Journal of Artificial Intelligence Research 44 (2012) 97-140 Submitted 2/2012; published 5/2012

Solving Limited Memory Influence Diagrams

Denis Deratani Mauá DENIS@IDSIA.CH
Cassio Polpo de Campos CASSIO@IDSIA.CH
Marco Zaffalon ZAFFALON@IDSIA.CH
Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA)
Galleria 2, Manno, 6928 Switzerland

Abstract

We present a new algorithm for exactly solving decision making problems represented as influence diagrams. We do not require the usual assumptions of no forgetting and regularity; this allows us to solve problems with simultaneous decisions and limited information. The algorithm is empirically shown to outperform a state-of-the-art algorithm on randomly generated problems of up to 150 variables and 10^{64} solutions. We show that these problems are NP-hard even if the underlying graph structure of the problem has low treewidth and the variables take on a bounded number of states, and that they admit no provably good approximation if variables can take on an arbitrary number of states.

1. Introduction

Influence diagrams (Howard & Matheson, 1984) are graphical models aimed at the representation of problems of decision making under uncertainty. Traditionally, they are designed to handle situations involving a single, non-forgetful decision maker. Limited memory influence diagrams (hereafter LIMIDs) are generalizations of influence diagrams that allow for decision making with limited information, as in the case of simultaneous decisions, bounded memory controllers and non-communicating cooperative agents (Zhang, Qi, & Poole, 1994; Lauritzen & Nilsson, 2001; Poupart & Boutilier, 2003; Detwarasiti & Shachter, 2005). More precisely, LIMIDs relax the *regularity* and *no forgetting* assumptions of influence diagrams, namely, that there is a complete temporal ordering over the decision variables, and that any disclosed information (i.e., decisions and observations made) is remembered and considered for future decisions. These assumptions might not only be hard to meet in some applications, but they might lead to an exponential growth in the size of policies, and consequently to intractability.

Solving a (limited memory) influence diagram refers to finding an optimal plan of action, that is, a combination of decision rules, or policies, that associate any possible observation to an action. Optimality is understood as maximizing expected utility. This task has been empirically and theoretically shown to be very hard (de Campos & Ji, 2008). In fact, we show here that solving a LIMID is NP-hard even if we admit only singly connected diagrams with bounded number of states per variable,[1] and that devising an algorithm that produces provably good approximate solutions within any fixed factor is unlike to exist even for diagrams of low treewidth.

1. A diagram is singly connected if the underlying (undirected) graph contains no cycles.

Lauritzen and Nilsson (2001) have shown that LIMIDS that satisfy certain graph-structural conditions (which no forgetting and regularity imply) can be solved exactly by a dynamic programming procedure with complexity exponential in the treewidth. Hence, solving such LIMIDs is computationally similar to performing probabilistic inference in Bayesian networks (Koller & Friedman, 2009). In fact, the single policy updating (SPU) algorithm of Lauritzen and Nilsson (2001) performs a local search in the space of policies and at each step performs a probabilistic inference to evaluate each candidate solution. However, many problems fail to meet the conditions necessary for SPU achieving optimality, and in these cases SPU might converge to a local optimum that is much inferior to the actual (global) optimum. To circumvent this problem, de Campos and Ji (2008) formulated the credal reformulation (CR) algorithm that maps a LIMID into a mixed integer linear programming problem. They showed that the CR algorithm is able to solve small problems exactly and to obtain good approximations for medium-sized problems by relaxing the integrality constraints.

We show in this paper that LIMIDs can be solved exactly by a variable elimination scheme that simultaneously propagates sets of (partial) solutions. Although the algorithm runs in exponential time in the worst case (which is just to be expected, as the problem is NP-hard), we show that for many problem instances it is possible to obtain an optimal solution efficiently by pruning solutions that are Pareto-dominated by others. At the heart of the algorithm's efficiency is the property that at any moment during variable elimination local Pareto dominance implies global Pareto dominance, that is, that a partial solution that is Pareto-dominated by another partial solution cannot be part of an optimal solution, and hence can be safely discarded. We show experimentally that the pruning of Pareto-dominated local solutions can enormously save computational resources, and enable us to compute exact solutions for much bigger problems than previous algorithms. In fact, the algorithm is orders of magnitude faster than the CR algorithm on randomly generated diagrams containing up to 150 variables and 10^{64} strategies.

The paper is organized as follows. Section 2 describes the LIMID formalism and presents new results about the complexity of solving a LIMID. The variable elimination algorithm for computing exact solutions is presented in Section 3, and evaluated in Section 4. At last, Sections 5 and 6 contain related work and a final discussion. To improve readability, some of the proofs and supporting results are given in the appendix.

2. Limited Memory Influence Diagrams

In this section, we describe the LIMID formalism, state the complexity of solving a LIMID instance, and show that any LIMID can be transformed into an equivalent (in terms of maximum expected utility) diagram whose utilities are nonnegative and decision variables have no parents. Such LIMIDs are the input of our algorithm in the next section. We start with an example of a decision problem with limited information, which we use throughout the rest of the paper to illustrate and motivate concepts. Although this example (which is essentially a team coordination problem) is rather simple, it can easily be extended to account for more realistic scenarios.

2.1 The Fire Dispatching Problem

A particular fire station contains a group of firefighters divided in three units. The fire dispatcher decides which units to dispatch for each reported accident. Each dispatched unit costs -1 utile, and units not dispatched cost no utiles. In case of fire, the higher the number of dispatched teams the higher the chances of minimum damage (which implies saving lives and preventing third-party financial losses). To make things simple, we consider that an accident can be handled either appropriately, in which case we say it is a success, or inappropriately, in which case we say it is a failure. Ideally, the dispatcher wants to maximize the chance of success while minimizing the number of dispatched teams (and hence the cost of the operation). A successful operation is rewarded with 7/2 utiles, while a failure gets zero utiles.

2.2 Variables and Domains

In the formalism of (limited memory) influence diagrams, the quantities and events of interest are represented by three distinct types of variables or nodes.[2] *Chance variables* represent events on which the decision maker has no control, such as outcomes of tests or consequences of actions. *Decision variables* represent the options available to a decision maker. Finally, *value variables* represent additive parcels of the utility associated to a state of the world. The set of all variables considered relevant for a problem is denoted by $\mathcal{U}$. Each variable X in $\mathcal{U}$ has an associated *domain* Ω_X, which is the finite non-empty set of values that X can assume. The elements of Ω_X are called *states*. We assume the existence of the *empty domain* $\Omega_\emptyset \triangleq \{\boldsymbol{\lambda}\}$, which contains a single element $\boldsymbol{\lambda}$ which is not in any other domain. Decision and chance variables are assumed to have domains different from the empty domain, whereas value variables are always associated to the empty domain.

In the fire dispatching problem, we can represent the act of dispatching or not the unit i by a decision variable T_i; hence we have three decision variables T_1, T_2, and T_3 with domains $\Omega_{T_1} = \Omega_{T_2} = \Omega_{T_3} = \{\boldsymbol{a}, \boldsymbol{w}\}$, where $\boldsymbol{a}$ stands for act and means the unit is dispatched, while $\boldsymbol{w}$ stands for wait and means the unit is not dispatched. The outcome of the incident after the assignment of units is represented by a binary chance variable O with domain $\Omega_O = \{\boldsymbol{s}, \boldsymbol{f}\}$ (representing success and failure, respectively), and evaluated by a value variable V (which is associated to $\Omega_\emptyset$). There are also individual costs per unit dispatched, which are modeled by three value variables V_1, V_2 and V_3. The set of relevant variables for the problem is then $\mathcal{U} = \{T_1, V_1, T_2, V_2, T_3, V_3, O, V\}$.

The domain Ω_x of a set of variables $x = \{X_1, \ldots, X_n\} \subseteq \mathcal{U}$ is given by the Cartesian product $\Omega_{X_1} \times \cdots \times \Omega_{X_n}$ of the variable domains. Thus, an element $\boldsymbol{u} \in \Omega_{\mathcal{U}}$ defines a state of the world, that is, a realization of all actions and events of interest. If x and y are sets of variables such that $y \subseteq x \subseteq \mathcal{U}$, and $\boldsymbol{x}$ is an element of the domain Ω_x, we write $\boldsymbol{x}^{\downarrow y}$ to denote the projection of $\boldsymbol{x}$ onto the smaller domain Ω_y, that is, $\boldsymbol{x}^{\downarrow y} \in \Omega_y$ contains only the components of $\boldsymbol{x}$ that are compatible with the variables in y. By convention, $\boldsymbol{x}^{\downarrow \emptyset} \triangleq \lambda$. The *cylindrical extension* of $\boldsymbol{y} \in \Omega_y$ to Ω_x is the set $\boldsymbol{y}^{\uparrow x} \triangleq \{\boldsymbol{x} \in \Omega_x : \boldsymbol{x}^{\downarrow y} = \boldsymbol{y}\}$. Often, we write $X_1 \cdots X_n$ to denote the set $\{X_1, \ldots, X_n\}$ and, if clear from the context, X to denote the singleton $\{X\}$. For instance, if $x = \{T_1, O\}$ and $y = \{T_1\}$, then $\Omega_x =$

2. We make no distinction between a node in the graphical representation of a decision problem and its corresponding variable.

$\{(\boldsymbol{a},\boldsymbol{s}),(\boldsymbol{w},\boldsymbol{s}),(\boldsymbol{a},\boldsymbol{f}),(\boldsymbol{w},\boldsymbol{f})\}$. Also, if $\boldsymbol{x}=(\boldsymbol{w},\boldsymbol{s})\in\Omega_x$ then $\boldsymbol{x}^{\downarrow y}=\boldsymbol{w}$ and $\boldsymbol{x}^{\downarrow O}=\boldsymbol{s}$. The cylindrical extension of $\boldsymbol{s}\in\Omega_O$ to Ω_x is given by $\boldsymbol{s}^{\uparrow x}=\{(\boldsymbol{a},\boldsymbol{s}),(\boldsymbol{w},\boldsymbol{s})\}$.

2.3 Operations Over Real-Valued Functions

Some operations over real-valued functions need to be defined. Let f and g be functions over domains Ω_x and Ω_y, respectively. The product fg is defined as the function over the domain $\Omega_{x\cup y}$ such that $(fg)(\boldsymbol{w})=f(\boldsymbol{w}^{\downarrow x})g(\boldsymbol{w}^{\downarrow y})$ for any $\boldsymbol{w}$ of its domain. Sum of functions is defined analogously: $(f+g)(\boldsymbol{w})=f(\boldsymbol{w}^{\downarrow x})+g(\boldsymbol{w}^{\downarrow y})$. Notice that product and sum of functions are associative and commutative, and that product distributes over sum, that is, $fg=gf$, $f+g=g+f$, and $f(g+h)=fg+fh$. If f is a function over Ω_x, and $y\subseteq\mathcal{U}$, the *sum-marginal* $\sum_y f$ returns a function over $\Omega_{x\setminus y}$ such that for any element $\boldsymbol{w}$ of its domain we have $(\sum_y f)(\boldsymbol{w})=\sum_{\boldsymbol{x}\in\boldsymbol{w}^{\uparrow x}} f(\boldsymbol{x})$. Notice that if $y\cap x=\emptyset$, then $\sum_y f=f$. Also, the sum-marginal operation inherits commutativity and associativity from addition of real numbers, and hence $\sum_{x\cup y} f=\sum_{x\setminus y}\sum_y f=\sum_{y\setminus x}\sum_x f$.

If $\{f_x^{\boldsymbol{y}}\}_{\boldsymbol{y}\in\Omega_y}$ is a set containing functions $f_x^{\boldsymbol{y}}$ of domain Ω_x, one for each element of Ω_y, we write f_x^y to denote the function that for all $\boldsymbol{w}\in\Omega_{x\cup y}$ satisfies $f_x^y(\boldsymbol{w})=f_x^{\boldsymbol{w}^{\downarrow y}}(\boldsymbol{w}^{\downarrow x})$. For instance, if X and Y are two binary-valued variables with domains $\Omega_X=\{\boldsymbol{x}_1,\boldsymbol{x}_2\}$ and $\Omega_Y=\{\boldsymbol{y}_1,\boldsymbol{y}_2\}$, and $f_X^{\boldsymbol{y}_1}$ and $f_X^{\boldsymbol{y}_2}$ are two functions over Ω_X such that $f_X^{\boldsymbol{y}_1}(\boldsymbol{x_1})=1/2$, $f_X^{\boldsymbol{y}_1}(\boldsymbol{x_2})=1/2$, $f_X^{\boldsymbol{y}_2}(\boldsymbol{x_1})=0$ and $f_X^{\boldsymbol{y}_2}(\boldsymbol{x_2})=1$, then the function f_X^Y is such that

$$\begin{aligned} f_X^Y(\boldsymbol{x}_1,\boldsymbol{y}_1)&=f_X^{\boldsymbol{y}_1}(\boldsymbol{x}_1)\,,=1/2 & f_X^Y(\boldsymbol{x}_2,\boldsymbol{y}_1)&=f_X^{\boldsymbol{y}_1}(\boldsymbol{x}_2)=1/2\,,\\ f_X^Y(\boldsymbol{x}_1,\boldsymbol{y}_2)&=f_X^{\boldsymbol{y}_2}(\boldsymbol{x}_1)\,,=0 & f_X^Y(\boldsymbol{x}_2,\boldsymbol{y}_2)&=f_X^{\boldsymbol{y}_2}(\boldsymbol{x}_2)=1\,. \end{aligned}$$

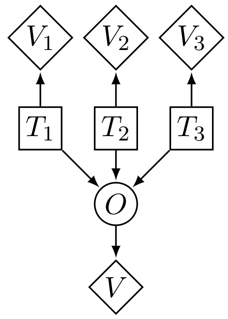
When clear from the context, we write 1 to denote a function that returns one to all values in its domain and 0 to denote a function that returns always zero. For $\tilde{\boldsymbol{x}}\in\Omega_x$, the indicator function $I_{\tilde{\boldsymbol{x}}}$ returns one if $\boldsymbol{x}=\tilde{\boldsymbol{x}}$ and zero otherwise.

If f and g are functions over a domain Ω_x and k is a real number, the expressions $f\geq g$ and $f=k$ denote that $f(\boldsymbol{x})\geq g(\boldsymbol{x})$ and $f(\boldsymbol{x})=k$, respectively, for all $\boldsymbol{x}\in\Omega_x$ (e.g., in the previous example we have $f_X^{\boldsymbol{y}_1}=1/2$). Finally, any function over a domain containing a single element (e.g., the empty domain) is identified with the real number it returns.

2.4 Definition

A LIMID $\mathcal{L}$ consists of a direct acyclic graph (DAG) over the set of variables $\mathcal{U}$ annotated with variable types (decision, chance and value), together with a collection of (conditional) probability mass functions (one per chance variable) and utility functions (one per value variable). The value nodes in the graph are assumed to have no children. The precise meaning of the arcs varies according to the type of node to which they point. Arcs entering chance and value nodes denote stochastic and functional dependency, respectively; arcs entering decision nodes describe information awareness at the time the decision is made.

For each variable X in $\mathcal{U}$, we denote by $\mathtt{pa}_X$ the set of parents of X, that is, the set of nodes of from which there is an arc pointing to X. Similarly, we let $\mathtt{ch}_X$ denote the set of children of X (i.e., nodes to which there is an arc from X), and $\mathtt{fa}_X\triangleq\mathtt{pa}_X\cup\{X\}$ denote its family. We let $\mathcal{C}$, $\mathcal{D}$ and $\mathcal{V}$ be a partition of $\mathcal{U}$ in sets of chance, decision and value variables, respectively. Each chance variable C in $\mathcal{C}$ has an associated set $\{p_C^{\boldsymbol{\pi}}:\boldsymbol{\pi}\in\Omega_{\mathtt{pa}_C}\}$



$$u_{V_1}(\boldsymbol{a}) = u_{V_2}(\boldsymbol{a}) = u_{V_3}(\boldsymbol{a}) = -1$$
$$u_{V_1}(\boldsymbol{w}) = u_{V_2}(\boldsymbol{w}) = u_{V_3}(\boldsymbol{w}) = 0$$
$$p_O^{T_1,T_2,T_3}(\boldsymbol{s}, \boldsymbol{t}_1, \boldsymbol{t}_2, \boldsymbol{t}_3) = I_{(\boldsymbol{a},\boldsymbol{a},\boldsymbol{a})}$$
$$u_V(\boldsymbol{s}) = 7/2$$
$$u_V(\boldsymbol{f}) = 0$$

Figure 1: A LIMID representing the fire dispatching problem.

of (conditional) probability mass functions $p_C^{\boldsymbol{\pi}}$ quantifying the decision maker's beliefs about states $\boldsymbol{x} \in \Omega_C$ conditional on a state $\boldsymbol{\pi}$ of its parents (if C has no parents, it has a single probability mass function assigned). Using the notation introduced in the previous section, we equivalently represent the set of probability mass functions associated to a variable C by a function $p_C^{\mathtt{pa}_C}$. We assume any chance variable $X \in \mathcal{C}$ to be stochastically independent of its non-descendant non-parents given its parents. Each value variable $V \in \mathcal{V}$ is associated with a real-valued utility function u_V over $\Omega_{\mathtt{pa}_V}$, which quantifies the (additive) contribution of the states of its parents to the overall utility. Thus, the overall utility of a state $\boldsymbol{x} \in \Omega_{\mathcal{C}\cup\mathcal{D}}$ is given by the sum of utility functions, that is, $u(\boldsymbol{x}) = \sum_{V\in\mathcal{V}} u_V(\boldsymbol{x}^{\downarrow\mathtt{pa}_V})$.

2.5 A LIMID for the Fire Dispatching Problem

Figure 1 depicts a LIMID for the fire dispatching problem. In the graph, chance, decision and value variables are represented by ovals, rectangles and diamonds, respectively. The value variables V_1, V_2 and V_3 have associated utility functions u_{V_1}, u_{V_2} and u_{V_3}, respectively, representing the cost per unit dispatched. The utility of the outcome is quantified by the function u_V associated to the value variable V. The chance variable O has associated a function $p_O^{T_1,T_2,T_3}$ that quantifies the conditional probabilities $P(O = \boldsymbol{o}|T_1 = \boldsymbol{t}^{\downarrow T_1}, T_2 = \boldsymbol{t}^{\downarrow T_2}, T_3 = \boldsymbol{t}^{\downarrow T_3})$ of success ($\boldsymbol{o} = \boldsymbol{s}$) or failure ($\boldsymbol{o} = \boldsymbol{f}$) given a joint decision $\boldsymbol{t} \in \Omega_{T_1,T_2,T_3}$. According to the model in the figure, dispatching the three units results in certain success, whereas dispatching less than three units leads to a failure.

2.6 Policies and Strategies

For any decision variable $D \in \mathcal{D}$ with at least one parent, a *policy* δ_D specifies an action for each possible state configuration of its parents, that is, $\delta_D : \Omega_{\mathtt{pa}_D} \to \Omega_D$. If D has no parents, then δ_D is a state in Ω_D. The set of all policies δ_D for a variable D is denoted by Δ_D. For instance, a policy δ_{T_1} for the first unit in the running example is a state from Ω_{T_1}. The space of policies for T_1 is given by $\Delta_{T_1} = \{\boldsymbol{a}, \boldsymbol{w}\}$.

Let $\Delta \triangleq \times_{D\in\mathcal{D}} \Delta_D$ denote the space of possible combination of policies. An element $s = (\delta_D)_{D\in\mathcal{D}} \in \Delta$ is said to be a *strategy* for $\mathcal{L}$. Given a policy δ_D and a state $\boldsymbol{\pi} \in \Omega_{\mathtt{pa}_D}$, let $p_D^{\boldsymbol{\pi}}$ denote a probability mass function for D conditional on $\mathtt{pa}_D = \boldsymbol{\pi}$ such that $p_D^{\boldsymbol{\pi}} = I_{\delta_D(\boldsymbol{\pi})}$. If D has no parents, then $p_D = I_{\delta_D}$ is an unconditional probability mass function over Ω_D.

To simplify notation, we sometimes write $p_D^{\mathtt{pa}_D}$ irrespective of whether D has any parent. There is a one-to-one correspondence between functions $p_D^{\mathtt{pa}_D}$ and policies $\delta_D \in \Delta_D$ such that specifying a policy δ_D is equivalent to specifying $p_D^{\mathtt{pa}_D}$ and vice-versa. We denote the set of all functions $p_D^{\mathtt{pa}_D}$ obtained in this way by $\mathcal{P}_D$. So, for instance, $\mathcal{P}_{T_1} = \{I_{\boldsymbol{a}}, I_{\boldsymbol{w}}\}$.

A strategy s induces a joint probability mass function over the variables in $\mathcal{C} \cup \mathcal{D}$ by

$$p_s \triangleq \prod_{C \in \mathcal{C}} p_C^{\mathtt{pa}_C} \prod_{D \in \mathcal{D}} p_D^{\mathtt{pa}_D}\,, \tag{1}$$

and has an associated expected utility

$$\mathrm{E}_s[\mathcal{L}] \triangleq \sum_{\mathcal{C} \cup \mathcal{D}} p_s \sum_{V \in \mathcal{V}} u_V\,. \tag{2}$$

Notice that the two sums in Eq. (2) have different semantics. The outer (leftmost) sum denotes the sum-marginal of the set of variables $\mathcal{C} \cup \mathcal{D}$, whereas the inner (rightmost) denotes the overall utility function over $\bigcup_{V \in \mathcal{V}} \mathtt{pa}_V$ that results from the sum of functions u_V.

