sum_{F' \in 2^{F^+}} (-1)^{|F'|} [\sum_{x_2} P(x_2|pa(x_2) \prod_{\{x_j : x_j \in F' \cup F^-, x_2 \in pa(x_j)\}} P(u_2^j = 0|x_2) \times ... \times$$

$$\times \sum_{x_k} P(x_k|pa(x_k) \prod_{\{x_j : x_j \in F' \cup F^-, x_k \in pa(x_j)\}} P(u_k^j = 0|x_k)].$$

This summation contains 2^p summands. However, each summand is more complex than in the case of BN2O networks. It can be computed by variable elimination over the variables x_2,...,x_k. The total complexity can be computed as follows. Let us moralize a network and delete all evidence nodes. The resulting network is denoted BN_e, and has induced width $w^*(BN_e)$. The complexity of computing each summand is $O(N2^{w^*(BN_e)})$. The total complexity is $O(N2^{w^*(BN_e)} \cdot 2^p) = O(N2^{w^*(BN_e)+p})$. Should we decide not to exploit the positive evidence decomposition, the summation would be expressed as:

$$\alpha[\sum_{x_2} P(x_2|pa(x_2) \prod_{\{x_j : x_j \in \cup F^-, x_2 \in pa(x_j)\}} P(u_2^j = 0|x_2) \times ... \times$$

```
Algorithm NOR-elim-bel

Input: A noisy-OR network $BN = \{X, P\}$, an ordering
       $o = (x_1, ..., x_N)$ of input variables, and evidence $e$, where
       $F^-$ is the set of 0-valued nodes and $F^+$ is the set of 1-valued nodes.
Output: $P(x_1|e)$.
1. Initialization:
   1.1. Restrict $BN$ to $BN_e = \{X', P'\}$: replace observed variables
        by their values in $P$, and delete them from $X$.
   1.2. Compute $D(BN_e) = \{X', P', C, G\}$.
   1.3. Find a legal ordering $o' = (Z_1, ..., Z_n)$ of $D(BN_e)$, $Z_1 = \{x_1\}$,
        that agrees with $o$.
   1.4. Buckets = Make_buckets($D, e, o$).
   /* Put the negative-evidence components in buckets */
   1.5. For each $x_i \in X'$
            For each $x_j \in F^-$, s.t. $x_i \in pa(x_j)$
                put $P(u_i^j = 0|x_i)$ in the $bucket_i \in Buckets$.
   1.6. $Bel(x_1) \leftarrow 0$.
2. For each $F' \in 2^{F^+}$
       2.1. $Buckets' = Buckets$ /* make a copy of Buckets */
       2.2. /* Put the positive-evidence components in buckets */
       For each $x_i \in X'$
           For each $x_j \in F^+$, s.t. $x_i \in pa(x_j)$
               put $P(u_i^j = 0|x_i)$ in the $bucket_i \in Buckets'$.
           Compute $Bel'(x_1) = ci - elim - bel(Buckets', o')$.
       2.4. $Bel(x_1) \leftarrow Bel(x_1) + (-1)^{|F'|} Bel'(x_1)$.
/* End for */
3. Return $Bel(x_1)$.
```

Figure 15: Algorithm NOR-elim-bel

$$\times \sum_{x_k} P(x_k|pa(x_k) \prod_{\{x_j:x_j \in \cup F^-, x_k \in pa(x_j)\}} P(u_k^j = 0|x_k) \prod_{x_l \in F^+} P(x_l = 0|pa(x_l))],$$

and can be computed in $O(N2^{w^*(BN_{e^-})})$ time and space, where BN_{e^-} is the moralized input network without the negative evidence nodes. Therefore, we can decide upfront, comparing $w^*(BN_{e^-})$ versus $(w^*(BN_e) + p)$, whether the decomposition due positive evidence helps or hurts.

