A New Bounding Scheme for Influence Diagrams

Radu Marinescu[1], **Junkyu Lee**[2], **Rina Dechter**[2]
[1] IBM Research Europe
[2] University of California Irvine
radu.marinescu@ie.ibm.com, junkyul@uci.edu, dechter@ics.uci.edu

Abstract

Influence diagrams provide a modeling and inference framework for sequential decision problems, representing the probabilistic knowledge by a Bayesian network and the preferences of an agent by utility functions over the random variables and decision variables. Computing the maximum expected utility (MEU) and the optimizing policy is exponential in the constrained induced width and therefore is notoriously difficult for larger models. In this paper, we develop a new bounding scheme for MEU that applies partitioning based approximations on top of the decomposition scheme called a multi-operator cluster DAG for influence diagrams that is more sensitive to the underlying structure of the model than the classical join-tree decomposition of influence diagrams. Our bounding scheme utilizes a cost-shifting mechanism to tighten the bound further. We demonstrate the effectiveness of the proposed scheme on various hard benchmarks.

Introduction

Influence diagrams (IDs) (Howard and Matheson 1984) are a powerful formalism for reasoning with sequential decision making problems under uncertainty. They involve both *chance* or random variables, where the outcome is determined randomly based on the values assigned to the other variables, and *decision* variables, which the decision maker can chose the value of, based on observations of some other variables. Uncertainty is represented (like in Bayesian networks) by a collection of conditional probability distributions, one for each chance variable. The overall value of an outcome is represented as the sum of a collection of utility functions. Solving an ID is finding the maximum expected utility (MEU) and the corresponding optimal policy for each decision, subject to the history of observations and previous decisions.

Computing the MEU has long been recognized as one of the most difficult inference task over graphical models. Therefore, a significant research effort has been dedicated over the past years to developing efficient bounding schemes for MEU. The most common approaches use either information relaxation techniques that reorder the chance and decision variables to form a simpler model that yields an upper bound on the original MEU (Nilson and Holhe 2011) or reformulate the MEU into a Marginal MAP (MMAP) query on a related graphical model and apply standard approximation schemes for MMAP (Liu and Ihler 2012). More recently, a class of decomposition bounds has emerged as a powerful approximation scheme for MEU. These methods apply partitioning schemes such as mini-buckets (Dechter and Rish 2003) or weighted mini-buckets (Liu and Ihler 2011) to a join-tree or join-graph decomposition of the ID and employ various cost-shifting mechanisms to tighten the bounds further in an anytime manner (Lee, Ihler, and Dechter 2018; Lee et al. 2019). They were developed within the framework of valuation algebras (Shenoy 1992; Maua, de Campos, and Zaffalon 2012; Moral 2018) that define combination and marginalization operators over pairs of probability and utility functions called potentials (Jensen, Jensen, and Dittmer 1994).

Contribution: In this paper, we take the above bounding ideas and apply them on top of a more effective decomposition scheme called a multi-operator cluster DAG (or MCDAG) decomposition for influence diagrams (Pralet, Schiex, and Verfaillie 2006, 2009) instead of the traditional tree-decomposition (Jensen, Jensen, and Dittmer 1994). MCDAGs were introduced as an alternative to the classical Shenoy-Shafer architecture (Shenoy and Shafer 1990) for influence diagrams in order to avoid the manipulation of probability-utility potentials and to exploit further the underlying structure encoded by the functions of the model. More specifically, given an influence diagram and its MCDAG decomposition we develop a mini-bucket approximation to the messages passed along the MCDAG and show how to tighten the generated bounds further using weighted elimination and cost-shifting techniques. To the best of our knowledge this is the first time that such partitioning-based approximations are used together with MCDAG decompositions. Experimental results on difficult benchmarks show that in many cases the proposed approach outperforms the current best bounding schemes for influence diagrams by a large margin in terms of the quality of the bounds produced.