In the fire dispatching problem, there are eight possible strategies consisting of a decision to act or wait for each of the units, for example, $s = (\delta_{T_1}, \delta_{T_2}, \delta_{T_3}) = (\boldsymbol{a}, \boldsymbol{w}, \boldsymbol{a})$ is a possible strategy. The policy $\delta_{T_1} = \boldsymbol{a}$ that dispatches unit T_1 induces a probability mass function $p_{T_1} = I_{\boldsymbol{a}}$ on Ω_{T_1}. Likewise, the policy $\delta_{T_2} = \boldsymbol{w}$ induces a function $p_{T_2} = I_{\boldsymbol{w}}$, and the policy $\delta_{T_3} = \boldsymbol{a}$ induces $p_{T_3} = I_{\boldsymbol{a}}$. The strategy $s = (\boldsymbol{a}, \boldsymbol{w}, \boldsymbol{a})$ then induces the joint probability mass function such that for $\boldsymbol{x} \in \Omega_{O,T_1,T_2,T_3}$

$$p_s(\boldsymbol{x}) = p_O^{T_1,T_2,T_3}(\boldsymbol{x}) p_{T_1}(\boldsymbol{x}^{\downarrow T_1}) p_{T_2}(\boldsymbol{x}^{\downarrow T_2}) p_{T_3}(\boldsymbol{x}^{\downarrow T_3})\,,$$

and has an expected utility of

$$\begin{aligned} E_s[\mathcal{L}] &= \sum_{O,T_1,T_2,T_3} p_s \left[u_{V_1} + u_{V_2} + u_{V_3} + u_V\right] \\ &= \sum_{\boldsymbol{x} \in \Omega_{O,T_1,T_2,T_3}} p_s(\boldsymbol{x}) \left[u_{V_1}(\boldsymbol{x}^{\downarrow T_1}) + u_{V_2}(\boldsymbol{x}^{\downarrow T_2}) + u_{V_3}(\boldsymbol{x}^{\downarrow T_3}) + u_V(\boldsymbol{x}^{\downarrow O})\right] = -2\,. \end{aligned}$$

The optimal strategy $s^* = (\boldsymbol{a}, \boldsymbol{a}, \boldsymbol{a})$ that dispatches all units, on the other hand, has an expected utility of $E_{s^*}[\mathcal{L}] = 1/2$.

2.7 Theoretical Complexity

The *treewidth* of a graph measures its resemblance to a tree and is given by the number of vertices in the largest clique of the corresponding triangulated moral graph minus one (Bodlaender, 1996). As in Bayesian networks, the complexity of solving a LIMID is strongly affected by its treewidth. Given a LIMID $\mathcal{L}$ of treewidth ω, we can evaluate the expected utility of any given strategy s in time and space at most exponential in ω (Koller & Friedman, 2009). Hence, if ω is bounded by a constant, computing $\mathrm{E}_s[\mathcal{L}]$ takes (at most) polynomial time in the input size.

The primary task of a LIMID is to find a strategy s^* with maximal expected utility, that is, to find $s^* \in \Delta$ such that

$$\mathrm{E}_s[\mathcal{L}] \leq \mathrm{E}_{s^*}[\mathcal{L}] \quad \text{for all } s. \tag{3}$$

The value $\mathrm{E}_{s^*}[\mathcal{L}]$ is called the *maximum expected utility* of $\mathcal{L}$ and it is denoted by $\mathrm{MEU}[\mathcal{L}]$. For most real problems, enumerating all the strategies is prohibitively costly. In fact, computing the MEU in bounded treewidth diagrams is NP-hard (de Campos & Ji, 2008), and, as the following result implies, it remains NP-hard in even simpler LIMIDs.

Theorem 1. *Given a singly connected LIMID with treewidth equal to two, and with variables having at most three states, deciding whether there is a strategy with expected utility greater than a given k is NP-complete.*

The proof, based on a reduction from the partition problem (Garey & Johnson, 1979), is given in the appendix.

Under the usual assumptions of complexity theory, when a problem is NP-hard to solve the best available options are (i) trying to devise an algorithm that runs efficiently on many instances but has exponential worst-case complexity, or (ii) trying to develop an approximation algorithm that for all instances provides in polynomial time a solution that is provably within a certain range of the optimal solution. In Section 3, we take option (i), and present an algorithm that efficiently computes optimal solutions for many LIMIDs, but runs in exponential time for many others. In the following we state a result that suggests that alternative (ii) is most likely unfeasible, even if we consider only diagrams of bounded treewidth.

Given $\rho > 1$, a ρ-approximation algorithm (for solving a LIMID) obtains a strategy s such that

$$\frac{\mathrm{MEU}[\mathcal{L}]}{\rho} \leq \mathrm{E}_s[\mathcal{L}]\,. \tag{4}$$

If we set $\rho = 1/(1-\epsilon)$, for $0 < \epsilon < 1$, then a ρ-approximation algorithm finds a solution whose induced relative error is at most ϵ, that is,

$$\frac{\mathrm{MEU}[\mathcal{L}] - \mathrm{E}_s[\mathcal{L}]}{\mathrm{MEU}[\mathcal{L}]} \leq \epsilon\,. \tag{5}$$

The following result indicates that provably good approximation algorithms do not exist unless P=NP.

Theorem 2. *Given a singly connected LIMID $\mathcal{L}$ with bounded treewidth, (unless P=NP) there is no polynomial time ρ-approximation algorithm, for any $1 < \rho < 2^\theta$, where θ is the number of numerical parameters (i.e., probabilities and utilities) required to specify $\mathcal{L}$.*

We defer the proof to the appendix. The result asserts that any algorithm that finds solutions to LIMIDs in polynomial time cannot guarantee a relative error smaller than $1 - 2^{-\theta}$, even if the set of inputs is restricted to LIMIDs of bounded treewidth. Hence, any polynomial-time algorithm for LIMIDs must eventually produce very poor solutions, with relative error close to one for large models. An exception is when both treewidth and the number of states per variable are bounded. In such cases, we have shown constructively in an early work (Mauá, de Campos, & Zaffalon, 2011) that there is a ρ-approximation algorithm that runs in polynomial time.

2.8 Constraining LIMIDs to Nonnegative Utilities

In principle, the utilities associated to value variables in a LIMID can take on any real value. This complicates the ordering between functions that we use in the algorithm we devise here. Fortunately, we can easily and efficiently transform any LIMID $\mathcal{L}$ into an equivalent LIMID $\mathcal{L}'$ where all utilities are nonnegative and whose optimal strategies s^* are also optimal strategies in $\mathcal{L}$. Moreover, obtaining $\mathrm{E}_s[\mathcal{L}]$ from $\mathrm{E}_s[\mathcal{L}']$ for any strategy s is straightforward.

Let $\mathcal{L}$ be a LIMID and let k denote the smallest utility value associated to any of the value variables, that is, for all $V \in \mathcal{V}$ it follows that $k \leq u_V$, and there is V such that $u_V(\boldsymbol{x}) = k$ for some $\boldsymbol{x} \in \Omega_{\mathtt{pa}_V}$. The following transformation generates a new LIMID $\mathcal{L}'$ whose value variables are associated only to nonnegative values.

Transformation 3. *For each value variable $V \in \mathcal{V}$, substitute its associated utility function u_V with a new utility function $u'_V = u_V - k$.*

The transformation shifts the utility functions so that $u_V \geq 0$, and makes $u_V(\boldsymbol{x}) = 0$ for at least one V and $\boldsymbol{x} \in \Omega_{\mathtt{pa}_V}$. Since it affects only value variables, any strategy for $\mathcal{L}$ (the LIMID before the transformation) is also a valid strategy for $\mathcal{L}'$ (the transformed LIMID). The expected utilities of a strategy s in $\mathcal{L}$ and $\mathcal{L}'$ are related according to the following result.

Proposition 4. *For any strategy s, $\mathrm{E}_s[\mathcal{L}] = \mathrm{E}_s[\mathcal{L}'] + k|\mathcal{V}|$.*

Proof. The expected utility of s with respect to $\mathcal{L}'$ is given by

$$
\begin{aligned}
E_s[\mathcal{L}'] &= \sum_{\boldsymbol{x}} p_s(\boldsymbol{x}) \sum_{V} u'_V(\boldsymbol{x}^{\downarrow \mathtt{pa}_V}) \\
&= \sum_{\boldsymbol{x}} p_s(\boldsymbol{x}) \sum_{V} [u_V(\boldsymbol{x}^{\downarrow \mathtt{pa}_V}) - k] \\
&= \sum_{\boldsymbol{x}} p_s(\boldsymbol{x}) \sum_{V} u_V(\boldsymbol{x}^{\downarrow \mathtt{pa}_V}) - k|\mathcal{V}| \sum_{\boldsymbol{x}} p_s(\boldsymbol{x}) \\
&= \mathrm{E}_s[\mathcal{L}] - k|\mathcal{V}| \,,
\end{aligned}
$$

where the last step follows from $\sum_{\boldsymbol{x}} p_s(\boldsymbol{x}) = 1$. □

An optimal strategy s^* for $\mathcal{L}$ satisfies $\mathrm{E}_{s^*}[\mathcal{L}] \geq \mathrm{E}_s[\mathcal{L}]$ for all s, and hence Proposition 4 ensures that $\mathrm{E}_{s^*}[\mathcal{L}] = \mathrm{E}_{s^*}[\mathcal{L}'] + k|\mathcal{V}| \geq \mathrm{E}_s[\mathcal{L}'] + k|\mathcal{V}| = \mathrm{E}_s[\mathcal{L}]$, which implies that s^* is also an optimal strategy for $\mathcal{L}'$. Similarly, if s^* is an optimal strategy for $\mathcal{L}'$, we have by the same proposition that $\mathrm{E}_{s^*}[\mathcal{L}'] = \mathrm{E}_{s^*}[\mathcal{L}] - k|\mathcal{V}| \geq \mathrm{E}_s[\mathcal{L}] - k|\mathcal{V}| = \mathrm{E}_s[\mathcal{L}']$ for all s, and therefore s^* is also optimal for $\mathcal{L}$. The following corollary summarizes these results.

Corollary 5. *A strategy s for $\mathcal{L}'$ is an optimal strategy if and only if it is also an optimal strategy for $\mathcal{L}$.*

Consider the running example once more. The smallest utility value is $k = -1$. The utilities associated to the value variables of the transformed LIMID $\mathcal{L}'$ are given by

$$
\begin{aligned}
u'_V(\boldsymbol{s}) &= 9/2 & \qquad u'_V(\boldsymbol{f}) &= 1 \\
u'_{V_1}(\boldsymbol{a}) &= 0 & u'_{V_1}(\boldsymbol{w}) &= 1 \\
u'_{V_2}(\boldsymbol{a}) &= 0 & u'_{V_2}(\boldsymbol{w}) &= 1 \\
u'_{V_3}(\boldsymbol{a}) &= 0 & u'_{V_3}(\boldsymbol{w}) &= 1\,.
\end{aligned}
$$

The strategy $s = (\boldsymbol{a}, \boldsymbol{w}, \boldsymbol{a})$ has expected utility $\mathrm{E}_s[\mathcal{L}'] = \mathrm{E}_s[\mathcal{L}] - k|\mathcal{V}| = -2 - (-1)4 = 2$. The optimal strategy $s^* = (\boldsymbol{a}, \boldsymbol{a}, \boldsymbol{a})$ obtains $E_{s^*}[\mathcal{L}'] = 9/2$.

In the rest of the paper, we consider only LIMIDs with nonnegative utilities, which due to Proposition 4 does not incur any loss of generality.

2.9 Decision Nodes with Many Parents Versus Parentless Decision Nodes

A policy for a decision variable with no parents corresponds to a choice of one of its states. Hence, the space of policies of such nodes contains a number of policies that is polynomial in the input. On the other hand, the cardinality of the space of policies for decision nodes with many parents is exponential in the number of states of the parents. To see this, consider a ten-state decision variable D. If D has no parents then the space of policies Δ_D contains 10 policies. However, if D has four ternary parent nodes, the space Δ_D contains $10^{3^4} = 10^{81}$ policies.

One might then wonder whether LIMIDs whose decision nodes have many parents are more difficult to solve than LIMIDs with parentless decision nodes. We will show that, at least from a theoretical perspective, this is not the case, and that any LIMID can be efficiently mapped into a MEU-equivalent LIMID where decision nodes have no parents. We then show how an optimal strategy for the original diagram can be produced from an optimal strategy for the transformed diagram. This is particularly relevant for algorithms that search the space of policies, as is the case of the algorithm we devise here, and it allow us, without loss of generality, to focus on LIMIDs whose decision nodes have no parents.

Before formally describing the transformation and showing that it produces a diagram with equal MEU, let us first give an idea of how and why it works. To this end, consider a LIMID $\mathcal{L}$ and a decision node D with at least one parent (e.g., the diagram in Figure 2(a)), and let $\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi_m}$ denote the configurations in $\Omega_{\mathtt{pa}_D}$. A policy δ_D maps a configuration $\boldsymbol{\pi}_i$ to a decision $\boldsymbol{d} \in \Omega_D$. A function $p_D^{\mathtt{pa}_D}$ associated to a policy δ_D can be seen as a set of probability mass functions $p_{T_1}, \ldots, p_{T_m}$ where $p_{T_i} = p_D^{\boldsymbol{\pi}_i} = I_{\delta_D(\boldsymbol{\pi}_i)}$, that is, each function p_{T_i} represents a choice of a state of D for a fixed configuration $\boldsymbol{\pi}_i$ of the parents. Recall that a policy associated to a parentless variable is simply a choice of a state. The transformation replaces the decision variable D with m decision variables $T_1, \ldots, T_m$ and m chance variables $X_1, \ldots, X_m$ such that each policy δ_{T_i} corresponds to a decision $\delta_D(\boldsymbol{\pi_i})$ of the original variable's policy (see diagram in Figure 2(b)). The chain $X_1 \to \cdots \to X_m$ of chance variables is responsible for making only the policy δ_{T_i} "active" when the parents assume configuration $\boldsymbol{\pi}_i$, as it occurs with δ_D, by either "blocking" or "allowing" information to flow according to the value of the parents of D. Thus, the parents of D act as a selector that

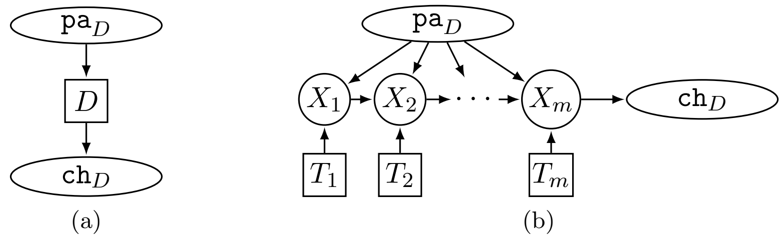


Figure 2: A piece of a diagram before (a) and after (b) Transformation 6.

decides which of the probability mass functions p_{T_i} associated to decision nodes $T_1, \dots, T_m$ is going to be used, so that the transformed diagram acts as the original one. The probability mass functions of $p_{X_i}^{X_{i-1},T_i,\mathtt{pa}_D}$ are set to ensure that $X_i = T_i$ if $\mathtt{pa}_D = \boldsymbol{\pi}_i$ and $X_i = X_{i-1}$ otherwise.

Transformation 6. *Consider a LIMID $\mathcal{L}$ and a decision node D with at least one parent, and let $\boldsymbol{\pi}_1, \dots, \boldsymbol{\pi}_m$ denote the configurations in $\Omega_{\mathtt{pa}_D}$. Remove D and add $m = |\Omega_{\mathtt{pa}_D}|$ chance nodes $X_1, \dots, X_m$ and m decision nodes $T_1, \dots, T_m$ with domains $\Omega_{X_i} = \Omega_{T_i} = \Omega_D$ (for $i = 1, \dots, m$). Add an arc from every parent of D to each of $X_1, \dots, X_m$, an arc from every X_i to X_{i+1}, with $i < m$, and an arc from every T_i to X_i , $i = 1, \dots, m$. Add an arc from X_m to each child of D. Associate to X_1 the function*

$$p_{X_1}^{T_1,\mathtt{pa}_D}(\boldsymbol{x}) = \begin{cases} 1, & \text{if } \boldsymbol{x}^{\downarrow \mathtt{pa}_D} = \boldsymbol{\pi}_1 \text{ and } \boldsymbol{x}^{\downarrow X_1} = \boldsymbol{x}^{\downarrow T_1} \\ 0, & \text{if } \boldsymbol{x}^{\downarrow \mathtt{pa}_D} = \boldsymbol{\pi}_1 \text{ and } \boldsymbol{x}^{\downarrow X_1} \neq \boldsymbol{x}^{\downarrow T_1} \\ 1/m & \text{if } \boldsymbol{x}^{\downarrow \mathtt{pa}_D} \neq \boldsymbol{\pi}_1 \,. \end{cases}$$

For each node X_i, $i = 2, \dots, m$, associate a function

$$p_{X_i}^{X_{i-1},T_i,\mathtt{pa}_D}(\boldsymbol{x}) = \begin{cases} 1, & \text{if } (\boldsymbol{x}^{\downarrow \mathtt{pa}_D} \neq \boldsymbol{\pi}_i \text{ and } \boldsymbol{x}^{\downarrow X_i} = \boldsymbol{x}^{\downarrow X_{i-1}}) \\ 0, & \text{if } (\boldsymbol{x}^{\downarrow \mathtt{pa}_D} \neq \boldsymbol{\pi}_i \text{ and } \boldsymbol{x}^{\downarrow X_i} \neq \boldsymbol{x}^{\downarrow X_{i-1}}) \\ 1, & \text{if } (\boldsymbol{x}^{\downarrow \mathtt{pa}_D} = \boldsymbol{\pi}_i \text{ and } \boldsymbol{x}^{\downarrow X_i} = \boldsymbol{x}^{\downarrow T_i}) \\ 0, & \text{if } (\boldsymbol{x}^{\downarrow \mathtt{pa}_D} = \boldsymbol{\pi}_i \text{ and } \boldsymbol{x}^{\downarrow X_i} \neq \boldsymbol{x}^{\downarrow T_i}) \,. \end{cases}$$

Finally, the functions $p_X^{\mathtt{pa}_X}$ for each child X of D have D substituted by X_m in their domain, without altering the numerical values.

Figure 2 depicts a decision node with many parents (on the left) and the new sub-diagram generated by Transformation 6 (on the right). It is not difficult to see that the treewidth of the transformed diagram is increased by at most three, because the subgraph containing the new nodes, the parents of D and the children of D is triangulated and contains cliques with at most $|\mathtt{pa}_D \cup \{X_i, X_{i-1}, D_i\}|$ variables.[3] Also, the transformation for two different decision variables affect different parts, and hence transforming a diagram in a diagram with parentless decisions does not increase the treewidth by more than three. The following result states that also optimality of strategies is preserved by the transformation.

3. Since the treewidth is given by the size of largest clique in the triangulated moral graph minus one, $|\mathtt{pa}_D|$ is a lower bound on the treewidth of the original graph.

Proposition 7. *Let $\mathcal{L}'$ be the result of applying Transformation 6 on a decision variable D in a LIMID $\mathcal{L}$, s' denote a strategy for $\mathcal{L}'$, and $\delta_{T_1}, \ldots, \delta_{T_m}$ denote the corresponding policies for $T_1, \ldots, T_m$ in s'. Let also δ_D be a policy for D such that $\delta_D(\boldsymbol{\pi}_i) = \delta_{T_i}$ for all $\boldsymbol{\pi}_i \in \Omega_{\mathtt{pa}_D}$. Finally, let s be a strategy for $\mathcal{L}$ obtained by substituting $\delta_{T_1}, \ldots, \delta_{T_m}$ in s' with δ_D (and keeping the remaining policies). Then s is an optimal strategy for $\mathcal{L}$ if and only if s' is an optimal strategy for $\mathcal{L}'$.*

The proof is in the appendix. For each decision variable D in the original LIMID, the transformed model contains m chance variables specifying $m|\Omega_D|^3$ values, and m decision nodes with $|\Omega_D|$ states. If the treewidth of the original diagram is bounded, then m is bounded and the transformation takes polynomial time.[4] In the example of a ten-state decision variable with four ternary parents, the transformation replaces the decision variable D with $3^4 = 81$ decision variables whose space of policies contain 10 elements each, besides the 81 chance variables. The combined space of policies, that is, $\Delta_{T_1} \times \cdots \times \Delta_{T_{81}}$ contains also 10^{81} elements, so that the total search space is still (doubly) exponential in the input. However, algorithms can take advantage of the smaller local policy spaces to reach better solutions, and this is particularly true for the algorithm we devise later on.