We did not exploit the causal independence in families with unknown variables $x_2,...,x_k$, assuming that each summand is computed using *elim-bel*. Instead, using *ci-elim-bel* gives the complexity $O(N2^{w^*(D_e)+p})$, where D_e is a network obtained by deleting all instantiated (singleton-domain) nodes from a decomposition network.

Algorithm *NOR-elim-bel*, shown in Figure 15, follows the computations presented above. Given a noisy-OR network, we choose *NOR-elim-bel* if $w^*(BN_e) + p < w^*(D_e)$, otherwise, *ci-elim-bel* is preferred. For each summand we eliminate $x_2,...,x_k$ by *ci-elim-bel*. The only difference is that additional factors $P(u_i^j|x_i)$ are added to the bucket of each x_i, where x_j are observed children of x_i.

10 Conclusion

In this paper we augmented bucket-elimination algorithms for probabilistic inference with features that exploit causal independence. The complexity of the modified algorithms for tasks such as belief updating, finding the maximum aposteriori hypothesis and finding the maximum expected utility is exponential in the induced width of the network's decomposition graph. This induced width does not exceed the induced width of the network's moral graph (if a proper ordering is used) and may be as small as the induced width of the unmoralized network's graph. Consequently, exploiting causal independence reduces complexity, sometimes by an exponential factor (e.g., poly-trees).

We investigated the degree to which causal independence can be exploited in belief updating, finding MPE, MAP, and MEU. We conclude that causal independence may not help finding MPE, while it helps (sometimes considerably) the other tasks.

We discussed how evidence can be exploited to speed up inference in causally-independent networks. Decomposed causally-independent networks include both probabilities and constraints. Constraints allow more efficient handling of observations known as constraint propagation (e.g., arc-consistency). We outlined a general evidence propagation technique for causally-independence networks and proposed an algorithm *NOR-elim-bel* that applies arc-consistency to noisy-OR networks.

Experimental evaluation of the proposed algorithms is a direction for future work.

References

[1] S. Arnborg, D.G. Corneil, and A. Proskurowski. Complexity of finding embedding in a k-tree. *Journal of SIAM, Algebraic Discrete Methods*, 8(2):177–184, 1987.

[2] A. Becker and D. Geiger. A sufficiently fast algorithm for finding close to optimal junction trees. In *Proc. Twelfth Conf. on Uncertainty in Artificial Intelligence*, pages 81–89, 1996.

[3] U. Bertele and F. Brioschi. *Nonserial Dynamic Programming.* Academic Press, New York, 1972.

[4] G.F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42(2–3):393–405, 1990.

[5] B. D'Ambrosio. Symbolic probabilistic inference in large BN2O networks. In *Proc. Tenth Conf. on Uncertainty in Artificial Intelligence*, pages 128–135, 1994.

[6] R. Dechter. Constraint networks. In *Encyclopedia of Artificial Intelligence.* John Wiley & Sons, 2nd edition, 1992.

[7] R. Dechter. Bucket elimination: A unifying framework for probabilistic inference. In *Proc. Twelfth Conf. on Uncertainty in Artificial Intelligence*, pages 211–219, 1996.

[8] R. Dechter and J. Pearl. Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, 34:1–38, 1987.

[9] D. Heckerman. A tractable inference algorithm for diagnosing multiple diseases. In *Proc. Fifth Conf. on Uncertainty in Artificial Intelligence*, pages 174–181, 1989.

[10] D. Heckerman and J. Breese. A new look at causal independence. In *Proc. Tenth Conf. on Uncertainty in Artificial Intelligence*, pages 286–292, 1994.

[11] F. Jensen, S. Lauritzen, and K. Olesen. Bayesian updating in recursive graphical models by local computations. *Computational Statistics Quarterly*, 4:269–282, 1990.

[12] S. Lauritzen and D. Spiegelhalter. Local computations with probabilities on graphical structures and their applications to expert systems. *Journal of the Royal Statistical Society, Series B*, 50(2):157–224, 1988.