Background

Influence Diagrams

An *influence diagram* (ID) (Howard and Matheson 1984) is defined by a tuple $\mathcal{I} = \langle \mathbf{X}, \mathbf{D}, \mathbf{P}, \mathbf{U} \rangle$, where $\mathbf{X} = \{X_1, \ldots, X_n\}$ is a set of *chance variables* which specify the uncertain decision environment and $\mathbf{D} = \{D_1, \ldots, D_m\}$ is a set of *decision variables* which specify the possible decisions to be made in the domain. The chance variables are further divided into *observable* meaning they will be observed during execution, or *unobservable*. As in Bayesian networks (Pearl 1988), each chance variable $X_i \in \mathbf{X}$ is associated with a *conditional probability table* (CPT) $P_i = P(X_i|pa(X_i))$, where $P_i \in \mathbf{P}$ and $pa(X_i) \subseteq \mathbf{X} \cup \mathbf{D} \setminus \{X_i\}$. Each decision variable $D_k \in \mathbf{D}$ has a parent set $pa(D_k) \subseteq \mathbf{X} \cup \mathbf{D} \setminus \{D_k\}$, denoting the variables whose values will be known at the time of the decision and may affect directly the decision. *Non-forgetting* is typically assumed for an ID, meaning that a decision node and its parents are parents to all subsequent decisions. The *utility functions* $\mathbf{U} = \{U_1, \ldots, U_r\}$ are defined over subsets of variables $Q = \{Q_1, \ldots, Q_r\}$, $Q_i \subseteq \mathbf{X} \cup \mathbf{D}$, called *scopes*. They are assumed to be additive, thus defining a global utility function $\mathcal{U} = \sum_{j=1}^{r} U_j$ that captures the agent's preferences.

The decision variables in an ID are typically assumed to be temporally ordered (aka *regularity*). Let $D_1, D_2, ..., D_m$ be the order in which the decisions are to be made. The chance variables can be partitioned into a collection of disjoint sets $\mathbf{I}_0, \mathbf{I}_1, \ldots, \mathbf{I}_m$. For each k, where $0 < k < m$, $\mathbf{I}_k$ is the set of chance variables that are observed between D_k and D_{k+1}. $\mathbf{I}_0$ is the set of initial evidence variables that are observed before D_1. $\mathbf{I}_m$ is the set of chance variables left unobserved when the last decision D_m is made. This induces a strict partial order $\prec$ over $\mathbf{X} \cup \mathbf{D}$, given by: $\mathbf{I}_0 \prec D_1 \prec \mathbf{I}_1 \prec \cdots \prec D_m \prec \mathbf{I}_m$ (Jensen, Jensen, and Dittmer 1994).

A *policy* for an ID is a list of decision rules $\Delta = (\delta_1, \ldots, \delta_m)$ consisting of one rule for each decision variable. A *decision rule* for the decision $D_k \in \mathbf{D}$ is a mapping $\delta_k : \Omega_{pa(D_k)} \rightarrow \Omega_{D_k}$, where for a set $S \subseteq \mathbf{X} \cup \mathbf{D}$, Ω_S is the Cartesian product of the individual domains of the variables in S. Given a utility function $\mathcal{U}$, a policy Δ yields a utility function $[\mathcal{U}]_\Delta : \Omega_\mathbf{X} \rightarrow \mathbb{R}$ that involves no decision variables, by assigning their values using Δ. The expected utility given policy Δ is $EU_\Delta = \sum_\mathbf{X} [(\prod_{i=1}^{n} P_i \times \sum_{j=1}^{r} U_j)]_\Delta$. Solving an ID means finding an *optimal policy* that maximizes the expected utility, i.e., to find $\mathrm{argmax}_\Delta EU_\Delta$. The maximum expected utility (MEU) can be shown to be equal to:

$$\sum_{\mathbf{I}_0} \max_{D_1} \cdots \sum_{\mathbf{I}_{m-1}} \max_{D_m} \sum_{\mathbf{I}_m} \left(\prod_{i=1}^{n} P_i \times \sum_{j=1}^{r} U_j \right) \qquad (1)$$

Variable Elimination Several exact methods have been proposed over the past decades for solving IDs using local computations (Shachter 1986; Tatman and Shachter 1990; Shenoy 1992; Jensen, Jensen, and Dittmer 1994; Dechter 2000b). These methods adapted classical *variable elimination* (VE) techniques, which compute a type of marginalization over a combination of local functions, in order to handle the multiple types of information involved in influence diagrams. The alternation of $\sum$ and $\max$ marginalizations, which do not commute in general, prevents from eliminating variables in any ordering. Therefore, the computation dictated by Eq. 1 must be performed along a *constrained elimination ordering* that respects $\prec$, namely the reverse of the elimination ordering is some extension of $\prec$ to a total order (Jensen, Jensen, and Dittmer 1994; Dechter 2000b). The complexity of VE is thus time and space exponential in the *constrained induced width* (i.e., the induced width of the constrained elimination order) (Dechter 2000b).