In the rest of the paper we assume without loss of generality that decision nodes have no parents and utilities are nonnegative.

3. Solving LIMIDs

In this section, we describe a new algorithm for solving LIMIDs exactly by propagating multiple non-dominated solutions. We start by defining the basic algebraic structure of our algorithm, which is given by the framework of valuation algebra. We show that this framework alone, similar to the one used by SPU, might lead to poor accuracy. We thus extend the framework with sets of valuations that attempt to improve accuracy by increasing complexity. Efficiency is obtained by pruning sets so that their cardinality is kept as small as possible without affecting accuracy.

3.1 Valuation Algebra

The basic ingredients of our algorithmic framework for representing and handling information in LIMIDs are the so called *valuations*, which encode information (probabilities, utilities and policies) about the elements of a domain. Each valuation is associated to a subset of the variables in $\mathcal{U}$, called its *scope*. More concretely, a valuation ϕ with scope x is a pair (p, u) of nonnegative real-valued functions p and u over the domain Ω_x; we refer to p and u as the probability and utility part, respectively, of ϕ. Often, we write ϕ_x to make explicit the scope x of a valuation ϕ. For any $x \subseteq \mathcal{U}$, we denoted the set of all possible valuations with scope x by Φ_x. The set of all possible valuations is thus given by $\Phi \triangleq \bigcup_{x \subseteq \mathcal{U}} \Phi_x$. The set Φ is closed under two basic operations of *combination* and *marginalization*. Combination represents the aggregation of information and is defined as follows.

Definition 8. *If $\phi = (p, u)$ and $\psi = (q, v)$ are valuations with scopes x and y, respectively, its combination $\phi \otimes \psi$ is the valuation $(pq, pv + qu)$ with scope $x \cup y$.*

4. If the treewidth is not bounded then the output of any algorithm, that is, an optimal strategy, might take space exponential in the input.

Marginalization, on the other hand, acts by coarsening information:

Definition 9. *If $\phi = (p, u)$ is a valuation with scope x, and y is a set of variables such that $y \subseteq x$, the marginal $\phi^{\downarrow y}$ is the valuation $(\sum_{x \setminus y} p, \sum_{x \setminus y} u)$ with scope y. In this case, we say that $z \triangleq x \setminus y$ has been* eliminated *from ϕ, which we denote by ϕ^{-z}.*

Notice that our definitions of combination and marginalization slightly differ from previous works on influence diagrams (e.g., Lauritzen & Nilsson, 2001), which usually require a division of the utility part by the probability part. The removal of the division operation turns out to be an important feature when we discuss maximality of valuations later on, but otherwise our definition is equivalent to valuations with division, in the sense that one could easily reformulate message-passing algorithms like SPU using our definition.

In terms of computational complexity, combining two valuations ϕ and ψ with scopes x and y, respectively, requires $3|\Omega_{x \cup y}|$ multiplications and $|\Omega_{x \cup y}|$ additions of numbers; computing $\phi^{\downarrow y}$, where $y \subseteq x$, costs $|\Omega_{x \cup y}|$ operations of addition. In other words, the cost of combining or marginalizing a valuation is exponential in the cardinality of its scope (and linear in the cardinality of its domain). Hence, we wish to work with valuations whose scope is as small as possible. The following result shows that our framework respects the necessary conditions for computing efficiently with valuations (in the sense of keeping the scope of valuations obtained from combinations and marginalizations of other valuations minimal).

Proposition 10. *The system $(\Phi, \mathcal{U}, \otimes, \downarrow)$ satisfies the following three axioms of a (weak)* labeled valuation algebra *(Shenoy & Shafer, 1990; Kohlas, 2003).*

(A1) Combination is commutative and associative, that is, for any $\phi_1, \phi_2, \phi_3 \in \Phi$ we have that

$$\phi_1 \otimes \phi_2 = \phi_2 \otimes \phi_1 \, ,$$
$$\phi_1 \otimes (\phi_2 \otimes \phi_3) = (\phi_1 \otimes \phi_2) \otimes \phi_3 \, .$$

(A2) Marginalization is transitive, that is, for $\phi_z \in \Phi_z$ and $y \subseteq x \subseteq z$ we have that

$$(\phi_z^{\downarrow x})^{\downarrow y} = \phi_z^{\downarrow y} \, .$$

(A3) Marginalization distributes over combination, that is, for $\phi_x \in \Phi_x$, $\phi_y \in \Phi_y$ and $x \subseteq z \subseteq x \cup y$ we have that

$$(\phi_x \otimes \phi_y)^{\downarrow z} = \phi_x \otimes \phi_y^{\downarrow y \cap z} \, .$$

Proof. (A1) follows directly from commutativity, associativity and distributivity of product and sum of real-valued functions, and (A2) follows directly from commutativity of the sum-marginal operation. To show (A3), consider any two valuations (p, u) and (q, v) with scopes x and y, respectively, and a set z such that $x \subseteq z \subseteq x \cup y$. By definition of $\otimes$ and $\downarrow$, we have that

$$[(p, u) \otimes (q, v)]^{\downarrow z} = \left(\sum_{x \cup y \setminus z} pq, \sum_{x \cup y \setminus z} (pv + qu) \right) \, .$$

Since $x \cup y \setminus z = y \setminus z$, and p and u are functions over Ω_x, it follows that

$$\left(\sum_{x \cup y \setminus z} pq, \sum_{x \cup y \setminus z} (pv + qu) \right) = \left(p \sum_{y \setminus z} q, p \sum_{y \setminus z} v + u \sum_{y \setminus z} q \right)$$
$$= (p, u) \otimes \left(\sum_{y \setminus z} q, \sum_{y \setminus z} v \right),$$

which equals $(p, y) \otimes (q, v)^{\downarrow y \cap z}$. □

The following result by Kohlas (2003, Section 2.2) is a direct consequence of (A3) that we shall use to prove the correctness of our algorithm.

Lemma 11. *If $\phi_x \in \Phi_x$, $\phi_y \in \Phi_y$, $z \subseteq y$ and $z \cap x = \emptyset$, then $(\phi_x \otimes \phi_y)^{-z} = \phi_x \otimes \phi_y^{-z}$.*

The primary goal of a valuation algebra is the computation of marginal valuations of the form $\psi = (\phi_1 \otimes \cdots \otimes \phi_m)^{\downarrow \emptyset}$. Let $\{X_1, \ldots, X_n\}$ be the set of variables appearing in the scopes of $\phi_1, \ldots, \phi_m$. The marginal ψ can be computed efficiently by a variable elimination procedure[5] that receives a set $\Psi = \{\phi_1, \ldots, \phi_m\}$ and a permutation σ of the variables $X_1, \ldots, X_n$, and for $i = 1, \ldots, n$ replaces all valuations $\phi_i^{(1)}, \ldots, \phi_i^{(k_i)}$ whose scope contains variable $\sigma(X_i)$ with the marginal $\phi_i = \left(\phi_i^{(1)} \otimes \cdots \otimes \phi_i^{(k_i)}\right)^{-\sigma(X_i)}$. Algorithm 1 describes the procedure. The algorithm returns a valuation $\phi_j \otimes \cdots \otimes \phi_k$, where $\phi_j, \ldots, \phi_k \in \Psi_n$, which equals ψ by Axioms (A1)–(A3) (Kohlas, 2003, Section 4.1).

Algorithm 1 VariableElimination(x,σ,Ψ)

Input: A permutation σ of the variables $x = \{X_1, \ldots, X_n\}$ and a set of valuations $\Psi = \{\phi_1, \ldots, \phi_m\}$ over subsets of x

Output: The marginal valuation $(\phi_1 \otimes \cdots \otimes \phi_m)^{\downarrow \emptyset}$

Let $\Psi_0 \leftarrow \Psi$
for $i \leftarrow 1$ **to** n **do**
  Let $\phi_i^{(1)}, \ldots, \phi_i^{(k)}$ denote the valuations in Ψ_{i-1} whose scope contains $\sigma(X_i)$
  Compute $\phi_i = \left(\phi_i^{(1)} \otimes \cdots \otimes \phi_i^{(k)}\right)^{-\sigma(X_i)}$
  Let $\Psi_i \leftarrow \Psi_{i-1} \cup \{\phi_i\} \setminus \{\phi_i^{(1)}, \ldots, \phi_i^{(k)}\}$
end for
return the combination of all valuations in Ψ_n

The complexity of the variable elimination procedure is given by the size of the largest valuation ϕ_i generated in the loop. This valuation might have a size exponential in the size of valuations $\phi_1, \ldots, \phi_m$ given as input, but, as we discuss later on, there are certain conditions under which the size of ϕ_i is bounded and the procedure takes time polynomial in the input.

5. Variable elimination algorithms are also known in the literature as fusion algorithms (Shenoy & Shafer, 1990) and bucket elimination (Dechter, 1999).

3.2 Computing Expected Utilities

We can use the valuation algebra framework introduced to compute the expected utility of a given strategy using variable elimination. Let $s = (\delta_D)_{D \in \mathcal{D}} \in \Delta$ be a strategy for a LIMID $\mathcal{L}$ whose expected utility we want to compute, and σ be a permutation of the variables in $\mathcal{C} \cup \mathcal{D}$. We assume that the decision nodes in $\mathcal{L}$ have no parents (otherwise we need first to apply Transformation 6), so that the strategy s is simply a configuration in $\Omega_{\mathcal{D}}$. The procedure in Algorithm 2 computes the expected utility induced by the strategy s. The procedure calls variable elimination with a set Ψ that contains a valuation $\phi_C = (p_C^{\mathtt{pa}_C}, 0)$ for each chance variable, a valuation $\phi_V = (1, u_V)$ for each value variable, and a valuation $\phi_D = (I_{\delta_D}, 0)$ for each decision variable. We have the following result.

Algorithm 2 ExpectedUtility($\mathcal{L}, \sigma, s$)

Input: A LIMID $\mathcal{L}$ whose decision nodes have no parents, a permutation σ of the variables in $\mathcal{C} \cup \mathcal{D}$, and a strategy $s = (\delta_D)_{D \in \mathcal{D}} \in \Delta$
Output: The expected utility of s

Let $\Psi \leftarrow \emptyset$
for $C \in \mathcal{C}$ **do**
    Add $\phi_C = (p_C^{\mathtt{pa}_C}, 0)$ to Ψ
end for
for $V \in \mathcal{V}$ **do**
    Add $\phi_V = (1, u_V)$ to Ψ
end for
for $D \in \mathcal{D}$ **do**
    Add $\phi_D = (I_{\delta_D}, 0)$ to Ψ
end for
Let $\phi_s \leftarrow$ VariableElimination($\mathcal{C} \cup \mathcal{D}, \sigma, \Psi$)
return the utility part of ϕ_s

Proposition 12. *The procedure described in Algorithm (2) returns the expected utility of the strategy s.*

Proof. Let ϕ_s be the output of the Variable Elimination Algorithm. According to Axioms (A1)–(A3), we have that $\phi_s = \psi^{\downarrow \emptyset}$, where

$$\psi = \left[\bigotimes_{C \in \mathcal{C}} (p_C^{\mathtt{pa}_C}, 0) \right] \otimes \left[\bigotimes_{D \in \mathcal{D}} (I_{\delta_D}, 0) \right] \otimes \left[\bigotimes_{V \in \mathcal{V}} (1, u_V) \right] .$$

Let p and u denote the probability and utility part, respectively, of ϕ_s. By definition of combination, we have that $\psi = (p_s, p_s \sum_{V \in \mathcal{V}} u_V)$, where $p_s = \prod_{X \in \mathcal{C} \cup \mathcal{D}} p_X^{\mathtt{pa}_X}$ as in (1). Since p_s is a probability distribution over $\mathcal{C} \cup \mathcal{D}$, it follows that $p = \sum_{\boldsymbol{x} \in \Omega_{\mathcal{C} \cup \mathcal{D}}} p_s(\boldsymbol{x}) = 1$. Finally, $u = \sum_{\mathcal{C} \cup \mathcal{D}} p_s \sum_{V \in \mathcal{V}} u_V$, which equals $\mathrm{E}_s[\mathcal{L}]$ by (2). $\square$

Consider the LIMID of the fire dispatching problem (Figure 1) and the strategy $s = (\boldsymbol{a}, \boldsymbol{w}, \boldsymbol{a})$ whose expected utility we want to compute using the procedure above. We assume that the utilities are nonnegative (i.e., we have already applied Transformation 3). According to the procedure in Algorithm 2, we first generate the set $\Psi = \{\phi_O = (p_O^{T_1,T_2,T_3}, 0), \phi_{V_1} = (1, u'_{V_1}), \phi_{V_2} = (1, u'_{V_2}), \phi_{V_3} = (1, u'_{V_3}), \phi_V = (1, u'_v), \phi_{T_1} = (I_{\boldsymbol{a}}, 0), \phi_{T_2} = (I_{\boldsymbol{w}}, 0), \phi_{T_2} = (I_{\boldsymbol{a}}, 0)\}$. For $X_1 = O$, $X_2 = T_1$, $X_3 = T_2$, $X_4 = T_3$, let σ be a permutation of the variables such that $\sigma(X_i) = X_i$ for $i = 1, \ldots, 4$. The variable elimination algorithm with Ψ and σ as input produces the valuations

$$\phi_1 = (\phi_V \otimes \phi_O)^{-O} \qquad \phi_2 = (\phi_{V_1} \otimes \phi_{T_1} \otimes \phi_1)^{-T_1}$$
$$\phi_3 = (\phi_{V_2} \otimes \phi_{T_2} \otimes \phi_2)^{-T_2} \qquad \phi_4 = (\phi_{V_3} \otimes \phi_{T_3} \otimes \phi_3)^{-T_3}$$

during its loop, and outputs the valuation $\phi_s = \phi_4 = (1, 2)$. Similarly, to compute the expected utility of the optimal strategy $s^* = (\boldsymbol{a}, \boldsymbol{a}, \boldsymbol{a})$ we run variable elimination with $\Psi = \{\phi_O = (p_O^{T_1,T_2,T_3}, 0), \phi_{V_1} = (1, u'_{V_1}), \phi_{V_2} = (1, u'_{V_2}), \phi_{V_3} = (1, u'_{V_3}), \phi_V = (1, u'_v), \phi_{T_1} = (I_{\boldsymbol{a}}, 0), \phi_{T_2} = (I_{\boldsymbol{a}}, 0), \phi_{T_2} = (I_{\boldsymbol{a}}, 0)\}$, which then outputs $\phi_{s^*} = (1, 9/2)$.

In general, the described procedure can take time exponential in the input. However, when $\mathcal{L}$ has bounded treewidth it can be shown that there exists a permutation σ for which the procedure takes time polynomial in the input. (e.g., Koller & Friedman, 2009, Section 23.4.3). Hence, if the space of strategies is sufficiently small, we can find an optimal strategy by simply ranking strategies according to their expected utilities. However, we do not expect this to be feasible for any realistic diagram as the space of strategies increases exponentially with the number of decision nodes (assuming they have no parents), even in diagrams of bounded treewidth and bounded number of states per variable.

3.3 Local Search Algorithms

As a first attempt to design a fast algorithm to solve LIMIDs, one might suggest a local search scheme that starts with a random solution and repeatedly explores its neighborhood in order to find a solution with higher expected utility. If the treewidth of the diagram is bounded, the expected utility of each neighbor solution can be efficiently computed, so the complexity of the algorithm is given by the size of the neighborhood. A possible approach is then to define the neighborhood of a solution to be the strategies obtained by changing a single policy, which gives a local search space polynomial in the input. Algorithm 3 describes a greedy procedure that at each step looks for a new policy that improves on the current best solution. The algorithm is guaranteed to find a strategy which is locally optimal in its neighborhood, that is, it cannot be improved by changing only one of its policies. Lauritzen and Nilsson (2001) stated sufficient conditions that a diagram has to satisfy in order to guarantee that the solution produced by a local search procedure is (globally) optimal. Unfortunately, as the following example shows, these conditions are violated by even structurally very simple chain diagrams, and in such cases a local search procedure might output local optima of very poor accuracy.

Consider the LIMID of our running example, and suppose we start with strategy $s_0 = (\boldsymbol{a}, \boldsymbol{w}, \boldsymbol{a})$, which has expected utility 2. At the first step we might try to improve the policy for T_1, producing strategy $s = (\boldsymbol{w}, \boldsymbol{w}, \boldsymbol{a})$ whose expected utility is 3. Since this is higher than the expected utility of the initial solution, we set $s_{\text{best}} \leftarrow s$ and update the highest expected utility found. Next, we try to search for a better policy for T_2, and we generate strategy $s = (\boldsymbol{w}, \boldsymbol{a}, \boldsymbol{a})$. This strategy has an expected utility of 2, which is less than the

Algorithm 3 GreedyPolicySearch($\mathcal{L}, \sigma, s_0$)

Input: A LIMID $\mathcal{L}$, a permutation σ of the variables in $\mathcal{C} \cup \mathcal{D}$, and an initial strategy $s_0 = (\delta_D)_{D \in \mathcal{D}}$

Output: A locally optimum strategy s_{best}

```
Let s_best ← s_0 and E_{s_best}[L] ← ExpectedUtility(L, σ, s_0)
repeat
   Generate a new candidate strategy s by replacing a single policy δ_D in s_best
   Compute E_s[L] ← ExpectedUtility(L, σ, s)
   if E_s[L] > E_{s_best}[L] then
      Set s_best ← s and E_{s_best}[L] ← E_s[L]
   end if
until current solution cannot be further improved in this way
return s_best
```

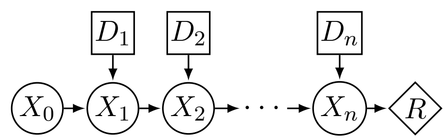


Figure 3: A chain structure diagram with n decision variables.

expected utility of the best solution found so far. Finally, we look for a better policy for T_3, which leads us to strategy $s = (\boldsymbol{w}, \boldsymbol{w}, \boldsymbol{w})$, whose expected utility is 4. Since this is better than our current best solution we set $s_{\text{best}} \leftarrow s$ and update the associated expected utility. Since no change of a single policy can improve this strategy, the algorithm halts with a solution whose expected utility is 4 against a maximum expected utility of 9/2 achieved by strategy $s^* = (\boldsymbol{a}, \boldsymbol{a}, \boldsymbol{a})$ (more than 10% relative error), or, in terms of the original diagram (by means of Proposition 4), an expected utility of zero against a maximum expected utility of 1/2.

The procedure we have just described is very similar to the SPU algorithm, and it illustrates the pitfalls of a local search. In fact, SPU will output just the same local optimum (but it would start with a uniform policy for every decision variable). Note that the solution obtained by the greedy local search on the example degrades as the ratio of the utility of success (achieved only by strategy $(\boldsymbol{a}, \boldsymbol{a}, \boldsymbol{a})$) and the utility of failure increases. For instance, if the utility of success were increased to $u'_V(\boldsymbol{s}) = 10$ and the utility of failure remained the same, that is, if $u'_V(\boldsymbol{f}) = 0$, the algorithm would reach a solution whose expected utility is 4, an error of 60% relative to the maximum expected utility of 10. Moreover, cases where SPU performs poorly are not rare. For instance, the plots in Figure 4 show SPU's relative performance in chain diagrams like the one in Figure 3. Each diagram was generated by independently sampling each conditional distribution associated to a chance node from a symmetric Dirichlet distribution with parameter $1/m$, where m is the number of variable states. The maximum expected utility of each diagram was computed using the algorithm we devise here, which took less than 3 seconds on any diagram in the experiment.

Each (blue) point in the plots in Figure 4 depict the relative error of SPU on a given diagram. The (red) line indicates the third quartile of each fixed configuration. The di-

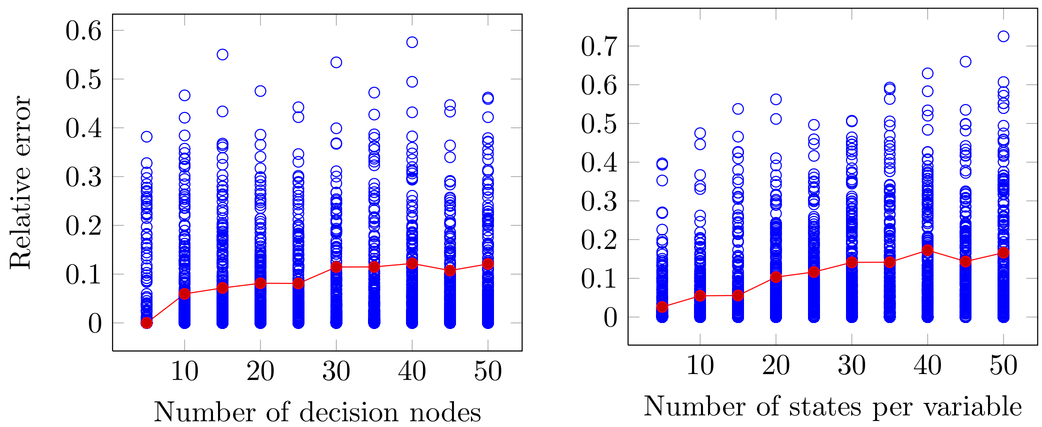


Figure 4: Relative performance of SPU on randomly generated chain diagrams. Each (blue) circle depicts an experiment and the (red) line depicts the third quartile.

agrams in the left hand-side plot were obtained with the number of states fixed at 15, while the diagrams on the right had the number of decision variables fixed in ten. For example, we see from the third quartile line on the right-hand side plot that in 25% of the chain diagrams of 20 states and ten decision variables, SPU returned a strategy s such that $(\text{MEU}[\mathcal{L}] - \text{E}_s[\mathcal{L}])/\text{MEU}[\mathcal{L}] \geq 0.1$. Also, there were cases where SPU obtains up to 70% relative error. On the other hand, we see that in the majority of the cases the solution returned by SPU achieved a relative error of less than 10%. All in all, these experiments show that local search is effective in many cases, but may produce very poor results.