[13] K.G. Olesen, U. Kjaerulff, F. Jensen, B. Falck, S. Andreassen, and S.K. Andersen. A Munin network for the median nerve - a case study on loops. *Applied Artificial Intelligence*, 3:384–403, 1989.

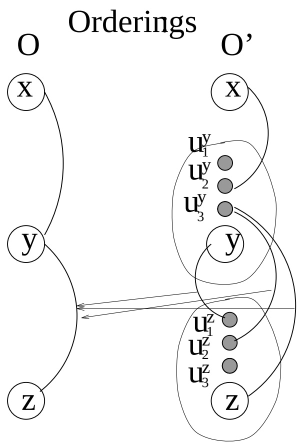


Figure 16: An extension of an ordering o of BN to an ordering o' of T such that all u_i^y immediately follow y.

[14] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann Publishers, San Mateo, California, 1988.

[15] M. Pradhan, G. Provan, B. Middleton, and M. Henrion. Knowledge engineering for large belief networks. In *Proc. Tenth Conf. on Uncertainty in Artificial Intelligence*, 1994.

[16] G. Shafer and P. Shenoy. Probability propagation. *Annals of Mathematics and Artificial Intelligence*, 2:327–352, 1990.

[17] S. Srinivas. A generalization of noisy-OR model. In *Proc. Ninth Conf. on Uncertainty in Artificial Intelligence*, pages 208–215, 1993.

[18] N.L. Zhang and D. Poole. Exploiting causal independence in Bayesian network inference. *Journal of Artificial Intelligence Research*, 5:301–328, 1996.

Appendix

Proofs

Proposition 4: *Given an (unmoralized) network BN having induced width w^* along an ordering o, and given a transformation T, there exists an extension of o to o' such that the induced width of moralized T along o', computed only with regard to the variables of BN, is not larger than w^*.*

Proof: Assume that G is the induced graph of BN along o, and that G_T is the induced graph of the *moralized* T along o'. Let x and y denote input variables, let u_i^x denote hidden variables of a convergent variable x, and let z denote both input and hidden variables in G_T. There are three types of edges in G_T: 1. an edge between two input variables (x, y); 2. an edge between two hidden variables (u_i^x, u_j^y), and 3. an edge between an input and a hidden variables, (u_i^x, y). In the last two cases, x may coincide with y.

We construct the ordering o' by placing hidden variables u_i^x immediately after x. Figure 16 demonstrates a sample ordering o' obtained from the input ordering $o = (z, y, x)$. The edges of each type are also shown. We will prove now the following statement S: *for every edge* (x, y), (u_i^x, u_j^y), *or* (u_i^x, y) *in* G_T, *where* $x \neq y$, *there is an edge* (x, y) *in* G. Obviously, it implies that $w_I^* < w^*$.

Initially, all edges in T satisfy S. Indeed, by definition of T, a hidden variable u_i^x can be connected either to its convergent variable x, or to another x's hidden variable u_j^x, or to the parents of x. Therefore, there are no edges (u_i^x, u_j^y) between hidden variables of two different convergent variables x and y, and every edge (u_i^x, y) in T corresponds to an edge (x, y) in BN. Thus, the statement S needs to be proven only for the induced edges in G_T. We do it by induction on the variables in G_T along the ordering $o' = (z_1, ..., z_m)$.

Induction basis:

The last variable in o', z_m, is either a) an input variable x or b) a hidden variable u_i^x.

a) $z_m = x$. Then x must be regular; otherwise hidden variables u_i^x would appear on top of x according to our construction of o'. Consider all possible edges that can be induced by eliminating x. Type-1 edge: an edge (y_1, y_2) between two input variables is created if there are edges (x, y_1) and (x, y_2) in T (and therefore, in BN), and y_1, y_2 precede x in the ordering o', and, therefore, in o. But then the edge (y_1, y_2) will be also induced in G along o. Type-2 edge: creating an edge $(u_i^{y_1}, u_j^{y_2})$ means that x is connected to $u_i^{y_1}$ and $u_i^{y_2}$ preceding x in o', and, therefore, x is connected to y_1 and y_2 which precede x in o, so that the edge (y_1, y_2) is induced in G. Type-3 edge: similarly, inducing an edge $(u_i^{y_1}, y_2)$ in G_T along o' corresponds to inducing an edge (y_1, y_2) in G along o.