Example 1. *Figure 1 shows a simple influence diagram with 3 decisions (D_0, D_2, D_4) and 2 chance variables (C_1, C_3). There are 2 probabilistic functions ($P(C_1|C_3)$, $P(C_3)$) and 4 utility functions ($u_1(D_0)$, $u_2(D_0, C_1, D_4)$, $u_3(D_2, C_3)$, $u_4(D_2, D_4)$), respectively. It is easy to see that the maximum expected utility can be found by solving:* $\max_{D_0} \sum_{C_1} \max_{D_2} \sum_{C_3} \max_{D_4} P(C_1|C_3) \cdot P(C_3) \cdot (u_1(D_0) + u_2(D_0, C_1, D_4) + u_3(D_2, C_3) + u_4(D_2, D_4))$.

Existing Bounding Schemes for Influence Diagrams

We next overview briefly several of the most effective schemes for bounding the MEU.

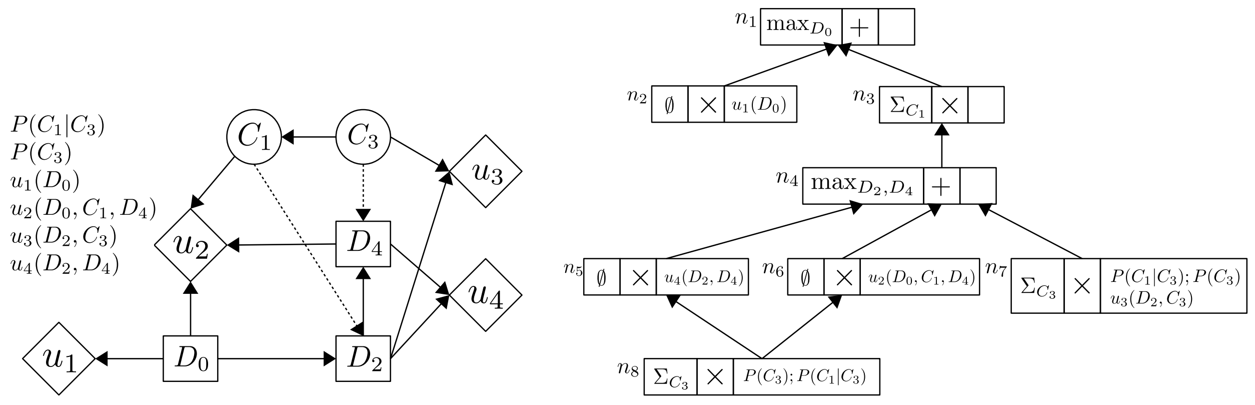
Mini-Bucket Elimination (MBE-ID) (Dechter 2000a) extends the original partitioning scheme (Dechter and Rish 2003) to upper bound the MEU. The scheme avoids the spaces and time complexity of exact variable elimination by partitioning large buckets into smaller subsets, called *mini-buckets*, each containing at most i (called i-bound) distinct variables. The mini-buckets are processed separately, either by summation or maximization depending on whether the bucket variable is a chance or a decision variable.

Join-Graph Decomposition (JGD-ID) (Lee, Ihler, and Dechter 2018) extends the generalized dual decomposition bounds (GDD) originally developed for Marginal MAP (Ping, Liu, and Ihler 2015) to the MEU task using the framework of *valuation algebras* (Shenoy 1992; Maua, de Campos, and Zaffalon 2012; Moral 2018). This results in a class of decomposition upper bounds that can be optimized through iterative cost-shifting updates. However, unlike GDD for MMAP, these bounds are non-convex and often difficult or slow to optimize, particularly for large clique sizes.

Weighted Mini-Buckets (WMB-ID) (Lee et al. 2019) extends the weighted mini-bucket scheme (Liu and Ihler 2011; Ihler et al. 2012; Marinescu, Dechter, and Ihler 2014) to the MEU task. Given a constrained elimination order, WMB-ID generates bounds for each variable conditioned on past histories, observations and decisions. It also applies a stage-wise cost-shifting procedure to minimize the bound locally during each approximate elimination step, based on optimizing a simplified upper bound on the unprocessed remainder of the model. The algorithm was shown to produce the tightest bounds compared to existing methods (Lee et al. 2019).