3.4 Ordered Valuations

Can we also exploit the redundancy in the computation of expected utility of neighboring strategies to decide whether a candidate solution improves the current solution without having to run variable elimination completely? For instance, when evaluating the quality of a new candidate strategy that differs from the current best strategy only by the policy associated to T_1, can we have any insight by inspecting two valuations ϕ_2 produced by variable elimination in the example in Section 3.2 using two different strategies? Fortunately, the answer is yes, and to show this we need the concept of ordered valuations.

Let us define a partial order (i.e., a reflexive, antisymmetric and transitive relation) over Φ, the set of all possible valuations, as follows.

Definition 13. *For any two valuations $\phi = (p, u)$ and $\psi = (q, v)$ in Φ, we say that ψ* dominates *ϕ (conversely, we say that ϕ* is dominated by *ψ), and we write $\phi \leq \psi$, if ϕ and ψ have equal scope, $p \leq q$, and $u \leq v$.*

If ϕ and ψ have scope x, deciding whether ψ dominates ϕ costs at most $2|\Omega_x|$ operations of comparison of numbers. The following result shows that the algebra of valuations is monotonic with respect to dominance.

Proposition 14. *The system $(\Phi, \mathcal{U}, \otimes, \downarrow, \leq)$ satisfies the following two additional axioms of an* ordered valuation algebra *(Haenni, 2004).*

(A4) Combination is monotonic with respect to dominance, that is,

$$\text{if } \phi_x \leq \psi_x \text{ and } \phi_y \leq \psi_y \text{ then } (\phi_x \otimes \phi_y) \leq (\psi_x \otimes \psi_y)\,.$$

(A5) Marginalization is monotonic with respect to dominance, that is,

$$\text{if } \phi_x \leq \psi_x \text{ then } \phi_x^{\downarrow y} \leq \psi_x^{\downarrow y}\,.$$

Proof. (A4). Consider two valuations (p_x, u_x) and (q_x, v_x) with scope x such that $(p_x, u_x) \leq (q_x, v_x)$, and two valuations (p_y, u_y) and (q_y, v_y) with scope y satisfying $(p_y, u_y) \leq (q_y, v_y)$. By definition of $\leq$, we have that $p_x \leq q_x$, $u_x \leq v_x$, $p_y \leq q_y$ and $u_y \leq v_y$. Since all functions are nonnegative, it follows that $p_x p_y \leq q_x q_y$, $p_x u_y \leq q_x v_y$ and $p_y u_x \leq q_y v_x$. Hence, $(p_x, u_x) \otimes (p_y, u_y) = (p_x p_y, p_x u_y + p_y u_x) \leq (q_x q_y, q_x v_y + q_y v_x) = (q_x, v_x) \otimes (q_y, v_y)$. (A5). Let y be a subset of x. It follows from monotonicity of $\leq$ with respect to addition of real numbers that

$$(p_x, u_x)^{\downarrow y} = \left(\sum_{x \setminus y} p_x, \sum_{x \setminus y} u_x \right) \leq \left(\sum_{x \setminus y} q_x, \sum_{x \setminus y} v_x \right) = (q_x, v_x)^{\downarrow y}\,.$$

Hence, the result follows. □

Axioms (A4) and (A5) assert that combination and marginalization preserve the partial ordering between valuations. This allow us to detect suboptimal strategies early in the variable elimination procedure. Consider comparing the strategies $s = (\boldsymbol{w}, \boldsymbol{w}, \boldsymbol{w})$ and $s' = (\boldsymbol{w}, \boldsymbol{a}, \boldsymbol{w})$ for the LIMID of our running example. At the third iteration of the loop (i.e., when $i = 3$), the variable elimination procedure produces valuations $\phi_3^s = (p_3^s, u_3^s)$ and $\phi_3^{s'} = (p_3^{s'}, u_3^{s'})$ for the strategies s and s', respectively, such that

$$\begin{aligned} p_3^s(\boldsymbol{a}) &= 1\,, & p_3^s(\boldsymbol{w}) &= 1\,, \\ u_3^s(\boldsymbol{a}) &= 3\,, & u_3^s(\boldsymbol{w}) &= 3\,, \\ p_3^{s'}(\boldsymbol{a}) &= 1\,, & p_3^{s'}(\boldsymbol{w}) &= 1\,, \\ u_3^{s'}(\boldsymbol{a}) &= 2\,, & u_3^{s'}(\boldsymbol{w}) &= 2\,. \end{aligned}$$

Thus, $\phi_3^{s'} \leq \phi_3^s$. Since $\phi_4^{s'} = (\phi_{T_3}^s \otimes \phi_{V_3} \otimes \phi_3^{s'})^{-T_3}$ and $\phi_4^s = (\phi_{T_3}^s \otimes \phi_{V_3} \otimes \phi_3^s)^{-T_3}$, we know by Axioms (A4) and (A5) that $\phi_4^{s'} \leq \phi_4^s$, and hence that $E_{s'}[\mathcal{L}] \leq E_s[\mathcal{L}]$. Therefore, there is no need to continue the execution of variable elimination for s', as its expected value cannot be higher than that of s.

Unfortunately, suboptimal solutions do not always produce valuations that are dominated by an optimal one during variable elimination. As an example, consider strategies $s = (\boldsymbol{a}, \boldsymbol{w}, \boldsymbol{a})$ and $s^* = (\boldsymbol{a}, \boldsymbol{a}, \boldsymbol{a})$. At the third step, variable elimination generates valuations $\phi_3^s = (p_3^s, u_3^s)$ and $\phi_3^{s^*} = (p_3^{s^*}, u_3^{s^*})$ such that

$$\begin{aligned} p_3^s(\boldsymbol{a}) &= 1\,, & p_3^s(\boldsymbol{w}) &= 1\,, \\ u_3^s(\boldsymbol{a}) &= 2\,, & u_3^s(\boldsymbol{w}) &= 2\,, \\ p_3^{s^*}(\boldsymbol{a}) &= 1\,, & p_3^{s^*}(\boldsymbol{w}) &= 1\,, \\ u_3^{s^*}(\boldsymbol{a}) &= 9/2\,, & u_3^{s^*}(\boldsymbol{w}) &= 1\,. \end{aligned}$$

Thus, even though s^* is an optimal strategy, $\phi_3^s \not\leq \phi_3^{s^*}$.

The algorithm we devise later on exploits that fact that some suboptimal solutions can be early detected and eliminated from the search space. Because some suboptimal solutions might not be eliminated during variable elimination, the algorithm runs in exponential time in the worst case, but this is just to be expected, as the problem is NP-hard. Fortunately, our experiments with random problems suggest that situations like this are not frequent.

3.5 Sets of Valuations

The multiple runs of variable elimination for different inputs but same elimination ordering (i.e., a permutation of the variables) are represented by sets in the framework of the algorithm we devise later on. For instance, we might consider the set Ψ_3 of all valuations ϕ_3 produced by variable elimination at the third iteration of the loop for every possible strategy. Due to the monotonicity of combination and marginalization with respect to $\leq$, we can immediately halt the computation of valuations from Ψ_3 that are dominated by some other, that is, we can remove dominated valuations from Ψ_3. This is formalized by the concept of maximal valuations and the operator max:

Definition 15. *Given a finite set of valuations $\Psi \subseteq \Phi$, we say that $\phi \in \Psi$ is* maximal *if for all $\psi \in \Psi$ such that $\phi \leq \psi$ it holds that $\psi \leq \phi$. The operator* max *returns the set* $\max(\Psi)$ *of maximal valuations of Ψ.*

If Ψ_x is a set with m valuations of scope x, the set of maximal valuations $\max(\Psi_x)$ can be obtained by m^2 comparisons $\phi \leq \psi$, where $(\phi, \psi) \in \Psi_x \times \Psi_x$.

When all valuations in the set Ψ_x have the same scope x, we say that Ψ_x also have scope x. We can extend combination and marginalization to sets of valuations as follows.

Definition 16. *If Ψ_x and Ψ_y are any two sets of valuations in Φ,*

$$\Psi_x \otimes \Psi_y \triangleq \{\phi_x \otimes \phi_y : \phi_x \in \Psi_x, \phi_y \in \Psi_y\}$$

denotes the set obtained from all combinations of a valuation in Ψ_x and a valuation in Ψ_y.

Definition 17. *If $\Psi_x \subseteq \Phi_x$ is a set of valuations with scope x and $y \subseteq x$,*

$$\Psi_x^{\downarrow y} \triangleq \{\phi_x^{\downarrow y} : \phi_x \in \Psi_x\}$$

denote the set of valuations obtained by element-wise marginalization of valuations to y.

It can be checked that sets of valuations with combination and marginalization defined element-wise satisfy axioms (A1)–(A3), and therefore form a valuation algebra. Hence, Lemma 11 applies also for sets of valuations with marginalization and combination defined as above.

Lemma 18. *If $\Psi_x \subseteq \Phi_x$ and $\Psi_y \subseteq \Phi_y$ are two sets of valuations with scope x and y, respectively, and z is a set of variables such that $z \subseteq y$ and $z \cap x = \emptyset$, then $(\Psi_x \otimes \Psi_y)^{-z} = \Psi_x \otimes \Psi_y^{-z}$.*

Proof. The result follows from element-wise application of Lemma 11 to $(\phi_x \otimes \phi_y)^{-z} \in (\Psi_x \otimes \Psi_y)^{-z}$. $\square$

3.6 Solving LIMIDs Exactly

We are now ready to describe the MultiplePolicyUpdating (MPU) algorithm, which solves arbitrary LIMIDs exactly. The algorithm assumes that the decision nodes have no variables, and that utilities are nonnegative, hence Transformations 6 and 3 have to be applied before running the algorithm in case some of these assumptions fail.

Consider a LIMID $\mathcal{L}$ and a permutation σ of the variables in $\mathcal{C} \cup \mathcal{D}$, and let $n = |\mathcal{C} \cup \mathcal{D}|$. The algorithm is initialized by generating a set $\mathcal{S}_0$ that contains a singleton $\{(p_C^{\mathtt{pa}_C}, 0)\}$ for each chance variable C, a singleton $\{(1, u_V)\}$ for each value variable V and a set of valuations $\{(p_D, 0)\}$, which contains one element $(p_D, 0)$ per policy δ_D, for each decision variable D. Then a set-valued variable elimination is performed for the sets of valuations in S_0, with dominated valuations being discarded after each set marginalization. Finally, an optimal solution is obtained from the utility part of the single maximal valuation in the set combination of all sets of valuations in $\mathcal{S}_n$ obtained after the variable elimination. The procedure is detailed in Algorithm 4.

Algorithm 4 MultiplePolicyUpdating($\mathcal{L}, \sigma$)

Input: A LIMID $\mathcal{L}$ and a permutation σ of the variables in $\mathcal{C} \cup \mathcal{D}$
Output: The maximum expected utility

1: Let $\mathcal{S}_0 \leftarrow \emptyset$
2: **for** $C \in \mathcal{C}$ **do**
3:     Add a singleton $\{(p_C^{\mathtt{pa}_C}, 0)\}$ to $\mathcal{S}_0$
4: **end for**
5: **for** $V \in \mathcal{V}$ **do**
6:     Add a singleton $\{(1, u_V)\}$ to $\mathcal{S}_0$
7: **end for**
8: **for** $D \in \mathcal{D}$ **do**
9:     Add a set $\{(I_{\boldsymbol{d}}, 0) : \boldsymbol{d} \in \Omega_D\}$ to $\mathcal{S}_0$
10: **end for**
11: **for** $i \leftarrow 1$ **to** n **do**
12:     Let $\Psi_i^{(1)}, \ldots, \Psi_i^{(k)}$ denote the sets in $\mathcal{S}_{i-1}$ whose scope contains $\sigma(X_i)$
13:     Compute $\Psi_i = \max\left(\left[\Psi_i^{(1)} \otimes \cdots \otimes \Psi_i^{(k)}\right]^{-\sigma(X_i)}\right)$
14:     Let $\mathcal{S}_i \leftarrow \mathcal{S}_{i-1} \cup \{\Psi_i\} \setminus \{\Psi_i^{(1)}, \ldots, \Psi_i^{(k)}\}$
15: **end for**
16: Let $\mathcal{S}$ denote the set combination of all sets in $\mathcal{S}_n$
17: **return** the utility part u of $(p, u) \in \max(\mathcal{S})$

Since all variables have been eliminated at the end of the loop, the valuations in sets in $\mathcal{S}$ have empty scope and have both their probability and utility parts identified with numbers. Hence, the algorithm outputs an expected utility (i.e., a real number) at line 17.

Let us illustrate the algorithm with an example. Once more, consider the LIMID $\mathcal{L}$ of the fire dispatching problem after applying Transformation 3 and assume the same elimination ordering of the variables used in the example in Section 3.2. We start with an empty set

$\mathcal{S}_0$. O is the only chance variable, and we add a set

$$\Psi_O = \{(p_O^{T_1,T_2,T_3}, 0)\}$$

to $\mathcal{S}_0$. Then we add the sets

$$\Psi_{V_1} = \{(1, u'_{V_1})\}, \qquad \Psi_{V_2} = \{(1, u'_{V_2})\},$$
$$\Psi_{V_3} = \{(1, u'_{V_3})\}, \qquad \Psi_{V} = \{(1, u'_{V})\}$$

to $\mathcal{S}_0$ due to the value variables V_1, V_2, V_3 and V, respectively. The decision variable T_1 causes a set

$$\Psi_{T_1} = \{(I_{\boldsymbol{a}}, 0), (I_{\boldsymbol{w}}, 0)\}$$

with scope T_1 to be included in $\mathcal{S}_0$, and similarly, for variables T_2 and T_3, that is, we add the sets

$$\Psi_{T_2} = \{(I_{\boldsymbol{a}}, 0), (I_{\boldsymbol{w}}, 0)\}, \qquad \Psi_{T_3} = \{(I_{\boldsymbol{a}}, 0), (I_{\boldsymbol{w}}, 0)\},$$

with scopes T_2 and T_3, respectively, to $\mathcal{S}_0$, obtaining $\mathcal{S}_0 = \{\Psi_O, \Psi_{V_1}, \Psi_{V_2}, \Psi_{V_3}, \Psi_V, \Psi_{T_1}, \Psi_{T_2}, \Psi_{T_3}\}$. In the first iteration ($i = 1$) of the variable elimination loop (lines 11–15), we have that

$$\Psi_1 = \max\left([\Psi_V \otimes \Psi_O]^{-O}\right) = \{(p_1, u_1)\},$$

where p_1 and u_1 are functions over Ω_{T_1,T_2,T_3} such that $p_1 = 1$ and $u_1(\boldsymbol{a},\boldsymbol{a},\boldsymbol{a}) = 9/2$, and $u_1(\boldsymbol{x}) = 1$ for all $x \neq (\boldsymbol{a},\boldsymbol{a},\boldsymbol{a})$. Note that Ψ_1 is a singleton since only singletons were involved is its computation. In the second iteration we have that

$$\Psi_2 = \max\left([\Psi_{V_1} \otimes \Psi_{T_1} \otimes \Psi_1]^{-T_1}\right) = \{(p_2^{\boldsymbol{a}}, u_2^{\boldsymbol{a}}), (p_2^{\boldsymbol{w}}, u_2^{\boldsymbol{w}})\},$$

where $p_2^{\boldsymbol{a}}, u_2^{\boldsymbol{a}}, p_2^{\boldsymbol{w}}, u_2^{\boldsymbol{w}}$ are functions over Ω_{T_2,T_3} such that $p_2^{\boldsymbol{a}} = p_2^{\boldsymbol{w}} = 1$, $u_2^{\boldsymbol{a}}(\boldsymbol{a},\boldsymbol{a}) = 9/2$, $u_2^{\boldsymbol{a}}(\boldsymbol{x}) = 1$ for all $\boldsymbol{x} \neq (\boldsymbol{a},\boldsymbol{a})$, and $u_2^{\boldsymbol{w}} = 2$. Note that we have labeled the functions according to the policies δ_{T_1} that generated them. This allows us to easily extract the optimal strategy at the end of the algorithm.

In the third iteration we need to compute

$$\begin{aligned}
\Psi_3 &= \max\left([\Psi_{V_2} \otimes \Psi_{T_2} \otimes \Psi_2]^{-T_2}\right) \\
&= \max(\{(p_3^{(\boldsymbol{a},\boldsymbol{a})}, u_3^{(\boldsymbol{a},\boldsymbol{a})}), (p_3^{(\boldsymbol{a},\boldsymbol{w})}, u_3^{(\boldsymbol{a},\boldsymbol{w})}), (p_3^{(\boldsymbol{w},\boldsymbol{w})}, u_3^{(\boldsymbol{w},\boldsymbol{w})})\}) \\
&= \{(p_3^{(\boldsymbol{a},\boldsymbol{a})}, u_3^{(\boldsymbol{a},\boldsymbol{a})}), (p_3^{(\boldsymbol{w},\boldsymbol{w})}, u_3^{(\boldsymbol{w},\boldsymbol{w})})\},
\end{aligned}$$

where $p_3^{(\boldsymbol{a},\boldsymbol{a})}, u_3^{(\boldsymbol{a},\boldsymbol{a})}, p_3^{(\boldsymbol{a},\boldsymbol{w})}, u_3^{(\boldsymbol{a},\boldsymbol{w})}, p_3^{(\boldsymbol{w},\boldsymbol{w})}, u_3^{(\boldsymbol{w},\boldsymbol{w})}$ are functions over T_3 such that $p_3^{(\boldsymbol{a},\boldsymbol{a})} = p_3^{(\boldsymbol{a},\boldsymbol{w})} = p_3^{(\boldsymbol{w},\boldsymbol{w})} = 1$, $u_3^{(\boldsymbol{a},\boldsymbol{a})}(\boldsymbol{a}) = 9/2$, $u_3^{(\boldsymbol{a},\boldsymbol{a})}(\boldsymbol{w}) = 1$, $u_3^{(\boldsymbol{a},\boldsymbol{w})} = 2$, and $u_3^{(\boldsymbol{w},\boldsymbol{w})} = 3$. Note that the valuation associated to the policies $\delta_{T_2} = \boldsymbol{w}$ and $\delta_{T_2} = \boldsymbol{a}$ do not appear in Ψ_3 because they generate a valuation equal to $(p_3^{(\boldsymbol{a},\boldsymbol{w})}, u_3^{(\boldsymbol{a},\boldsymbol{w})})$. This implies that strategies $(\boldsymbol{a},\boldsymbol{w},\boldsymbol{w})$ and $(\boldsymbol{w},\boldsymbol{a},\boldsymbol{w})$ have the same expected utility, and also strategies $(\boldsymbol{a},\boldsymbol{w},\boldsymbol{a})$ and $(\boldsymbol{w},\boldsymbol{a},\boldsymbol{a})$.

In the last iteration, we generate the set

$$\begin{aligned}
\Psi_4 &= \max\left([\Psi_{V_3} \otimes \Psi_{T_3} \otimes \Psi_3]^{-T_3}\right) \\
&= \max(\{(p_4^{(\boldsymbol{a},\boldsymbol{a},\boldsymbol{a})}, u_4^{(\boldsymbol{a},\boldsymbol{a},\boldsymbol{a})}), (p_4^{(\boldsymbol{w},\boldsymbol{w},\boldsymbol{a})}, u_4^{(\boldsymbol{w},\boldsymbol{w},\boldsymbol{a})}), (p_4^{(\boldsymbol{a},\boldsymbol{a},\boldsymbol{w})}, u_4^{(\boldsymbol{a},\boldsymbol{a},\boldsymbol{w})}), (p_4^{(\boldsymbol{w},\boldsymbol{w},\boldsymbol{w})}, u_4^{(\boldsymbol{w},\boldsymbol{w},\boldsymbol{w})})\}) \\
&= \{(p_4^{(\boldsymbol{a},\boldsymbol{a},\boldsymbol{a})}, u_4^{(\boldsymbol{a},\boldsymbol{a},\boldsymbol{a})})\}\,,
\end{aligned}$$

where $p_4^{(\boldsymbol{a},\boldsymbol{a},\boldsymbol{a})}, u_4^{(\boldsymbol{a},\boldsymbol{a},\boldsymbol{a})}, p_4^{(\boldsymbol{w},\boldsymbol{w},\boldsymbol{a})}, u_4^{(\boldsymbol{w},\boldsymbol{w},\boldsymbol{a})}, p_4^{(\boldsymbol{a},\boldsymbol{a},\boldsymbol{w})}, u_4^{(\boldsymbol{a},\boldsymbol{a},\boldsymbol{w})}, p_4^{(\boldsymbol{w},\boldsymbol{w},\boldsymbol{w})}, u_4^{(\boldsymbol{w},\boldsymbol{w},\boldsymbol{w})}$ are functions over the empty set such that $p_4^{(\boldsymbol{a},\boldsymbol{a},\boldsymbol{a})} = p_4^{(\boldsymbol{w},\boldsymbol{w},\boldsymbol{a})} = p_4^{(\boldsymbol{a},\boldsymbol{a},\boldsymbol{w})} = p_4^{(\boldsymbol{w},\boldsymbol{w},\boldsymbol{w})} = 1$, $u_4^{(\boldsymbol{a},\boldsymbol{a},\boldsymbol{a})} = 9/2$, $u_4^{(\boldsymbol{w},\boldsymbol{w},\boldsymbol{a})} = 3$, $u_4^{(\boldsymbol{a},\boldsymbol{a},\boldsymbol{w})} = 2$, and $u_4^{(\boldsymbol{w},\boldsymbol{w},\boldsymbol{w})} = 4$.