b). $z_m = u_i^x$. As shown above, u_i^x can be connected either to other hidden variables of x, or to x, or to x's parents. Therefore, a type-1 edge $(u_i^{y_1}, u_j^{y_2})$, where $y_1 \neq y_2$, cannot be created. A type-2 edge (u_i^x, y) corresponds to the edge between x and its parent y in G. Finally, a new type-3 edge (y_1, y_2) can be created only if nodes y_1 and y_2 are both parents of x, and precede x in the orderings o'. But then y_1 and y_2 precede x in o as well, thus inducing the edge (y_1, y_2) in G.

Induction step:

Assume that statement S is correct for all edges induced by the last k variables in o'. Namely, for every edge (x, y), (u_i^x, u_j^y), or (u_i^x, y) in G_T, induced by some z_i, $i = n, ..., n-k$, there is an edge (x, y) in G. Then we can show that the same property holds for every edge induced by z_{n-k-1}.

Type-1 edge: indeed, creating an edge of type-1 between two input variables y_1 and y_2 means that those variables are already connected to z_{n-k-1} in G_T and precede it in the ordering o'. If $z_{n-k-1} = x$, where x is an input variable, then both y_1 and y_2 are connected to x in G and precede it in the ordering o, by definition of o'. Therefore, the edge (y_1, y_2) is induced in G. If $z_{n-k-1} = u_i^x$, then, by induction, the edges (u_i^x, y_1) and (u_i^x, y_2) must correspond to the edges (x, y_1) and (x, y_2) in G. Also, y_1 and y_1 must precede x in o, since they precede u_i^x in o', by definition of the ordering o'. Then the edge (y_1, y_2) is created in G.
Type-2 edge: creating a new edge $(u_i^{y_1}, u_j^{y_2})$ implies that z_{n-k-1} is already connected to both $u_i^{y_1}$ and $u_j^{y_2}$ preceding z_{n-k-1} in o'. Either $z_{n-k-1} = x$ or $z_{n-k-1} = u_i^x$; in both cases, by induction, the edges (x, y_1) and (x, y_2) have been also induced in G, and, by definition of o', both y_1 and y_1 must precede x in o, thus resulting into a new induced edge (y_1, y_2) in G. Similar argument holds for a new type-3 edge $(u_i^{y_1}, y_2)$, thus proving that, in all three cases a corresponding edge (y_1, y_2) is created in G along o. Therefore, we proved by induction that for every edge (x, y), (u_i^x, u_j^y), or (u_i^x, y) in G_T, where $x \neq y$, there is an edge (x, y) in G. This implies that the induced width of G_T restricted to the input variables is not larger than the induced width of G.

□

Proposition 5: *If no two convergent variables in a network BN have more than one common parent (e.g., Figure 4a), then the induced width of a transformation T equals the induced width of the unmoralized BN.*
Proof: We will show that, for every ordering o of BN, there exists an ordering o' of T such that $w_o^*(BN) = w_{o'}^*(T)$. Let us extend the ordering o of BN to the ordering o' of T so that the hidden variables associated with a convergent variable x are always eliminated immediately before x (namely, we always place them right on top of x in the ordering o'). Similarly to proposition 4, we can show that in such a case elimination along the ordering o' does not induce new edges between the input nodes which are not already induced in the unmoralized BN along the ordering o. Also, the hidden variables of x should follow a width-1 ordering of the corresponding binary decomposition tree. Since convergent variables x_i and x_j do not share parents, their hidden variables will never become connected to each other. Therefore, the induced width of T along o' coincides with the induced width of BN along o. □