Multi-Operator Cluster DAG Decompositions

The *Multi-operator Cluster DAG* (MCDAG) decomposition (Pralet, Schiex, and Verfaillie 2006) was introduced as an alternative to the strong join-tree decomposition for influence diagrams (Jensen, Jensen, and Dittmer 1994). MCDAGs may often have smaller induced widths than strong join-trees which in turn may yield exponential gains for variable elimination algorithms. For a given ID $\mathcal{I} = \langle \mathbf{X}, \mathbf{D}, \mathbf{P}, \mathbf{U} \rangle$, the MCDAG reorganizes the computation defined by Eq. 1 into a DAG of *computation nodes* that exploits reordering freedom in the elimination order as well as the normalization conditions on conditional probability distributions.



$P(C_1|C_3)$
$P(C_3)$
$u_1(D_0)$
$u_2(D_0, C_1, D_4)$
$u_3(D_2, C_3)$
$u_4(D_2, D_4)$
C_1
C_3
u_3
u_2
D_4
u_4
u_1
D_0
D_2
n_1 $\max_{D_0}$ $+$
n_2 $\emptyset$ $\times$ $u_1(D_0)$
n_3 Σ_{C_1} $\times$
n_4 $\max_{D_2,D_4}$ $+$
n_5 $\emptyset$ $\times$ $u_4(D_2, D_4)$
n_6 $\emptyset$ $\times$ $u_2(D_0, C_1, D_4)$
n_7 Σ_{C_3} $\times$ $P(C_1|C_3); P(C_3)$ $u_3(D_2, C_3)$
n_8 Σ_{C_3} $\times$ $P(C_3); P(C_1|C_3)$

(a) Influence diagram (b) MCDAG decomposition

Figure 1: A simple influence diagram and its MCDAG decomposition.

DEFINITION **1** (computation node)**.** *An* atomic computation node n *is a scoped function* $f \in \mathbf{P} \cup \mathbf{U}$. *The value of* n *is* $val(n) = f$ *and its scope is* $sc(n) = sc(f)$. *A* computation node *is either an atomic computation node or a triple* $(Sov, \circledast, N)$*, where* Sov *is a sequence of operator-variable pairs* op_X *where* $op \in \{\sum, \max\}$*,* $\circledast \in \{+, \times\}$ *is an associative and commutative operator with an identity, and* N *is a set of computation nodes. The value of* n *is given by* $val(n) = Sov(\circledast_{n' \in N} val(n'))$ *and its scope is given by* $sc(n) = \bigcup_{n' \in N} sc(n') \setminus \{X | op_X \in Sov\}$*.*

A computation node $(Sov, \circledast, N)$ defines a sequence of marginalizations on a combination of computation nodes with a specific operator. Clearly, given $\mathcal{I} = \langle \mathbf{X}, \mathbf{D}, \mathbf{P}, \mathbf{U} \rangle$, the value of Eq. 1 is equal to the value of the computation node $n_0 = (Sov_0, +, \{(\emptyset, \times, \mathbf{P} \cup \{u_i\}), u_i \in \mathbf{U}\})$, where $Sov_0 = \sum_{\mathbf{I}_0} \max_{D_1} \dots \sum_{\mathbf{I}_{m-1}} \max_{D_m} \sum_{\mathbf{I}_m}$ is the initial sequence of operator-variable pairs defined by $\mathcal{I}$.

Building MCDAGs The basic idea is to refine the root computation node n_0 by applying three types of rewriting rules to the elimination sequence defined by Sov_0, namely: (1) decomposition rules which decompose the structure using a duplication mechanism, (2) recomposition rules which reveal freedoms in the elimination order, and (3) simplification rules which remove useless computations by using normalization conditions. For sum-marginalization, we use a decomposition rule $D_{\sum}$, a recomposition rule $R_{\sum}$, and two simplification rules $S^1_{\sum}$ and $S^2_{\sum}$, respectively. For max-marginalization there is only a decomposition rule $D_{\max}$ and a recomposition rule $R_{\max}$. These rules are applied successively from right to left to each operator-variable pair op_X in Sov_0 as follows. If $op_X = \sum_X$, then rule $D_{\sum}$ is applied once followed by R_Σ on each of the newly created computation nodes. If R_Σ is applied then simplification rules S^1_Σ and then S^2_Σ are applied until they cannot be applied anymore. If $op_X = \max_X$, then $D_{\max}$ is applied once followed by $R_{\max}$. Identical computation nodes can be easily identified and merged to reuse identical computations. Applying the rules in the order described converges to a unique DAG of mono-operator computation nodes looking like $(\max_S, +, N)$, $(\sum_S, \times, N)$ and $(\emptyset, \times, N)$, respectively (see also the supplementary material).