Finally, we have that $\mathcal{S}_4 = \{\Psi_4\}$, so the algorithm returns $u_4^{(\boldsymbol{a},\boldsymbol{a},\boldsymbol{a})} = 9/2$, which is the expected utility of the optimal strategy $(\boldsymbol{a},\boldsymbol{a},\boldsymbol{a})$. As one can see, the optimal strategy is easily recovered by labeling valuations with their corresponding policies.

Differently from other message-passing algorithms that obtain approximate solutions to LIMIDs by (repeatedly) propagating a single valuation (e.g., the SPU algorithm), the MPU algorithm computes exact solutions by propagating several maximal valuations that correspond to partial combinations of local decision rules. The efficiency of the algorithm in handling the propagation of many valuations derives from the early removal of valuations performed by the max operation in the propagation step.

Consider the set $\Psi_{\mathcal{L}} \triangleq \{\phi_s : s \in \Delta\}$, where each ϕ_s is given by

$$\phi_s = \left(\left[\bigotimes_{C\in\mathcal{C}} (p_C^{\mathsf{pa}_C}, 0)\right] \otimes \left[\bigotimes_{D\in\mathcal{D}} (I_{\boldsymbol{d}}, 0)\right] \otimes \left[\bigotimes_{V\in\mathcal{V}} (1, u_V)\right]\right)^{\downarrow\emptyset}$$

such that the functions $I_{\boldsymbol{d}}$ are consistent with the policies in s. It is not difficult to see that

$$\begin{aligned}
\Psi_{\mathcal{L}} &= \left(\left[\bigotimes_{C\in\mathcal{C}} \left\{\left(p_C^{\mathsf{pa}_C}, 0\right)\right\}\right] \otimes \left[\bigotimes_{D\in\mathcal{D}} \{(I_{\boldsymbol{d}}, 0) : \boldsymbol{d} \in \Omega_D\}\right] \otimes \left[\bigotimes_{V\in\mathcal{V}} \{(1, u_V)\}\right]\right)^{\downarrow\emptyset} \\
&= \left(\bigotimes_{\Psi_X\in\mathcal{S}_0} \Psi_X\right)^{\downarrow\emptyset}\,.
\end{aligned}$$

Hence, by Proposition 12 we have that each ϕ_s in $\Psi_{\mathcal{L}}$ is a valuation with probability part one and utility part equal to the expected utility of some strategy s in Δ. Since the relation $\leq$ induces a strict (linear) order over $\Psi_{\mathcal{L}}$, the MEU of the diagram equals the utility part of the (single) valuation in $\max(\Psi_{\mathcal{L}})$. The variable elimination procedure in the propagation step is responsible for obtaining $\max(\bigotimes_{\Psi\in\mathcal{S}_n}\Psi) = \max(\Psi_{\mathcal{L}})$ more efficiently by distributing max and $\downarrow$ over $\bigotimes_{\Psi_X\in\mathcal{S}_0}\Psi_X$, which allows for a significant reduction in the cardinalities of sets and scopes of valuations produced.

We now formally prove the correctness of the algorithm. We start by showing that max distributes over marginalization and combination:

Lemma 19. *(Distributivity of maximality). If $\Psi_x \subset \Phi_x$ and $\Psi_y \subset \Phi_y$ are two finite sets of ordered valuations and $z \subseteq x$, the following holds.*

(i) $\max(\Psi_x \otimes \max(\Psi_y)) = \max(\Psi_x \otimes \Psi_y)$*;*

(ii) $\max(\max(\Psi_x)^{\downarrow z}) = \max(\Psi_x^{\downarrow z})$.

Proof. Part (i) has been shown by Fargier, Rollon, and Wilson (2010, Lemma 1(iv)). We use a similar proof to show that part (ii) also holds. First, we show that $\max(\Psi_x^{\downarrow z}) \subseteq \max(\max(\Psi_x)^{\downarrow z})$. Assume, to show a contradiction, that there is an element $\phi_x^{\downarrow z} \in \max(\Psi_x^{\downarrow z})$, where $\phi_x \in \Psi_x$, which is not an element of $\max(\max(\Psi_x)^{\downarrow z})$. By definition of $\max(\Psi_x)$, there is $\psi_x \in \max(\Psi_x)$ such that $\phi_x \leq \psi_x$. Hence, (A5) implies $\phi_x^{\downarrow z} \leq \psi_x^{\downarrow z}$, and because $\psi_x^{\downarrow z} \in \Psi_x^{\downarrow z}$ it follows that $\phi_x^{\downarrow z} = \psi_x^{\downarrow z}$, and therefore $\phi_x^{\downarrow z} \in \max(\Psi_x)^{\downarrow z}$. Since $\phi_x^{\downarrow z} \notin \max(\max(\Psi_x)^{\downarrow z})$ there is $\phi_z \in \max(\max(\Psi_x)^{\downarrow z})$ such that $\phi_x^{\downarrow z} \leq \phi_z$. But this contradicts our initial assumption since $\phi_z \in \Psi_x^{\downarrow z}$.

Let us now show that $\max(\Psi_x^{\downarrow z}) \supseteq \max(\max(\Psi_x)^{\downarrow z})$. Assume by contradiction that there is $\psi_z \in \max(\max(\Psi_x)^{\downarrow z}) \setminus \max(\Psi_x^{\downarrow z})$. Since $\psi_z \in \Psi_x^{\downarrow z}$, there is $\phi_z \in \max(\Psi_x^{\downarrow z})$ such that $\psi_z \leq \phi_z$. But we have shown that $\max(\Psi_x^{\downarrow z}) \subseteq \max(\max(\Psi_x)^{\downarrow z})$, hence $\psi_z = \phi_z$ and $\psi_z \in \max(\Psi_x^{\downarrow z})$, a contradiction. $\square$

At any iteration i of the propagation step, the combination of all sets in the current pool of sets $\mathcal{S}_i$ produces the set of maximal valuations of the initial factorization marginalized to $X_{i+1}, \ldots, X_n$:

Lemma 20. *For* $i \in \{0, 1, \ldots, n\}$*, it follows that*

$$\max\left(\left[\bigotimes_{\Psi \in \mathcal{S}_0} \Psi\right]^{-\{X_1,\ldots,X_i\}}\right) = \max\left(\bigotimes_{\Psi \in \mathcal{S}_i} \Psi\right),$$

where for each i*,* $\mathcal{S}_i$ *is the collection of sets of valuations generated at the* i*-th iteration of the propagation step of MPU.*

Proof. By induction on i. The basis ($i = 0$) follows trivially.

Assume the result holds at i, that is,

$$\max\left(\left[\bigotimes_{\Psi \in \mathcal{S}_0} \Psi\right]^{-\{X_1,\ldots,X_i\}}\right) = \max\left(\bigotimes_{\Psi \in \mathcal{S}_i} \Psi\right).$$

By eliminating X_{i+1} from both sides and then applying the max operation we get to

$$\max\left(\left[\max\left(\left[\bigotimes_{\Psi \in \mathcal{S}_0} \Psi\right]^{-\{X_1,\ldots,X_i\}}\right)\right]^{-X_{i+1}}\right) = \max\left(\left[\max\left(\bigotimes_{\Psi \in \mathcal{S}_i} \Psi\right)\right]^{-X_{i+1}}\right).$$

Applying Lemma 19(ii) to both sides and (A2) to the left-hand side yields

$$
\begin{aligned}
\max\left(\left[\bigotimes_{\Psi\in\mathcal{S}_0}\Psi\right]^{-\{X_1,\dots,X_{i+1}\}}\right) &= \max\left(\left[\bigotimes_{\Psi\in\mathcal{S}_i}\Psi\right]^{-X_{i+1}}\right)\\
&= \max\left(\left[\bigotimes_{\Psi\in\mathcal{S}_i\setminus\mathcal{B}_{i+1}}\Psi\right]\otimes\left[\bigotimes_{\Psi\in\mathcal{B}_{i+1}}\Psi\right]^{-X_{i+1}}\right)\\
&= \max\left(\left[\bigotimes_{\Psi\in\mathcal{S}_i\setminus\mathcal{B}_{i+1}}\Psi\right]\otimes\max\left(\left[\bigotimes_{\Psi\in\mathcal{B}_{i+1}}\Psi\right]^{-X_{i+1}}\right)\right)\\
&= \max\left(\left[\bigotimes_{\Psi\in\mathcal{S}_i\setminus\mathcal{B}_{i+1}}\Psi\right]\otimes\Psi_i\right)\\
&= \max\left(\bigotimes_{\Psi\in\mathcal{S}_{i+1}}\Psi\right),
\end{aligned}
$$

where the passage from the first to the second identity follows from element-wise application of (A1) and Lemma 11, the third follows from the second by Lemma 19(i), and the last two follow from the definitions of Ψ_i and $\mathcal{S}_{i+1}$, respectively. $\square$

We are now able to show the correctness of the algorithm in solving LIMIDs exactly.

Theorem 21. *Given a LIMID $\mathcal{L}$, MPU outputs* $\mathrm{MEU}[\mathcal{L}]$.

Proof. The algorithm returns the utility part of a valuation (p,u) in $\max\left(\bigotimes_{\Psi\in\mathcal{S}_n}\Psi\right)$, which, by Lemma 20 for $i=n$, equals $\max\left(\left[\bigotimes_{\Psi\in\mathcal{S}_0}\Psi\right]^{\downarrow\emptyset}\right)$. By definition of $\mathcal{S}_0$, any valuation ϕ in $\left(\bigotimes_{\Psi\in\mathcal{S}_0}\Psi\right)$ satisfies

$$
\phi = \left[\bigotimes_{C\in\mathcal{C}}\left(p_C^{\mathtt{pa}_C},0\right)\right]\otimes\left[\bigotimes_{D\in\mathcal{D}}(I_{\boldsymbol{d}},0)\right]\otimes\left[\bigotimes_{V\in\mathcal{V}}(1,u_V)\right],
$$

for some combination of decisions $(\boldsymbol{d})\in\Omega_{\mathcal{D}}$, which corresponds to a strategy in Δ, and there is exactly one valuation $\phi\in\left(\bigotimes_{\Psi\in\mathcal{S}_0}\Psi\right)$ for each strategy in Δ. Hence, by Proposition 12, the set $\left(\bigotimes_{\Psi\in\mathcal{S}_0}\Psi\right)^{\downarrow\emptyset}$ contains a pair $(1,\mathrm{E}_s[\mathcal{L}])$ for every strategy s inducing a distinct expected utility. Moreover, since functions with empty scope correspond to numbers, the relation $\leq$ specifies a total ordering over the valuations in $\left(\bigotimes_{\Psi\in\mathcal{S}_0}\Psi\right)^{\downarrow\emptyset}$, which implies a single maximal element. Let s^* be a strategy associated to (p,u). Since $(p,u)\in\max\left(\left[\bigotimes_{\Psi\in\mathcal{S}_0}\Psi\right]^{\downarrow\emptyset}\right)$, it follows from maximality that $\mathrm{E}_{s^*}[\mathcal{L}]\geq\mathrm{E}_s[\mathcal{L}]$ for all s, and hence $u=\mathrm{MEU}[\mathcal{L}]$. $\square$

3.7 Complexity Analysis

As with any variable elimination, the complexity of the algorithm depends on the permutation σ given as input. The time complexity of the algorithm is given by the cost of creating the sets of valuations in the initialization step plus the overall cost of the combination and marginalization operations performed during the propagation step. Regarding the initialization step, the loops for chance and value variables generate singletons, and thus take time linear in the input. Since decision nodes have no parents, each set Ψ_D added due to a decision variable D contains $\rho_D \triangleq |\Omega_D|$ valuations. Let $\rho \triangleq \max_{D \in \mathcal{D}} \rho_D$ be the cardinality of the largest domain of a decision variable. Then the initialization loop for decision variables takes $O(|\mathcal{D}|\rho)$ time, which is polynomial in the input. Let us now analyze the propagation step. The running time of propagating (sets of) valuations is exponential in the maximum number of variables in the scope of the valuations generated during the loop step. This number depends on the permutation σ chosen and is in the best case equal to the treewidth of the diagram plus one. Although finding an optimal permutation (i.e., one that leads to a minimum maximum number of variables per scope) is an NP-hard task, we can generate permutations σ using the standard heuristics for variable elimination in Bayesian networks, such as minimizing the number of fill-ins or the cardinality of the domain of the neighbor set, which have been empirically shown to produce good elimination orderings (Jensen & Nielsen, 2007; Koller & Friedman, 2009).

Consider a permutation σ that induces a maximum number of variables per scope of ω, and a diagram with bounded number of states per variable κ. Then the cost of each combination or marginalization is bounded by a constant, and the complexity depends only on the number of operations performed. Moreover, we have in this case that $\rho \leq \kappa$. Let ν denote the cardinality of the largest set Ψ_i, for $i = 1, \ldots, n$. Thus, computing Ψ_i requires at most $\nu^{|\mathcal{U}|-1}$ operations of combination (because that is the maximum number of sets that we might need to combine to compute $\bigotimes_{\Psi \in \mathcal{B}_i} \Psi$ in the propagation step) and ν operations of marginalization. In the worst case, ν is equal to $\rho^{|\mathcal{D}|} \leq O(\kappa^{|\mathcal{D}|})$, that is, all sets associated to decision variables have been combined without discarding any valuation. Hence, the worst-case complexity of the propagation step is exponential in the number of decision variables, even if the width of the elimination ordering and the number of states per variable are bounded. Note however that this is a very pessimistic scenario and, on average, the removal of non-maximal elements greatly reduces the complexity, as the experiments in Section 4 show.

3.8 Reverse Topological Ordering

The valuations used by MPU specify twice as many numbers as the cardinality of the domain of their associated scope. It is possible to decrease the number of numerical parameters per valuation the algorithm needs to handle by a factor of two by constraining the elimination of variables to follow a reverse topological ordering according to the diagram, that is, by requiring each variable to be processed only after all its descendants have been processed. As the following result shows, any reverse topological ordering produces valuations whose probability part equals one in all coordinates.

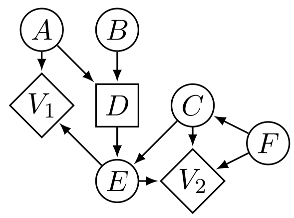


Figure 5: A LIMID in which a reverse topological ordering increases treewidth.

Proposition 22. *If σ defines a reverse topological ordering over the variables in $\mathcal{C} \cup \mathcal{D}$, then for $i = 1, \ldots, n$ the valuations in Ψ_i have probability part $p = 1$, where 1 is the function that always returns the unity.*

Proof. We show the result by induction on i. Regarding the basis, we have from the reverse topological ordering that X_1 is a variable containing only value nodes as children. Hence, $\mathcal{B}_1 = \{\Psi_{X_1}\} \cup \{\{(1, u_V)\} : V \in \mathtt{ch}_{X_1}\}$, where by definition Ψ_{X_1} equals $\{(p_{X_1}^{\mathtt{pa}_{X_1}}, 0)\}$ if X_1 is a chance node, and $\{(p_{X_1}^{\mathtt{pa}_{X_1}}, 0) : p_{X_1}^{\mathtt{pa}_{X_1}} \in \mathcal{P}_{X_1}\}$ if it is a decision node. It follows that $\Psi_1 = \max(\{(\sum_{X_1} p_{X_1}^{\mathtt{pa}_{X_1}}, \sum_{X_1} p_{X_1}^{\mathtt{pa}_{X_1}} \sum_{V \in \mathtt{ch}_{X_1}} u_V)\})$. Since for any $\boldsymbol{\pi} \in \Omega_{\mathtt{pa}_{X_1}}$, $p_{X_1}^{\boldsymbol{\pi}}$ is a probability mass function over X_1, we have that $p = \sum_{X_1} p_{X_1}^{\mathtt{pa}_{X_1}} = 1$. Assume by inductive hypothesis that the result holds for $1, \ldots, i-1$, and let $\Psi_x \triangleq \bigotimes_{\Psi \in \mathcal{B}_i \setminus \mathcal{S}_0} \Psi$. Then $\Psi_i = \max([\bigotimes_{\Psi \in \mathcal{B}_i \cap \mathcal{S}_0} \Psi] \otimes \Psi_x)$. By inductive hypothesis all valuations in a set Ψ in $\mathcal{B}_i \setminus \mathcal{S}_0$ have probability part $p = 1$. Hence, by definition of combination, the valuations in Ψ_x contain also probability part equal to one. The reverse topological ordering implies that by the time variable X_i is processed in the propagation step, all its children have been processed. Hence, the only element of $\mathcal{B}_i \cap \mathcal{S}_0$ is the set Ψ_{X_i}, which equals $\{(p_{X_i}^{\mathtt{pa}_{X_i}}, 0)\}$ if X_i is a chance node, $\{(p_{X_i}^{\mathtt{pa}_{X_i}}, 0) : p_{X_i}^{\mathtt{pa}_{X_i}} \in \mathcal{P}_{X_i}\}$ if X_i is a decision node, and $\{(1, u_{X_i})\}$ if it is a value node. Thus, we have that $\Psi_i = \max(\Psi_{X_i} \otimes \Psi_x)$. The case when X_i is a value node is immediate, since any valuation in Ψ_i is the result of a combination of two valuations with probability part equal to one. If X_i is not a value node then

$$\Psi_i = \max\left(\left\{\left(\sum_{X_i} p_{X_i}^{\mathtt{pa}_{X_i}}, \sum_{X_i} p_{X_i}^{\mathtt{pa}_{X_i}} u_x\right) : (p_{X_i}^{\mathtt{pa}_{X_i}}, 0) \in \Psi_{\mathtt{fa}_{X_i}}, (1, u_x) \in \Psi_x\right\}\right)$$

$$= \max\left(\left\{\left(1, \sum_{X_i} p_{X_i}^{\mathtt{pa}_{X_i}} u_x\right) : (p_{X_i}^{\mathtt{pa}_{X_i}}, 0) \in \Psi_{X_i}, (1, u_x) \in \Psi_x\right\}\right),$$

since $p_{X_i}^{\boldsymbol{\pi}}$ is a probability mass function for any $\boldsymbol{\pi} \in \Omega_{\mathtt{pa}_{X_i}}$. ☐

The result states that if we assume a reverse topological elimination ordering, then MPU needs to care only about the utility part of the valuations. Unfortunately, constraining the elimination order might increase the complexity of the algorithm, as the following example shows.

Consider the LIMID in Figure 5, where all variables are assumed binary (we omit the specification of probabilities and utilities as they are not relevant for the matter). After the initialization, we have that $\mathcal{S}_0 = \{\Psi_A, \Psi_B, \Psi_C, \Psi_D, \Psi_E, \Psi_F, \Psi_{V_1}, \Psi_{V_2}\}$. Using a reverse topological elimination ordering implies we first have to eliminate E, which generates the set

$$\Psi_1 = \max\left([\Psi_E, \otimes \Psi_{V_1} \otimes \Psi_{V_2}]^{-E} \right) = \{(1, u_1)\} \,,$$

whose single element $(1, u_1)$ has scope $\{A, C, D, F\}$ and size $2^4 = 16$. Eliminating variables in the ordering F, C, B, A, D, E, on the other hand, generates the following sets.

$$\begin{aligned}
\Psi_1 &= \max\left([\Psi_F \otimes \Psi_{V_2} \otimes \Psi_C \otimes]^{-F} \right) = \{(p_1, u_1)\} \ , \\
\Psi_2 &= \max\left([\Psi_C \otimes \Psi_E \otimes \Psi_1 \otimes]^{-C} \right) = \{(p_2, u_2)\} \ , \\
\Psi_3 &= \max\left([\Psi_B \otimes \Psi_D]^{-B} \right) = \left\{ (p_3^{(\boldsymbol{d})}, u_3^{(\boldsymbol{d})}) : \boldsymbol{d} \in \Omega_D \right\} \ , \\
\Psi_4 &= \max\left([\Psi_A \otimes \Psi_{V_1} \otimes \Psi_3]^{-A} \right) = \left\{ (p_4^{(\boldsymbol{d})}, u_4^{(\boldsymbol{d})}) : \boldsymbol{d} \in \Omega_D \right\} \ , \\
\Psi_5 &= \max\left([\Psi_2 \otimes \Psi_4]^{-D} \right) = \left\{ (p_5^{(\boldsymbol{d})}, u_5^{(\boldsymbol{d})}) : \boldsymbol{d} \in \Omega_D \right\} \ , \\
\Psi_6 &= \max\left(\Psi_5^{-E} \right) = \left\{ (1, u_6^{(\boldsymbol{d})}) : \boldsymbol{d} \in \Omega_D \right\} \ .
\end{aligned}$$

The scopes of the valuations in $\Psi_1, \Psi_2, \Psi_3, \Psi_4, \Psi_5$ and Ψ_6 are, respectively, $\{E, C\}$, $\{D, E\}$, $\{D, A\}$, $\{E, D\}$, $\{E\}$ and $\{\}$. As one can see, the largest valuation generated using ordering F, C, B, A, D, E contains two variables in its scope and therefore has size $2^2 = 4$. This is a four-fold decrease in size compared to the size of the set Ψ_1 generated using the reverse topological ordering.