DEFINITION **2** (MCDAG)**.** *A* multi-operator cluster DAG (MCDAG) *is a DAG where each vertex* c *(called a* cluster*) is labeled with four elements: a set of variables* $V(c)$*, a set of scoped functions* $\Psi(c)$ *taking values in a set* E*, a set of child clusters* $ch(c)$ *and a pair of operators* $(\oplus, \otimes)$ *on* E *such that* $(\oplus, \otimes, E)$ *is a commutative semiring (where* $\oplus \in \{\sum, \max\}$ *and* $\otimes \in \{\times, +\}$*).*

Variable Elimination Given an MCDAG $\mathcal{M}$, the value of a cluster $n \in \mathcal{M}$ is given by $val(n) = \oplus_{V(n) \setminus V(pa(n))}((\otimes_{\psi \in \Psi(n)} \psi) \otimes (\otimes_{c \in ch(n)} val(n)))$. The value of $\mathcal{M}$ which is equivalent to Eq. 1 is the value of its root cluster and it can be computed by variable elimination (VE) (Pralet, Schiex, and Verfaillie 2006). Specifically, the clusters in $\mathcal{M}$ send messages to their parents. Whenever a cluster n has received all messages $val(v)$ from all its children, n can compute its value $val(n)$ and send it to its parents. Therefore, messages go from leaves to the root, and the value computed by the root cluster is the MEU. The complexity of VE is time and space exponential in the induced width s^* of $\mathcal{M}$ (Pralet, Schiex, and Verfaillie 2006). The induced width of $\mathcal{M}$ is defined by $s^* = \max_{n \in \mathcal{R}} w^*(n)$ where $\mathcal{R}$ is the set of nodes in $\mathcal{M}$ looking like $n = (\max_S, +, N)$ or $n = (\sum_S, \times, N)$ and $w^*(n)$ is the induced width of the graph $G(n) = (\mathcal{V}, \mathcal{E})$ for the elimination of the variables in S (Pralet, Schiex, and Verfaillie 2006). The vertices in $\mathcal{V}$ correspond to variables in $sc(N) = \bigcup_{n' \in N} sc(n')$ and an edge in $\mathcal{E}$ connects two vertices whose variables appear in a scope $sc(n')$, $n' \in N$.

Example 2. *Figure 1(b) shows the MCDAG of the ID from Figure 1. Algorithm VE starts with cluster* n_8 *sending message* $val(n_8) = \sum_{C_3} P(C_3) \cdot P(C_1|C_3) = \lambda_8(C_1)$ *to its parents* n_5 *and* n_6 *which subsequently compute their values as* $val(n_5) = \lambda_8(C_1) \cdot u_4(D_2, D_4) = \lambda_5(C_1, D_2, D_4)$ *and* $val(n_6) = \lambda_8(C_1) \cdot u_2(D_0, C_1, D_4) = \lambda_6(D_0, C_1, D_4)$*, respectively. Next, cluster* n_7 *sends message* $val(n_7) = \sum_{C_3} P(C_3) \cdot P(C_1|C_3) \cdot u_3(D_2, C_3) =$

$\lambda_7(C_1, D_2)$ to its parent, while cluster n_4 computes its value as $val(n_4) = \max_{D_2} \max_{D_4} \lambda_5(C_1, D_2, D_4) + \lambda_6(D_0, C_1, D_4) + \lambda_7(C_1, D_2) = \lambda_4(D_0, C_1)$. Subsequently, we have $val(n_3) = \sum_{C_1} \lambda_4(D_0, C_1) = \lambda_3(D_0)$, $val(n_2) = u_1(D_0)$ and $val(n_1) = \max_{D_0} u_1(D_0) + \lambda_3(D_0)$. The latter is the MEU. The induced width of the MCDAG is 3.

A Weighted Mini-Bucket Approximation for MCDAG Decompositions

In many practical situations however, the induced width of the MCDAG may still be too large for variable elimination to be effective. Therefore, we develop next a new scheme that applies a partitioning-based approximation to the messages propagated in the MCDAG together with cost-shifting to generate an improved upper bound on the MEU.

Approximation of Cluster-to-Cluster Messages

We will first illustrate the scheme using several simple examples and then present the general algorithm.

Consider again the MCDAG from Figure 1(b) and assume that due to memory restrictions we can process at most 2 variables when computing a new cluster-to-cluster message. Looking for example at the empty cluster n_5, we can no longer multiply the functions $\lambda_8(C_1)$ – the message received from its child n_8 and $u_4(D_2, D_4)$. Therefore, we must send them separately, as a compound message $\{\lambda_8(C_1), u_2(D_2, D_4)\}$, to its parent n_4 while ensuring that they will be combined by multiplication. Similarly, at cluster n_7, we avoid processing more than 2 variables by pushing the summation inside multiplication and therefore send to n_4 the compound message $\{\lambda_7^1(C_1), \lambda_7^2(D_2)\}$ whose components are to be multiplied together, where $\lambda_7^1(C_1) = \sum_{C_3} P(C_3) \cdot P(C_1|C_3)$ and $\lambda_7^2(D_2) = \sum_{C_3} u_3(D_2, C_3)$.

On the other hand, cluster n_4 processes the messages received from its children by $\max_{D_2} \max_{D_4}(\lambda_8(C_1) \cdot u_4(D_2, D_4) + \lambda_8(C_1) \cdot u_2(D_0, C_1, D_4) + \lambda_7^1(C_1) \cdot \lambda_7^2(D_2))$. In this case, maximization is pushed inside summation yielding the following approximation: $val(n_4) \leq \max_{D_2} \lambda_7^1(C_1) \cdot \lambda_7^2(D_2) + (\max_{D_4} \lambda_8(C_1) \cdot u_4(D_2, D_4) + \max_{D_4} \lambda_8(C_1) \cdot u_2(D_0, C_1, D_4)) = \max_{D_2} \lambda_7^1(C_1) \cdot \lambda_7^2(D_2) + \lambda_8(C_1) \cdot \lambda_4^1(D_2) + \lambda_8(C_1) \cdot \lambda_4^2(D_0, C_1) \leq \lambda_8(C_1) \cdot \lambda_4^2(D_0, C_1) + \max_{D_2} \lambda_7^1(C_1) \cdot \lambda_7^2(D_2) + \max_{D_2} \lambda_8(C_1) \cdot \lambda_4^1(D_2) = \lambda_8(C_1) \cdot \lambda_4^2(D_0, C_1) + \alpha \cdot \lambda_7^1(C_1) + \beta \cdot \lambda_8(C_1)$, where $\alpha = \max_{D_2} \lambda_7^2(D_2)$ and $\beta = \max_{D_2} \lambda_4^1(D_2)$, respectively. Subsequently, n_4 sends to its parent a compound message $\{\lambda_8(C_1) \cdot \lambda_4^2(D_0, C_1), \alpha \cdot \lambda_7^1(C_1), \beta \cdot \lambda_8(C_1)\}$ ensuring that its components are combined by summation.

Therefore, the basic idea behind our approach is to approximate the larger cluster-to-cluster messages by sets of smaller functions called *compound messages* while ensuring that their components are combined in the corresponding clusters either by multiplication or by summation. It is important to emphasize that unlike Marginal MAP where maximization is followed by summation and functions are combined only by multiplication, in the case of MCDAGs the sum clusters alternate with max clusters. Furthermore, these clusters combine their corresponding functions in different ways, namely sum clusters use multiplication while max clusters use summation. Thus, if the components of the compound messages are not combined appropriately then the scheme is not guaranteed to correctly upper bound the MEU.

We define formally the two types of compound messages that will be propagated between the clusters of an MCDAG.

DEFINITION 3 (π-message, σ-message). *A π-message is a compound message defined by a set of functions $\{f_1, \dots, f_k\}$ that must be combined by multiplication, i.e., $\pi = \prod_{i=1}^k f_i$. A σ-message is a compound message defined by a set of π-messages $\{\pi_1, \dots, \pi_l\}$ that must be combined by summation, i.e., $\sigma = \sum_{j=1}^l \pi_j$. Their scopes are defined by $vars(\pi) = \bigcup_{i=1}^k vars(f_i)$ and $vars(\sigma) = \bigcup_{j=1}^l vars(\pi_j)$.*

We next show in detail how to approximate the values computed by clusters of the form $(\sum_S, \times, N)$ and $(\max_S, +, N)$ (also called SUM and MAX clusters). For simplicity and without loss of generality we assume in the subsequent derivations that S contains a single variable, i.e., $S = \{X\}$.