Notice however that even though using reverse topological ordering might increase the size of the valuations generated during variable elimination, it does not necessarily results in higher complexity for the MPU. This is because the overall complexity of the algorithm depends not only on the size of the largest valuation generated but also on the cardinality of the generated sets, and it is possible that a reverse topological ordering induces significantly smaller sets, as it produces valuations whose probability parts are always equal to one, which might increase the number of dominated elements.

4. Experiments

We evaluate the performance of the algorithm on random LIMIDs generated in the following way. Each LIMID is parameterized by the number of decision nodes $d \triangleq |\mathcal{D}|$, the number of chance nodes $c \triangleq |\mathcal{C}|$, the maximum cardinality of the domain of the family of a chance variable $\omega_C \triangleq \max_C |\Omega_{\mathtt{fa}_C}|$, and the maximum cardinality of the domain of the family of a decision variable $\omega_D \triangleq \max_D |\Omega_{\mathtt{fa}_D}|$. We set the number of value nodes v to be $d + 2$. For each variable X_i, $i = 1, \ldots, c + d + v$, we sample Ω_{X_i} to contain from 2 to 4 states. Then we repeatedly add an arc from a decision node with no children to a value node with no parents (so that each decision node has at least one value node as children). This step guarantees that all decisions are relevant for the computation of the MEU. Finally, we repeatedly add an arc that neither makes the domain of a variable greater than the given bounds nor makes the treewidth more than 10, until no arcs can be added without exceeding the bounds.[6] Note that this generates diagrams where decision and chance variables have at most $\log_2 \omega_D - 1$ and $\log_2 \omega_C - 1$ parents, respectively. Once the DAG is obtained, we randomly sample the probability mass functions and utility functions associated to chance and value variables, respectively.

We compare MPU against the CR algorithm of de Campos and Ji (2008) in 1620 LIMIDs randomly generated by the described procedure with parameters $5 \leq d \leq 50$, $8 \leq c \leq 50$, $8 \leq \omega_D \leq 64$ and $16 \leq \omega_C \leq 64$. MPU was implemented in C++ and tested in the same computer as CR.[7] Table 1 contrasts the running times of each algorithm (averages $\pm$ standard deviation) for different configurations of randomly generated LIMIDs. Each row contains the percentage of solved diagrams (S_{CR} and S_{MPU}) and time performance (T_{CR} and T_{MPU}) of each of the algorithms for N diagrams randomly generated using parameters d, c, v, ω_D, and ω_C. For each fixed parameter configuration, MPU outperforms CR by orders of magnitude (line 12 contains the only case in which the average running time of CR is lower than MPU's, but note that in this case CR solve it only one instance, whereas MPU solved 86% of the instances). Also, CR was unable to solve most of the diagrams with more than 50 variables, whereas MPU could solve diagrams containing up to 150 variables and with $\omega_D \leq 32$. Both algorithms failed to solve diagrams with $\omega_D = 64$. A diagram is consider unsolved by an algorithm if the algorithm was not able to reach the exact solution within the limit of 12 hours. All in all, MPU appears to scale well on the number of nodes (i.e., on d, c and v) but poorly on the domain cardinality of the family of decision variables (i.e., on ω_D).

A good succinct measure of the hardness of solving a LIMID is the total number of strategies $|\Delta|$, which represents the size of the search space in a brute-force approach. $|\Delta|$ can also be loosely interpreted as the total number of alternatives (over all decision variables) in the problem instance. Figure 6 depicts running time against number of strategies in a log-log scale for the two algorithms on the same test set of random diagrams. For each algorithm, only solved instances are shown, which covers approximately 96% of the cases for MPU, and 68% for CR. We note that MPU solved all cases that CR solved (but not the opposite). Again, we see that MPU is orders of magnitude faster than CR. Within the limit of 12 hours, MPU was able to compute diagrams containing up to 10^{64} strategies, whereas CR solved diagrams with at most 10^{25} strategies.

The reduction in complexity obtained by the removal of non-maximal valuations during the propagation step can be checked in Figure 7, which shows the maximum cardinality of a set Ψ_i generated in the propagation step in contrast to the number of strategies. For each diagram (a point in the figure) solved by MPU, the cardinality of the sets remains bounded above by 10^6 while we vary the number of strategies (which equals the largest cardinality of a propagated set in the worst case where no valuation is discarded). This shows that the worst-case analysis in Section 3.7 is very pessimistic.

6. Since current algorithms for checking whether the treewidth of a graph exceeds a fixed k are too slow for $k \geq 5$ (Bodlaender, 1996), we resort to a greedy heuristic that resulted in diagrams whose actual treewidth ranged from 5 to 10.
7. We used the CR implementation available at `http://www.idsia.ch/~cassio/id2mip/` and CPLEX (`http://www.ilog.com`) as mixed integer programming solver. Our implementation of MPU can be downloaded at `http://www.idsia.ch/~cassio/mpu/`.

N	d	c	v	ω_D	ω_C	S_{CR} (%)	T_{CR} (s)	S_{MPU} (%)	T_{MPU} (s)
60	5	8	7	12	16	100	6 ± 45	100	0.006 ± 0.01
60	5	8	7	16	16	100	9 ± 43	100	0.02 ± 0.05
60	5	8	7	8	16	100	6 ± 51	100	0.002 ± 0.01
60	10	8	12	12	16	98	15 ± 53	100	0.02 ± 0.02
60	10	8	12	16	16	93	107 ± 273	100	103 ± 786
60	10	8	12	8	16	100	0.4 ± 0.2	100	0.007 ± 0.01
60	10	28	12	12	16	96	1175 ± 6126	100	0.05 ± 0.08
60	10	28	12	16	16	83	3340 ± 8966	100	0.2 ± 0.2
30	10	28	12	16	64	10	2838 ± 1493	96	47 ± 142
30	10	28	12	32	16	93	1070 ± 2461	100	0.2 ± 0.4
60	10	28	12	32	32	0	—	93	905 ± 2847
30	10	28	12	32	64	3	73 ± 0	86	2440 ± 7606
30	10	28	12	64	64	0	—	0	—
60	10	28	12	8	16	100	1 ± 3	100	0.01 ± 0.007
60	20	8	22	12	16	93	2687 ± 7564	100	155 ± 1196
90	20	8	22	16	16	38	5443 ± 10070	98	270 ± 1822
30	20	8	22	16	64	30	9660 ± 10303	100	29 ± 84
60	20	8	22	32	32	0	—	78	938 ± 1417
30	20	8	22	32	64	0	—	76	1592 ± 3402
30	20	8	22	64	64	0	—	0	—
60	20	8	22	8	16	100	7 ± 20	100	0.02 ± 0.008
60	10	78	12	16	16	60	5944 ± 9920	100	0.5 ± 0.5
30	10	78	12	32	16	70	3820 ± 8127	100	0.6 ± 1
60	20	58	22	12	16	50	6455 ± 9344	100	522 ± 4011
60	20	58	22	16	16	11	11895 ± 12662	100	2 ± 11
60	20	58	22	8	16	96	849 ± 4098	100	0.07 ± 0.04
60	30	38	32	12	16	28	3416 ± 4827	98	35 ± 214
30	30	38	32	16	16	0	—	100	2 ± 10
60	30	38	32	8	16	96	2261 ± 6572	100	0.1 ± 0.03
30	30	88	32	12	16	0	—	100	230 ± 1027
30	30	88	32	8	16	60	3448 ± 5837	100	0.2 ± 0.1
30	50	48	52	12	16	0	—	96	1753 ± 7405
30	50	48	52	8	16	10	5014 ± 2974	100	0.5 ± 0.09

Table 1: Performance of MPU and CR on randomly generated LIMIDs (numbers are rounded down).

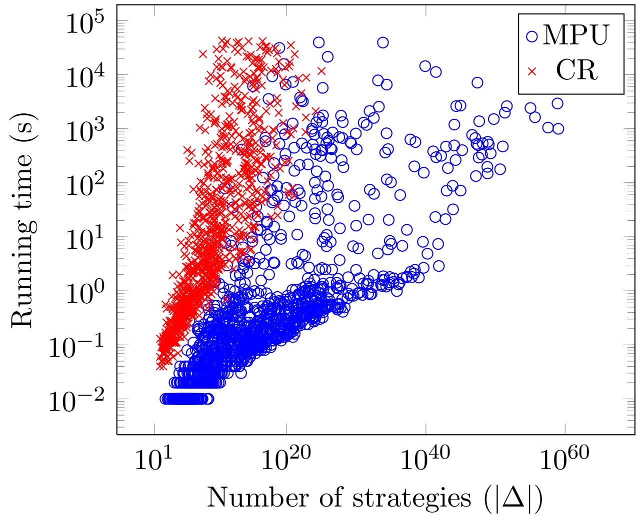


Figure 6: Running time of MPU and CR on randomly generated LIMIDs.

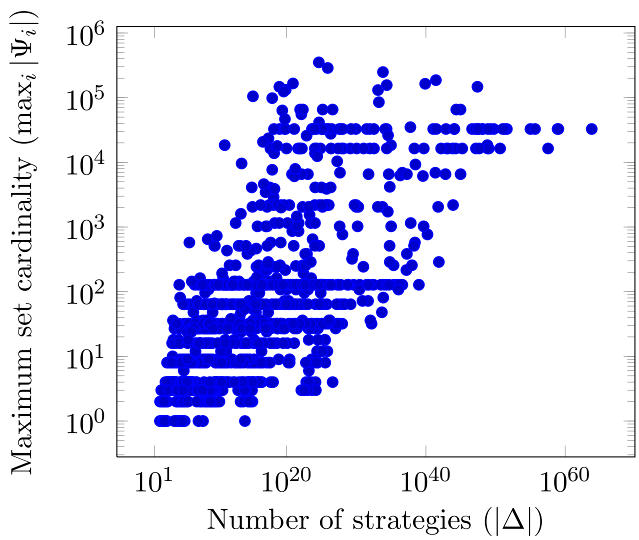


Figure 7: Maximum number of valuations in a set during the propagation step of MPU.

5. Related Work

Influence diagrams were introduced by Howard and Matheson (1984) as a concise language for the specification of utility-based decision problems. There is a substantial literature that formalizes influence diagrams and develop algorithms under the premises of no forgetting and regularity (Cooper, 1988; Qi & Poole, 1995; Shachter & Peot, 1992). We point the interested reader to the works of Jensen and Nielsen (2007) and Koller and Friedman (2009).

Zhang et al. (1994) studied families of LIMIDs that could be solved by dynamic programming, such as LIMIDs respecting no forgetting and regularity. The SPU algorithm of Lauritzen and Nilsson (2001) solves these cases in polynomial time if the diagram has bounded treewidth. To the best of our knowledge, the only attempt to (globally) solve arbitrary LIMIDs exactly without recurring to an exhaustive search on the space of strategies is the CR algorithm of de Campos and Ji (2008) against which we compare our algorithm.

Shenoy and Shafer (1990) introduced the framework of valuation algebras, which states the basic algebraic requirements for efficient computation with valuations. More recently, Haenni (2004) incorporated partially ordered preferences in the algebra to enable approximate computation. Fargier et al. (2010) then extended the framework with a preference degree structure in order to capture the common algebraic structure of optimization problems based on a partial order. The algebra we develop in Section 3 can be partly casted in this framework.

The PFU framework of Pralet, Verfaillie, and Schiex (2007) subsumes many formalisms of probabilistic reasoning, constraint satisfaction and decision making. When it comes to decision problems the framework is geared towards sequential decision making under equivalent assumptions of non-forgetting, although the authors mention the possibility of extending it to limited information decision scenarios.

The variable elimination algorithm we develop here is conceptually close to the message passing algorithm of Dubus, Gonzales, and Perny (2009). Their algorithm, however, does not handle uncertainty and target primarily the obtention of Pareto-efficient solutions for a specific class of multi-objective optimization problems.

There is a close relation between maximum a posteriori (MAP) inference in Bayesian networks and LIMIDs whose decision variables have no parents. In this sense, the algorithm of de Campos (2011), which solves MAP by propagating Pareto efficient probability potentials in a join tree, relates to ours.

6. Conclusion

Solving limited memory influence diagrams is a very hard task. The complexity results presented here show that the problem is NP-hard even for diagrams with bounded treewidth and number of states per variable, and that obtaining provably good approximations in polynomial time is unlikely if the number of states is not small.

Despite the theoretical hardness of the problem, we developed an algorithm that in spite of its exponential worst-case complexity performed empirically well on a large set of randomly generated problems. The algorithm's efficiency is based on the early removal of suboptimal solutions, which helps the algorithm to drastically reduce the search space.

Designing good heuristics for elimination orderings with our algorithm seems to be a more complex task than with standard variable elimination algorithms (e.g., for belief updating in Bayesian networks), because there is a second component, the cardinality of a set, that together with domain cardinalities we wish to minimize. In fact, some preliminary experimentation has shown that favoring set cardinality at expense of domain cardinality might be a good approach. Unlike standard variable elimination, given an elimination ordering and a LIMID, it does not seem to be possible to determine the true complexity of MPU in advance (i.e., prior to running the algorithm). It is an open question whether MPU's complexity can be estimated beforehand, and which heuristics for finding elimination orderings perform better.

Acknowledgments

This work was partially supported by the Swiss National Science Foundation (SNSF) grants no. 200020_134759/1, 200020_137680/1 and 200020_132252, Hasler Foundation grant no. 10030, and the Canton Ticino Computational Life Sciences Project. We thank the reviewers for pointing us to related work and making a number of comments that helped us improve the readability of the paper. A short version of this paper appeared in NIPS '11 (Mauá & de Campos, 2011).

Appendix A. Missing Proofs

This section contains long proofs and supporting results that were left out of the main part of the text to improve readability.

The following two lemmas are used in the proof of Theorem 1 later on.

Lemma 23. *If $\alpha \geq -2$ is a real number and i is a nonnegative integer then $2^\alpha + 2^{-(i+3)} < 2^{\alpha+2^{-i}}$.*

Proof. Since $2^\alpha \geq 2^{-2}$, we have that $2^\alpha + 2^{-(i+3)} = 2^\alpha + 2^{-2} \cdot 2^{-i-1} \leq 2^\alpha(1 + 2^{-i-1})$, and it is sufficient to show that $1 + 2^{-i-1} < 2^{2^{-i}}$. From the Binomial Theorem we have that

$$(1 + 2^{-i-1})^{2^i} = \sum_{k=0}^{2^i} \binom{2^i}{k} (2^{-i-1})^k \,.$$

For $k = 0, \ldots, 2^i$, we have that

$$\binom{2^i}{k} = \frac{2^i(2^i-1)\cdots(2^i-k+1)}{k!} \leq (2^i)^k \,.$$

Hence,

$$(1 + 2^{-i-1})^{2^i} \leq \sum_{k=0}^{2^i} (2^i)^k (2^{-i-1})^k = \sum_{k=0}^{2^i} 2^{-k} \leq \sum_{k=0}^{\infty} 2^{-k} = 2 \,,$$

and therefore $1 + 2^{-i-1} < 2^{2^{-i}}$. □

Lemma 24. *If $0 \leq x \leq 1/2$ then $2^{x-1} + 2^{-x-1} \geq 2^{x^4}$.*

Proof. We obtain the result by approximating the functions on the left- and right-hand side of the inequalities by their truncated Taylor expansions $f(x)$ and $g(x)$, respectively, and then showing that $2^{x-1} + 2^{-x-1} \geq f(x) \geq g(x) \geq 2^{x^4}$. The n-th order Taylor expansion of the left-hand side around zero is given by

$$T_n(x) = 1 + \sum_{k=1}^{n/2} \frac{[\ln(2)]^{2k}}{(2k)!} x^{2k} \,.$$

Clearly, the series converges and hence $2^{x-1} + 2^{-x-1} = \lim_{n\to\infty} T_n(x)$. Moreover, for any n, the residual $R_n(x) = 2^{x-1} + 2^{-x-1} - T_n(x)$ is positive because the terms of the sum are all non negative. Thus,

$$f(x) = T_2(x) = 1 + \frac{[\ln(2)]^2}{2} x^2 \leq 2^{x-1} + 2^{-x-1} .$$

In a similar fashion, we apply the variable change $y = x^4$ on the right-hand side and obtain its Taylor expansion around zero, given by

$$\begin{aligned} T'_n(y) &= 1 + \sum_{k=1}^{n} \frac{[\ln(2)]^k}{k!} y^k \\ &= 1 + \sum_{k=1}^{n} \frac{[\ln(2)]^k}{k!} x^{4k} , \end{aligned}$$

which also converges and has positive residual. Hence,

$$\begin{aligned} 2^{x^4} &= \lim_{n\to\infty} T'_n(x) \\ &= 1 + x^4 \ln(2) + x^2 \ln(2) \left(\sum_{k=2}^{\infty} \frac{[\ln(2)]^{k-1}}{k!} x^{4k-2} \right) \\ &\leq 1 + x^4 \ln(2) + x^2 \ln(2) \left(\sum_{k=2}^{\infty} \frac{1}{2^{4k-1}} \right) \\ &= 1 + x^4 \ln(2) + \frac{[\ln(2)]^2}{32} x^2 = g(x) . \end{aligned}$$

The inequality is obtained by noticing that $[\ln(2)]^{k-1}/k! < 1/2$, $x \leq 1/2 \leq \ln(2)$ and that the geometric series

$$\sum_{k=2}^{\infty} \frac{1}{2^{4k-1}} = \frac{1}{2^7} \sum_{k=0}^{\infty} \left(\frac{1}{2^4}\right)^k < \frac{1}{2^7} \sum_{k=0}^{\infty} \left(\frac{1}{2}\right)^k = \frac{1}{2^6} < \frac{\ln(2)}{32} .$$

Finally, since $x^2 \leq 1/4 < 15\ln(2)/32$ we have that

$$\begin{aligned} g(x) &= 1 + x^2 \ln(2) \left(x^2 + \frac{\ln(2)}{32} \right) \\ &< 1 + x^2 \ln(2) \left(\frac{15}{32} \ln(2) + \frac{\ln(2)}{32} \right) \\ &= 1 + \frac{[\ln(2)]^2}{2} x^2 = f(x) . \end{aligned}$$

Hence, $2^{x^4} \leq g(x) \leq f(x) \leq 2^{x-1} + 2^{-x-1}$ and the result holds. □

The following result shows that solving LIMIDs is NP-hard even if we assume bounded treewidth and number of states per variable.

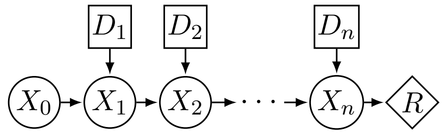


Figure 8: LIMID used to solve the partition problem in the Proof of Theorem 1.

Proof of Theorem 1. Given a strategy s, deciding whether $E_s[\mathcal{L}] > k$ can be done in polynomial time (Koller & Friedman, 2009), so the problem is in NP. Hardness is shown using a reduction from the *partition* problem, which is NP-complete (Garey & Johnson, 1979) and can be stated as follows. *Given a set of n positive integers $a_1, \ldots, a_n$, is there a set $\mathcal{I} \subset \mathcal{A} = \{1, \ldots, n\}$ such that $\sum_{i \in \mathcal{I}} a_i = \sum_{i \in \mathcal{A} \setminus \mathcal{I}} a_i$?* We assume that $n > 3$.

Let $a = \frac{1}{2} \sum_{i \in \mathcal{A}} a_i$. An *even partition* is a subset $\mathcal{I} \subset \mathcal{A}$ that achieves $\sum_{i \in \mathcal{I}} a_i = a$. To solve *partition*, we consider the rescaled problem (dividing every element by a), so that $v_i = a_i/a \leq 2$ are the elements and we look for a partition such that $\sum_{i \in \mathcal{I}} v_i = 1$ (because $\sum_{i \in \mathcal{A}} v_i = 2$).

Consider the following LIMID with topology as in Figure 8. There are n binary decision nodes labeled $D_1, \ldots, D_n$. Each decision D_i can take on states $\boldsymbol{d}_1$ and $\boldsymbol{d}_2$. The chain of chance nodes has $n+1$ ternary variables $X_0, X_1, \ldots, X_n$ with states $\boldsymbol{x}$, $\boldsymbol{y}$, and $\boldsymbol{z}$. There is an arc from X_n to the single value node R. For notational purposes, we specify a function f over the domain $\{\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}\}$ as a triple $(f(\boldsymbol{x}), f(\boldsymbol{y}), f(\boldsymbol{z}))$. The value node has an associated utility function $u_R = (0, 0, 1)$. For $i = 1, \ldots, n$, each chance node X_i has an associated set of conditional probability mass functions given by

$$
\begin{aligned}
p_{X_i}^{\boldsymbol{d}_1, \boldsymbol{x}} &= (t_i, 0, 1 - t_i), & \qquad p_{X_i}^{\boldsymbol{d}_2, \boldsymbol{x}} &= (1, 0, 0), \\
p_{X_i}^{\boldsymbol{d}_1, \boldsymbol{y}} &= (0, 1, 0), & \qquad p_{X_i}^{\boldsymbol{d}_2, \boldsymbol{y}} &= (0, t_i, 1 - t_i), \\
p_{X_i}^{\boldsymbol{d}_1, \boldsymbol{z}} &= (0, 0, 1), & \qquad p_{X_i}^{\boldsymbol{d}_2, \boldsymbol{z}} &= (0, 0, 1),
\end{aligned}
$$

for $t_i \in [0, 1]$ (we specify these variables later on). Note that $p_{X_i}^{D_i X_{i-1}}(\boldsymbol{w}) = 0$ for every $\boldsymbol{w} \in \Omega_{\mathtt{fa}_{X_i}}$ such that $\boldsymbol{w}^{\downarrow X_i} \neq \boldsymbol{w}^{\downarrow X_{i-1}}$ and $\boldsymbol{w}^{\downarrow X_i} \neq \boldsymbol{z}$. Finally, we define $p_{X_0} = (1/3, 1/3, 1/3)$.

Given a strategy $s = (\delta_{D_1}, \ldots, \delta_{D_n})$, let $\mathcal{I} \triangleq \{i : \delta_{D_i} = \boldsymbol{d}_1\}$ be the index set of policies in s such that $\delta_{D_i}(\lambda) = \boldsymbol{d}_1$. We have that

$$
\begin{aligned}
E_s[\mathcal{L}] &= \sum_{\mathcal{C} \cup \mathcal{D}} \left(p_{X_0} \prod_{i=1}^{n} p_{X_i}^{D_i X_{i-1}} p_{D_i} \right) u_R \\
&= \sum_{X_n} \left(\sum_{\mathcal{C} \cup \mathcal{D} \setminus \{X_n\}} p_{X_0} \prod_{i=1}^{n} p_{X_i}^{D_i X_{i-1}} p_{D_i} \right) u_R \, .
\end{aligned}
$$

Let

$$
p_s \triangleq p_{X_0} \prod_{i=1}^{n} p_{X_i}^{D_i X_{i-1}} p_{D_i}
$$

and

$$p_{X_n} \triangleq \sum_{\mathcal{C}\cup\mathcal{D}\setminus\{X_n\}} p_{X_0} \prod_{i=1}^{n} p_{X_i}^{D_i X_{i-1}} p_{D_i} = \sum_{\mathcal{C}\cup\mathcal{D}\setminus\{X_n\}} p_s \, .$$

For $\boldsymbol{w} \in \Omega_{\mathcal{C}\cup\mathcal{D}}$ such that $\boldsymbol{w}^{\downarrow X_n} = \boldsymbol{x}$ (i.e., for $\boldsymbol{w} \in \boldsymbol{x}^{\uparrow \mathcal{C}\cup\mathcal{D}}$) it follows that $p_{X_n}^{D_n X_{n-1}}(\boldsymbol{w}^{\downarrow \mathtt{fa}_{X_n}}) \neq 0$ if and only if $\boldsymbol{w}^{\downarrow X_{n-1}} = \boldsymbol{x}$. But for $\boldsymbol{w}^{\downarrow X_{n-1}} = \boldsymbol{x}$ we have that $p_{X_{n-1}}^{D_{n-1} X_{n-2}}(\boldsymbol{w}^{\downarrow \mathtt{fa}_{X_{n-1}}}) \neq 0$ if and only if $\boldsymbol{w}^{\downarrow X_{n-2}} = \boldsymbol{x}$ and so recursively. Also, for any $i \in \{1, \ldots, n\}$, $p_{X_i}^{D_i X_{i-1}}(\boldsymbol{w}^{\downarrow \mathtt{fa}_{X_i}})$ equals t_i if $i \in \mathcal{I}$ and 1 otherwise. Hence,

$$p_s(\boldsymbol{w}) = \begin{cases} \frac{1}{3}\prod_{i\in\mathcal{I}} t_i, & \text{if } \boldsymbol{w}^{\downarrow X_i} = \boldsymbol{x} \text{ for } i = 1, \ldots, n-1 \\ 0, & \text{otherwise,} \end{cases}$$

and

$$p_{X_n}(\boldsymbol{x}) = \sum_{\boldsymbol{w}\in\boldsymbol{x}^{\uparrow \mathcal{C}\cup\mathcal{D}}} p_s(\boldsymbol{w}) = \frac{1}{3}\prod_{i\in\mathcal{I}}^{n} t_i \, .$$

Likewise, it holds for $\boldsymbol{w} \in \boldsymbol{y}^{\uparrow \mathcal{C}\cup\mathcal{D}}$ that

$$p_s(\boldsymbol{w}) = \begin{cases} \frac{1}{3}\prod_{i\in\mathcal{A}\setminus\mathcal{I}} t_i, & \text{if } \boldsymbol{w}^{\downarrow X_i} = \boldsymbol{y} \text{ for } i = 1, \ldots, n-1 \\ 0, & \text{otherwise,} \end{cases}$$

and therefore

$$p_{X_n}(\boldsymbol{x}) = \frac{1}{3}\prod_{i\in\mathcal{A}\setminus\mathcal{I}}^{n} t_i \, .$$

Since p_{X_n} is a probability mass function on X_n, $p_{X_n}(\boldsymbol{z}) = 1 - p_{X_n}(\boldsymbol{x}) - p_{X_n}(\boldsymbol{y})$, and

$$\begin{aligned} \mathrm{E}_s[\mathcal{L}] &= \sum_{X_n} p_{X_n} u_R \\ &= 1 - p_{X_n}(\boldsymbol{x}) - p_{X_n}(\boldsymbol{y}) \\ &= 1 - \frac{1}{3}\prod_{i\in\mathcal{I}} t_i - \frac{1}{3}\prod_{i\in\mathcal{A}\setminus\mathcal{I}} t_i \, . \end{aligned}$$

Let us assume initially that $t_i = 2^{-v_i}$. The reduction from the original problem in this way is not polynomial, and we will use it only as an upper bound for the outcome of the reduction we obtain later. It is not difficult to see that $\mathrm{E}_s[\mathcal{L}]$ is a concave function of $v_1, \ldots, v_n$ that achieves its maximum at $\sum_{i\in\mathcal{I}} v_i = \sum_{i\in\mathcal{A}\setminus\mathcal{I}} v_i = 1$. Since each strategy s defines a partition of $\mathcal{A}$ and vice-versa, there is an even partition if and only if $\mathrm{MEU}[\mathcal{L}] = 1 - 1/3(1/2 + 1/2) = 2/3$.

We will now show a reduction that encodes the numbers t_i in time and space polynomial in b, the number of bits used to encode the original problem. This part is in close analogy with the last part of the proof of hardness of MAP in Bayesian networks by de Campos (2011, Theorem 10).

By setting t_i to represent 2^{-v_i} with $6b+3$ bits of precision (rounding up if necessary), that is, by choosing t_i so that $2^{-v_i} \leq t_i < 2^{-v_i} + \epsilon_i$, where $0 \leq \varepsilon_i < 2^{-(6b+3)}$, we have that

$2^{-v_i} \leq t_i < 2^{-v_i} + 2^{-(6b+3)}$, which implies (by using Lemma 23 with $\alpha = -v_i \geq -2$ and $i = 6b$) that $2^{-v_i} \leq t_i < 2^{-v_i + 2^{-6b}}$.

Assume that an even partition $\mathcal{I}$ exists. Then[8]

$$\prod_{i \in \mathcal{I}} t_i < 2^{2^{-6b}n - \sum_{i \in \mathcal{I}} v_i} = 2^{-1+2^{-6b}n} \leq 2^{-1+2^{-5b}} \,,$$

$$\prod_{i \in \mathcal{A} \setminus \mathcal{I}} t_i < 2^{2^{-6b}n - \sum_{i \in \mathcal{A} \setminus \mathcal{I}} v_i} = 2^{-1+2^{-6b}n} \leq 2^{-1+2^{-5b}} \,,$$

and

$$\mathrm{MEU}[\mathcal{L}] > 1 - \frac{1}{3}\left(2^{-1+2^{-5b}} + 2^{-1+2^{-5b}}\right) = 1 - \frac{2^{2^{-5b}}}{3}. \tag{6}$$

Let r be equal to $2^{2^{-5b}}$ encoded with $5b+3$ bits of precision (and rounded up), that is, $2^{2^{-5b}} \leq r < 2^{2^{-5b}} + 2^{-(5b+3)}$, which implies (by Lemma 24 with $\alpha = 2^{-5b} \geq -2$ and $i = 5b$) that

$$2^{2^{-5b}} \leq r < 2^{2^{-5b} + 2^{-5b}} = 2^{2^{1-5b}} < 2^{2^{-4b}} \,. \tag{7}$$

The reduction is done by verifying whether $\mathrm{MEU}[\mathcal{L}] > 1 - r/3$. We already know that an even partition has an associated strategy which obtains an expected utility greater than $1 - r/3$, because of Equality (6) and the fact that r is rounded up. Let us consider the case where an even partition does not exist. We want to show that in this case $\mathrm{MEU}[\mathcal{L}] \leq 1 - 2^{2^{-4b}}/3$, which by Inequality (7) implies $\mathrm{MEU}[\mathcal{L}] < 1 - r/3$. Since there is not an even partition, any strategy induces a partition such that, for some integer $-a \leq c \leq a$ different from zero, we have that $\sum_{i \in \mathcal{I}} a_i = a - c$ and $\sum_{i \in \mathcal{A} \setminus \mathcal{I}} a_i = a + c$, because the original numbers a_i are positive integers that add up to $2a$. It follows that

$$\prod_{i \in \mathcal{I}} t_i + \prod_{i \in \mathcal{A} \setminus \mathcal{I}} t_i = 2^{c/a-1} + 2^{-c/a-1} \,.$$

The right-hand side of the equality is a function on $c \in \{-a, \ldots, a\} \setminus \{0\}$, which is symmetric with respect to the y-axis (i.e., $f(c) = f(-c)$) and monotonically increasing for $c > 0$. Therefore, it obtains its minimum at $c = 1$. Hence,

$$\prod_{i \in \mathcal{I}} t_i + \prod_{i \in \mathcal{A} \setminus \mathcal{I}} t_i \geq 2^{1/a-1} + 2^{-1/a-1} \,.$$

Since $n > 3$ implies $a \geq 2$ (because the numbers a_i are positive integers), we have by Lemma 24 that

$$2^{1/a-1} + 2^{-1/a-1} \geq 2^{1/a^4} \,.$$

Each number a_i is encoded with at least $\log_2 a_i$ bits, and therefore $b \geq \log_2(a_1) + \cdots + \log_2(a_n) = \log_2(a_1 \cdots a_n)$. The latter is greater than or equal to $\log_2(a_1 + \cdots + a_n)$, and hence is also greater than $\log_2 a$. Thus, we have that $a \leq 2^b$, which implies $a^4 \leq 2^{4b}$ and therefore $1/a^4 \geq 2^{-4b}$ and $2^{1/a^4} \geq 2^{2^{-4b}}$. Hence,

$$2^{1/a-1} + 2^{-1/a-1} \geq 2^{2^{-4b}} \,.$$

8. Since the number of bits used to encode the partition problem must be greater than or equal to n, we have that $n/2^b \leq n/b \leq 1$, and hence $2^{-(j+1)b}n < 2^{-jb}$, for any $j > 0$.

Thus, if an even partition does not exist we have that

$$\mathrm{MEU}[\mathcal{L}] = 1 - \frac{1}{3}\left(\prod_{i \in \mathcal{I}} t_i + \prod_{i \in \mathcal{A} \setminus \mathcal{I}} t_i\right) \leq 1 - \frac{2^{2^{-4b}}}{3} < 1 - r/3\,.$$

To summarize, we have built a LIMID $\mathcal{L}$ in polynomial time since each t_i was specified using $O(b)$ bits and there are n functions $p_{X_i}^{D_i X_{i-1}}$, each encoding 18 numbers (which are either 1, 0 or t_i), and $2n+2$ variables with bounded number of states. We have shown that there is a one-to-one correspondence between partitions of $\mathcal{A}$ in the original problem and strategies of $\mathcal{L}$, and that for a given rational $r = f(b)$ encoded with $O(b)$ bits the existence of an even partition is equivalent to $\mathrm{MEU}[\mathcal{L}] > 1 - r/3$. ☐

The following lemma is used in the proof of Theorem 2. A similar result has been shown by Park and Darwiche (2004, Lemma 9).

Lemma 25. *For any $x \geq 1$ it follows that $x + 1/2 > 1/\ln(1 + 1/x)$.*

Proof. Let $f(x) = \ln(1 + 1/x) - 1/(x + 1/2)$. Then

$$f'(x) = -\frac{1}{x^2 + x} + \frac{1}{x^2 + x + 1/4}\,,$$

which is strictly negative for $x \geq 1$ since $x^2+x < x^2+x+1/4$. Hence, $f(x)$ is a monotonically decreasing function for $x \geq 1$. Because $\lim_{x \to \infty} f(x) = 0$, $f(x)$ is strictly positive in $[1, \infty)$. Thus, the result follows from $\ln(1 + 1/x) > 1/(x + 1/2)$, since $x \geq 1$. ☐

We now show that approximately solving LIMIDs of bounded treewidth for any given minimum performance is NP-hard.

Proof of Theorem 2. We will show that for any fixed $0 < \gamma < 1$ the existence of a polynomial time 2^{θ^γ}-approximation algorithm for solving a LIMID would imply the existence of a polynomial time algorithm for the CNF-SAT problem, which is known to be impossible unless P=NP (Garey & Johnson, 1979). A very similar reduction was used by Park and Darwiche (2004, Theorem 8) to show an analogous inapproximability result for maximum a posteriori inference in Bayesian networks. Notice that for any $1 < \rho < 2^\theta$ there is $0 < \gamma < 1$ such that $\rho = 2^{\theta^\gamma}$, hence the existence of an ρ-approximation algorithm implies the existence of a 2^{θ^γ}-approximation, and it suffices for the desired result to show that the latter cannot be true (unless P=NP).

A clause is a disjunction of literals, each literal being either a boolean variable or its negation. We say that a clause is satisfied if, given an assignment of truth values to its variables, at least one of the literals evaluates to 1. Thus, we can decide if a truth-value assignment satisfies a clause in time linear in the number of variables. The CNF-SAT problem is defined as follows. *Given a set of clauses $C_1, \ldots, C_m$ over (subsets of) boolean variables $X_1, \ldots, X_n$, is there an assignment of truth values to the variables that satisfies all the clauses?*

For a positive integer q that we specify later on, consider the LIMID obtained as follows (the topology is depicted in Figure 9). For each boolean variable X_i we add q binary

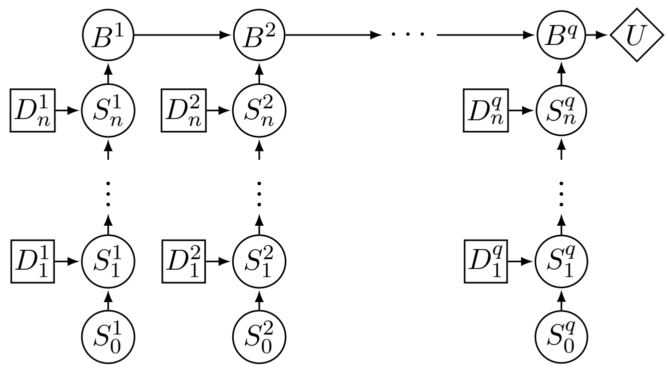


Figure 9: Graph structure of the LIMID used in the proof of Theorem 2.

decision variables $D_i^1, \ldots, D_i^q$ and q chance variables $S_i^1, \ldots, S_i^q$ with domain $\{0, 1, \ldots, m\}$. Additionally, there are q clause selector variables $S_0^1, \ldots, S_0^q$ taking values on $\{1, 2, \ldots, m\}$, q binary variables $B^1, \ldots, B^q$, and a value node U with B^q as parent. As illustrated in Figure 9, the LIMID consists of q replicas of a polytree-shaped diagram over variables $D_1^j, \ldots, D_n^j, S_0^j, \ldots, S_n^j, B^j$, and the probability mass functions for the variables $B^1, \ldots, B^q$ are chosen so as to make the expected utility equal the product of the expected utilities of each replica. In any of the replicas (i.e., for $j \in \{1, \ldots, q\}$), a variable D_i^j $(i = 1, \ldots, n)$ represents an assignment of truth value for X_i and has no parents. The selector variables S_0^j represent the choice of a clause to process, that is, $S_0^j = k$ denotes clause C_k is being "processed", and by summing out S_0^j we process all clauses. Each variable S_i^j, for $i = 1, \ldots, n$ and $j = 1, \ldots, q$, has D_i^j and S_{i-1}^j as parents. The variables B^j have S_n^j and, if $j > 1$, B^{j-1} as parents. For all j, we assign uniform probabilities to S_0^j, that is, $p_{S_0^j} \triangleq 1/m$. For $j = 1, \ldots, q$, we set the probabilities associated to variables $S_1^j, \ldots, S_n^j$ so that if C_k is the clause selected by S_0^j then S_i^j is set to zero if C_k is satisfied by D_i but not by any of $D_1, \ldots, D_{i-1}$, and $S_i^j = S_{i-1}^j$ otherwise. Formally, for $\boldsymbol{x} \in \Omega_{\{S_i^j, D_i^j, S_{i-1}^j\}}$ we have that

$$p_{S_i^j}^{D_i^j S_{i-1}^j}(\boldsymbol{x}) \triangleq \begin{cases} 1, & \text{if } \boldsymbol{x}^{\downarrow S_i^j} = \boldsymbol{x}^{\downarrow S_{i-1}^j} = 0\,; \\ 1, & \text{if } \boldsymbol{x}^{\downarrow S_i^j} = 0 \text{ and } \boldsymbol{x}^{\downarrow S_{i-1}^j} = k \geq 1 \text{ and } X_i = \boldsymbol{x}^{\downarrow D_i} \text{ satisfies } C_k\,; \\ 1, & \text{if } \boldsymbol{x}^{\downarrow S_i^j} = \boldsymbol{x}^{\downarrow S_{i-1}^j} = k \geq 1 \text{ and } X_i = \boldsymbol{x}^{\downarrow D_i} \text{ does not satisfy } C_k\,; \\ 0, & \text{otherwise.} \end{cases}$$

Notice that for S_1^j the first case never occurs since S_0^j takes values on $\{1, \ldots, m\}$. For any joint state configuration $\boldsymbol{x}$ of $S_0^j, \ldots, S_n^j, D_1^j, \ldots, D_n^j$ such that $\boldsymbol{x}^{\downarrow S_0^j} = k \in \{1, \ldots, m\}$ (i.e., clause C_k is being processed) and $\boldsymbol{x}^{\downarrow S_n^j} = 0$, it follows that

$$\left(p_{S_0^j} \prod_{i=1}^{n} p_{S_i^j}^{D_i^j S_{i-1}^j} p_{D_i^j} \right)(\boldsymbol{x})$$

equals $1/m$ only if for some $0 < i \leq n$ clause C_k is satisfied by $X_i = \boldsymbol{x}^{\downarrow D_i}$ but not by any of $X_1 = \boldsymbol{x}^{\downarrow D_1}, \ldots, X_{i-1} = \boldsymbol{x}^{\downarrow D_{i-1}}$, variables $S_1^j, \ldots, S_{i-1}^j$ all assume value k (i.e.,

$\boldsymbol{x}^{\downarrow S_1^j} = \cdots = \boldsymbol{x}^{\downarrow S_{i-1}^j} = k$), and $\boldsymbol{x}^{\downarrow S_i^j} = \cdots = \boldsymbol{x}^{\downarrow S_n^j} = 0$. Otherwise, it equals 0. Hence, for any (partial) strategy $s^j = (\delta_{D_1^j}, \ldots, \delta_{D_n^j})$ we have for $\boldsymbol{x} = 0$ that

$$p_{S_n^j}^{s^j}(\boldsymbol{x}) \triangleq \left(\sum_{\substack{S_0^j,\ldots,S_{n-1}^j \\ D_1^j,\ldots,D_n^j}} p_{S_0^j} \prod_{i=1}^{n} p_{S_i^j}^{D_i^j S_{i-1}^j} p_{D_i^j} \right)(\boldsymbol{x}) = \frac{SAT(s^j)}{m} \,,$$

where $SAT(s^j)$ denotes the number of clauses satisfied by the truth-value assignment of $X_1, \ldots, X_n$ according to s^j. Each variable B^j is associated to a function $p_{B^j}^{S_n^j B^{j-1}}$ such that for $\boldsymbol{x} \in \Omega_{\mathtt{fa}_{B^j}}$,

$$p_{B^j}^{S_n^j B^{j-1}}(\boldsymbol{x}) = \begin{cases} 1, & \text{if } \boldsymbol{x}^{\downarrow B^j} = \boldsymbol{x}^{\downarrow B^{j-1}} \text{ and } \boldsymbol{x}^{\downarrow S_n^j} = 0\,; \\ 1, & \text{if } \boldsymbol{x}^{\downarrow B^j} = 0 \text{ and } \boldsymbol{x}^{\downarrow S_n^j} \neq 0\,; \\ 0, & \text{otherwise;} \end{cases}$$

where for B^1 we assume $\boldsymbol{x}^{\downarrow B^0} = 1$. Hence, we have for any joint state configuration $\boldsymbol{x}$ of $B^1, \ldots, B^q, S_n^1, \ldots, S_n^q$ that

$$\left(\prod_{j=1}^{q} p_{B^j}^{S_n^j B^{j-1}} \right)(\boldsymbol{x}) = \begin{cases} 1, & \text{if } \boldsymbol{x}^{\downarrow B^1} = \cdots = \boldsymbol{x}^{\downarrow B^q} = 1 \text{ and } \boldsymbol{x}^{\downarrow S_n^1} = \cdots = \boldsymbol{x}^{\downarrow S_n^q} = 0; \\ 1, & \text{if } \boldsymbol{x}^{\downarrow B^1} = \cdots = \boldsymbol{x}^{\downarrow B^q} = 0 \text{ and } \boldsymbol{x}^{\downarrow S_n^1} \neq 0; \\ 0, & \text{otherwise.} \end{cases}$$

Finally, we set the utility function u associated to U to return 1 if $B^q = 1$ and 0 otherwise. In this way, $\left(u \prod_{j=1}^{q} p_{B^j}^{S_n^j B^{j-1}}\right)(\boldsymbol{x})$ equals 1 if $\boldsymbol{x}^{\downarrow B^1} = \cdots = \boldsymbol{x}^{\downarrow B^q} = 1$ and $\boldsymbol{x}^{\downarrow S_n^1} = \cdots = \boldsymbol{x}^{\downarrow S_n^q} = 0$ and zero otherwise. Thus, for any strategy $s = (s^1, \ldots, s^q)$, where $s^j = \delta_{D_1^j}, \ldots, \delta_{D_n^j}$, it follows that

$$\begin{aligned}
\mathrm{E}_s[\mathcal{L}] &= \sum_{\mathcal{C} \cup \mathcal{D}} u \prod_{j=1}^{q} p_{B^j}^{S_n^j B^{j-1}} p_{S_0^j} \prod_{i=1}^{n} p_{S_i^j}^{D_i^j S_{i-1}^j} p_{D_i^j} \\
&= \sum_{\substack{B^1,\ldots,B^q \\ S_n^1,\ldots,S_n^q}} u \prod_{j=1}^{q} p_{B^j}^{S_n^j B^{j-1}} \sum_{\substack{S_0^j,\ldots,S_{n-1}^j \\ D_1^j,\ldots,D_n^j}} p_{S_0^j} \prod_{i=1}^{n} p_{S_i^j}^{D_i^j S_{i-1}^j} p_{D_i^j} \\
&= \sum_{\substack{B^1,\ldots,B^q \\ S_n^1,\ldots,S_n^q}} u \prod_{j=1}^{q} p_{B^j}^{S_n^j B^{j-1}} p_{S_n^j}^{s^j} \\
&= \prod_{j=1}^{q} p_{S_n^j}^{s^j}(0) = \frac{1}{m^q} \prod_{j=1}^{q} SAT(s^j)\,.
\end{aligned}$$

If the instance of CNF-SAT problem is satisfiable then there is an optimum strategy s such that $SAT(s^j) = m$ for all j, and $\mathrm{MEU}[\mathcal{L}] = 1$. On the other hand, if the instance is not satisfiable, we have for all j and strategy s that $SAT(s^j) \leq m-1$, and hence $\text{MEU}[\mathcal{L}] \leq (m-1)^q/m^q$. For some given $0 < \gamma < 1$, let q be a positive integer chosen so that $1/2^{\theta^\gamma} > m^q/(m+1)^q$. We show later on that q can be obtained from a polynomial on the input. If the CNF-SAT instance is satisfiable, a 2^{θ^γ}-approximation algorithm for $\text{MEU}[\mathcal{L}]$ returns a value $\text{E}_s[\mathcal{L}]$ such that

$$\text{E}_s[\mathcal{L}] \geq \frac{\text{MEU}[\mathcal{L}]}{2^{\theta^\gamma}} > \left(\frac{m}{m+1}\right)^q > \left(\frac{m-1}{m}\right)^q ,$$

where the rightmost strict inequality follows from $m/(m+1) > (m-1)/m$. On the other hand, if the CNF-SAT instance is not satisfiable, the approximation returns

$$\text{E}_s[\mathcal{L}] \leq \text{MEU}[\mathcal{L}] \leq \left(\frac{m-1}{m}\right)^q .$$

Hence, we can use a 2^{θ^γ}-approximation algorithm to solve CNF-SAT by checking whether its output $\text{E}[\mathcal{L}] > (m-1)^q/m^q$. Since q and m are positive integers, the test bound $(m-1)^q/m^q$ can be obtained in polynomnial time.

It remains to show that the reduction is polynomial in the input. The LIMID contains $q(2n+2)+1$ variables, each requiring the specification of at most $2(m+1)^2$ numbers in $\{0, 1/m, 1\}$. So θ, the number of numerical parameters in $\mathcal{L}$, is polynomially bounded by $q(m+1)^2(4n+4)+2$. Therefore, it suffices to show that q is a polynomial on m and n. By definition, q obeys

$$\left(1+\frac{1}{m}\right)^q > 2^{[q(m+1)^2(4n+4)+2]^\gamma} ,$$

which is equivalent to

$$\begin{aligned} q \ln\left(1+\frac{1}{m}\right) &> q^\gamma[(m+1)^2(4n+4)+2]^\gamma \ln 2 \\ \Leftrightarrow q^{1-\gamma} &> \frac{[(m+1)^2(4n+4)+2]^\gamma}{\ln\left(1+\frac{1}{m}\right)} \ln 2 \\ \Leftrightarrow q &> \left(\frac{[(m+1)^2(4n+4)+2]^\gamma}{\ln\left(1+\frac{1}{m}\right)} \ln 2\right)^{\frac{1}{1-\gamma}} \end{aligned}$$

Since (by Lemma 25) $m+1/2 > 1/\ln(1+1/m)$ and $2 > \ln(2)$, it suffices to choose q such that

$$q > \left((2m+1)[(m+1)^2(4n+4)+2]^\gamma\right)^{\frac{1}{1-\gamma}} .$$

In other words, q is polynomially bounded by $m^{\frac{2\gamma+1}{1-\gamma}} 4n^{\frac{\gamma}{1-\gamma}}$. Therefore, if $\text{MEU}[\mathcal{L}]$ can be approximated in polynomial time with an ratio no greater than 2^{θ^γ} then we can solve CNF-SAT in polynomial time. ☐

The next result show that Transformation 6 preserves expected utility of strategies, and that strategies can be easily mapped back and forward between original and transformed diagrams.

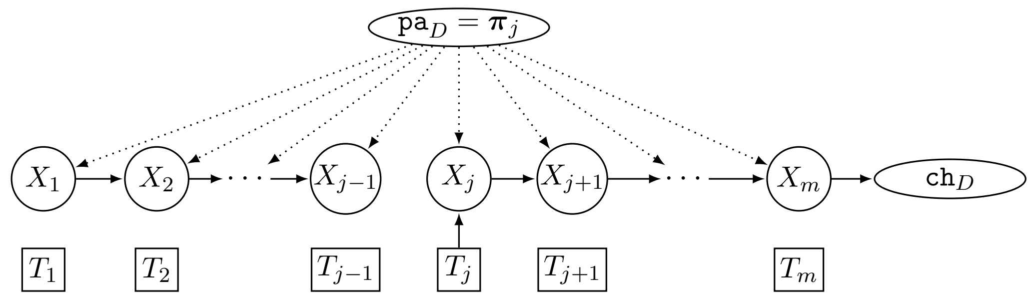


Figure 10: Reasoning of the proof of Transformation 6.

Proof of Proposition 7. By looking at the definition of the functions $p_{X_1}^{T_1,\mathtt{pa}_D}$ and $p_{X_i}^{X_{i-1},T_i,\mathtt{pa}_D}$, for $i = 2, \ldots, m$, we can see that when $\mathtt{pa}_D$ "selects" $\boldsymbol{\pi}_j$, that is, conditional on $\mathtt{pa}_D = \boldsymbol{\pi}_j$, the variable X_j is independent of X_{j-1} and for $i = 1, \ldots, m$, $i \neq j$, the variable X_i is independent of T_i. In other words, we have that

$$\Pr(X_j|X_{j-1}, T_j, \mathtt{pa}_D = \boldsymbol{\pi}_j) = \Pr(X_j|T_j, \mathtt{pa}_D = \boldsymbol{\pi}_j) \triangleq p_{X_j}^{T_j} .$$

and, for any $i \neq j$,

$$\Pr(X_i|X_{i-1}, T_i, \mathtt{pa}_D = \boldsymbol{\pi}_j) = \Pr(X_i|X_{i-1}, \mathtt{pa}_D = \boldsymbol{\pi}_j) \triangleq p_{X_i}^{X_{i-1}} .$$

We can visualize this situation by removing the arc from X_{j-1} to X_j and all the arcs from T_i to X_i for $i \neq j$ in the diagram of Figure 2(b) (the arcs leaving $\mathtt{pa}_D$ can also be removed as we are conditioning on a value of $\Omega_{\mathtt{pa}_D}$), which results in the diagram in Figure 10. Note that in principle the case for $j = 1$ deserves special attention, as the X_1 does not depend on any other X_i variable, and the function associated to X_1 slightly differ from others. Nevertheless, a similar reasoning can be applied. We will omit the case for $j = 1$ for the sake of simplicity.

It follows from the previous reasoning that

$$\begin{aligned}
p_{X_m}^{\boldsymbol{\pi}_j} &\triangleq \Pr(X_m|\mathtt{pa}_D = \boldsymbol{\pi}_j) \\
&= \sum_{T_1,\ldots,T_m} \sum_{X_1,\ldots,X_{m-1}} (p_{X_1} p_{T_1})(p_{X_j}^{T_j} p_{T_j}) \prod_{i=2, i\neq j}^{m} p_{X_i}^{X_{i-1}} p_{T_i} \\
&= \sum_{T_j} p_{T_j} \sum_{X_j,\ldots,X_{m-1}} p_{X_j}^{T_j} \prod_{i=j+1}^{m} p_{X_i}^{X_{i-1}} \underbrace{\sum_{X_1,\ldots,X_{j-1}} p_{X_1} \prod_{i=2}^{j-1} p_{X_i}^{X_{i-1}} \sum_{T_1,\ldots,T_m \setminus T_j} \prod_{i=1, i\neq j}^{m} p_{T_i}}_{=1} \\
&= \sum_{T_j} p_{T_j} \sum_{X_j,\ldots,X_{m-1}} p_{X_j}^{T_j} \prod_{i=j+1}^{m} p_{X_i}^{X_{i-1}} .
\end{aligned}$$

When $\mathtt{pa}_D = \boldsymbol{\pi}_j$, each function $p_{X_i}^{X_{i-1}}$ for $i \neq j$ equals the indicator function $I_{X_i=X_{i-1}}$, by design, and $p_{X_j}^{T_j} = I_{X_j=T_j}$. We thus have that

$$p_{X_m}^{\boldsymbol{\pi}_j} = \sum_{T_j} p_{T_j} \sum_{X_j,\dots,X_{m-1}} I_{X_j=T_j} \prod_{i=j+1}^{m} I_{X_i=X_{i-1}} .$$

For each term of the outer sum over T_j, the inner sum over $X_j, \dots, X_{m-1}$ differs from zero only when $T_j = X_j = X_{j+1} = \dots = X_m$, in which case it equals one. Hence, we have that

$$\begin{aligned} p_{X_m}^{\boldsymbol{\pi}_j} &= \sum_{T_j} p_{T_j} I_{X_m=T_j} \\ &= p_{T_j} . \end{aligned}$$

Now consider a strategy $s' = (\delta_{T_1}, \dots, \delta_{T_m}, \dots)$ for $\mathcal{L}'$, and let $p_{X_m}^{\mathtt{pa}_D}$ be a function that equals $p_{X_m}^{\boldsymbol{\pi}_j}$ for every value $\boldsymbol{\pi}_j \in \Omega_{\mathtt{pa}_D}$. Let also δ_D be a policy for the original decision variable D in $\mathcal{L}$ such that $\delta_D(\boldsymbol{\pi}_j) = \delta_{T_j}$ for all j, and s be a strategy for $\mathcal{L}$ obtained by substituting policies $\delta_{T_1}, \dots, \delta_{T_m}$ with δ_D in s'. Finally, let $p_{T_1}, \dots, p_{T_m}$ be the distributions induced by the policies for $T_1, \dots, T_m$ in s', and $p_D^{\mathtt{pa}_D}$ be the distribution induced by δ_D. Since for each value $\boldsymbol{\pi}_j$ of $\mathtt{pa}_D$ we have that $p_{X_m}^{\mathtt{pa}_D}(\boldsymbol{\pi}_j) = p_{T_j}$, and since by design $\Omega_{X_m} = \Omega_D$, it follows that $p_{X_m}^{\mathtt{pa}_D} = p_D^{\mathtt{pa}_D}$. Hence, for each combination of policies $\delta_{T_1}, \dots, \delta_{T_m}$ in $\mathcal{L}'$ we can derive a corresponding policy in $\mathcal{L}$. The converse is also true: for each policy δ_D we can generate $\delta_{T_1}, \dots, \delta_{T_m}$ such that $p_{X_m}^{\mathtt{pa}_D} = p_D^{\mathtt{pa}_D}$ (simply choose $\delta_{T_i} = \delta_D(\boldsymbol{\pi}_j)$ for all i). Thus, there is a one-to-one correspondence between policies $\delta_{T_1}, \dots, \delta_{T_m}$ and policies δ_D, and a one-to-one correspondence between the induced functions $p_{X_m}^{\mathtt{pa}_D}$ and $p_D^{\mathtt{pa}_D}$.

It remains to show that a combination of policies $\delta_{T_1}, \dots, \delta_{T_m}$ and a corresponding policy δ_D induce the same expected utility. Let $\mathcal{C}'$ and $\mathcal{D}'$ denote, respectively, the set of chance and decision variables in $\mathcal{L}'$, and $\mathcal{C}$ and $\mathcal{D}$ the set of chance and decision variables in $\mathcal{L}$. Also, for a given strategy s for $\mathcal{L}$, let

$$p'_s \triangleq \prod_{\mathcal{C}\cup\mathcal{D}\setminus\{D\}} p_X^{\mathtt{pa}_X} .$$

Note that the above function is independent of the choice of policy δ_D, and that $p_s = p'_s p_D^{\mathtt{pa}_D}$. Given any strategy s' for $\mathcal{L}'$ we have that

$$\begin{aligned} \mathrm{E}_{s'}[\mathcal{L}'] &= \sum_{\mathcal{C}'\cup\mathcal{D}'} p_{s'} \sum_{V\in\mathcal{V}} u_V \\ &= \sum_{\mathcal{C}'\cup\mathcal{D}'} p'_s \left(p_{X_1}^{T_1,\mathtt{pa}_D} \prod_{i=2}^{m} p_{X_i}^{X_{i-1},T_i,\mathtt{pa}_D} \prod_{i=1}^{m} p_{T_i} \right) \sum_{V\in\mathcal{V}} u_V \\ &= \sum_{\mathcal{C}\cup\mathcal{D}\setminus\{D\}} p'_s \left(\sum_{V\in\mathcal{V}} u_V \right) \underbrace{\sum_{\mathcal{C}'\setminus\mathcal{C}} \sum_{\mathcal{D}'\setminus\mathcal{D}} p_{X_1}^{T_1,\mathtt{pa}_D} \prod_{i=2}^{m} p_{X_i}^{X_{i-1},T_i,\mathtt{pa}_D} \prod_{i=1}^{m} p_{T_i}}_{=\sum_{X_m} p_{X_m}^{\mathtt{pa}_D}} \\ &= \sum_{\mathcal{C}\cup\mathcal{D}\setminus\{D\}} \sum_{X_m} p_{X_m}^{\mathtt{pa}_D} p'_s \sum_{V\in\mathcal{V}} u_V \end{aligned}$$

$$= \sum_{\mathcal{C}\cup\mathcal{D}\setminus\{D\}} \sum_{D} p_D^{\mathrm{pa}_D} p'_s \sum_{V\in\mathcal{V}} u_V$$
$$= \sum_{\mathcal{C}\cup\mathcal{D}} p_s \sum_{V\in\mathcal{V}} u_V$$
$$= \mathrm{E}_s[\mathcal{L}]\,,$$

where s is the strategy for $\mathcal{L}$ obtained from s' by substituting $\delta_{T_1}, \ldots, \delta_{T_m}$ with the corresponding policy δ_D. □

References

Bodlaender, H. L. (1996). A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, *25*(6), 1305–1317.

Cooper, G. F. (1988). *A method for using belief networks as influence diagrams.* Fourth Workshop on Uncertainty in Artificial Intelligence.

de Campos, C. P. (2011). New results for the MAP problem in Bayesian networks. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, pp. 2100–2106.

de Campos, C. P., & Ji, Q. (2008). Strategy selection in influence diagrams using imprecise probabilities. In *Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence*, pp. 121–128.

Dechter, R. (1999). Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, *113*(1-2), 41–85.

Detwarasiti, A., & Shachter, R. D. (2005). Influence diagrams for team decision analysis. *Decision Analysis*, *2*, 207–228.

Dubus, J.-P., Gonzales, C., & Perny, P. (2009). Multiobjective optimization using GAI models. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, pp. 1902–1907.

Fargier, H., Rollon, E., & Wilson, N. (2010). Enabling local computation for partially ordered preferences. *Constraints*, *15*, 516–539.

Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman.

Haenni, R. (2004). Ordered valuation algebras: a generic framework for approximating inference. *International Journal of Approximate Reasoning*, *37*(1), 1–41.

Howard, R. A., & Matheson, J. E. (1984). Influence diagrams. In *Readings on the Principles and Applications of Decision Analysis*, pp. 721–762. Strategic Decisions Group.

Jensen, F. V., & Nielsen, T. D. (2007). *Bayesian Networks and Decision Graphs* (2nd edition). Information Science and Statistics. Springer.

Kohlas, J. (2003). *Information Algebras: Generic Structures for Inference.* Springer-Verlag, New York, USA.

Koller, D., & Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques*. MIT Press.

Lauritzen, S. L., & Nilsson, D. (2001). Representing and solving decision problems with limited information. *Management Science*, *47*, 1235–1251.

Mauá, D. D., & de Campos, C. P. (2011). Solving decision problems with limited information. In *Advances in Neural Information Processing Systems 24*, pp. 603–611.

Mauá, D. D., de Campos, C. P., & Zaffalon, M. (2011). Solving limited memory influence diagrams. *ArXiv:1109.1754v2 [cs.AI]*.

Park, J. D., & Darwiche, A. (2004). Complexity results and approximation strategies for MAP explanations. *Journal of Artificial Intelligence Research*, *21*, 101–133.

Poupart, P., & Boutilier, C. (2003). Bounded finite state controllers. In *Advances in Neural Information Processing Systems 16 (NIPS)*.

Pralet, C., Verfaillie, G., & Schiex, T. (2007). An algebraic graphical model for decision with uncertainties, feasibilities, and utilities. *Journal of Artificial Intelligence Research*, *29*, 421–489.

Qi, R., & Poole, D. (1995). A new method for influence diagram evaluation. *Computational Intelligence*, *11*, 498–528.

Shachter, R. D., & Peot, M. A. (1992). Decision making using probabilistic inference methods. In *Proceedings of the 8th Conference on Uncertainty in Artificial Intelligence*, pp. 276–283.

Shenoy, P., & Shafer, G. (1990). Axioms for probability and belief-function propagation. In *Proceedings of the 4th Conference on Uncertainty in Artificial Intelligence*, pp. 169–198.

Zhang, N. L., Qi, R., & Poole, D. (1994). A computational theory of decision networks. *International Journal of Approximate Reasoning*, *11*(2), 83–158.