Processing a SUM Cluster: Let $n = (\sum_X, \times, N)$ be a SUM cluster where $N = \{f_1, f_2, \dots, f_n, \pi_1, \pi_2, \dots, \pi_m, \sigma_1, \sigma_2, \dots, \sigma_p\}$, that is N contains functions, π- and σ-messages, respectively. In this case, the cluster function is $\mathcal{F} = \prod_{i=1}^n f_i \cdot \prod_{j=1}^m \pi_j \cdot \prod_{k=1}^p \sigma_k$. Since $\times$ distributes over $+$, we can write $\prod_{k=1}^p \sigma_k = (\pi_1^1 + \cdots + \pi_{l_1}^1) \times \cdots \times (\pi_1^p + \cdots + \pi_{l_p}^p) = (\pi_1^1 \cdot \pi_1^2 \cdots \pi_1^p + \cdots + \pi_{l_1}^1 \cdot \pi_{l_2}^2 \cdots \pi_{l_p}^p)$ and subsequently rearrange $\mathcal{F}$ as a sum of products of functions, namely $\mathcal{F} = (\mathcal{P} \cdot \pi_1^1 \cdot \pi_1^2 \cdots \pi_1^p) + \cdots + (\mathcal{P} \cdot \pi_{l_1}^1 \cdot \pi_{l_2}^2 \cdots \pi_{l_p}^p)$, where $\mathcal{P} = \prod_{i=1}^n f_i \cdot \prod_{j=1}^m \pi_j$. Let $K = \prod_{k=1}^p l_k$ be the number of products in $\mathcal{F}$ and let F_t be the set of functions involved in the t-th product of $\mathcal{F}$. Given a parameter $\underline{i}$ (also called an i-bound), we can approximate $val(n)$ by applying a mini-bucket partitioning to each of the products in $\mathcal{F}$ separately, thus generating the following σ-message:

$$val(n) = \sum_X \mathcal{F} = \sum_X \sum_{t=1}^K \prod_{f \in F_t} f = \sum_{t=1}^K \sum_X \prod_{f \in F_t} f \leq \sum_{t=1}^K \prod_{q=1}^M \sum_X \prod_{f \in Q_q^t} f \quad (2)$$

where $\mathcal{Q}^t = \{Q_1^t, \dots, Q_M^t\}$ is a partitioning of F_t with M mini-buckets, where each mini-bucket $Q_q^t \in \mathcal{Q}^t$ contains at most $\underline{i}$ distinct variables. Clearly, if $n = (\sum_S, \times, N)$ is a leaf SUM cluster (e.g., cluster n_7 in Figure 1(b)) then N contains only functions and therefore Eq. 2 yields a π-message.

Example 3. *For illustration, let $n = (\sum_A, \times, N)$ be a SUM cluster where $N = \{f_1(A, B), \pi(A, B, C), \sigma(A, B, C)\}$, such that $\pi(A, B, C) = h_1(A, B) \cdot h_2(A, C)$ and $\sigma(A, B, C) = g_1(A, B) \cdot g_2(A, C) + g_3(A, B) \cdot g_4(A, C)$, respectively. Using distributivity and rearranging the terms we can write $\mathcal{F} = f_1(A, B) \cdot h_1(A, B) \cdot h_2(A, C) \cdot g_1(A, B) \cdot g_2(A, C) + f_1(A, B) \cdot h_2(A, C) \cdot g_3(A, B) \cdot g_4(A, C)$. For an i-bound of 2, we get the following approximation: $val(n) \leq (\sum_A f_1(A, B) \cdot h_1(A, B) \cdot g_1(A, B)) \cdot$*

Algorithm 1 MCDAG-MBE($\underline{i}$)

```
Require: ID $\mathcal{I} = (\mathbf{X}, \mathbf{D}, \mathbf{P}, \mathbf{U})$, $i$-bound $\underline{i}$
 1: Let $\mathcal{M}$ be the MCDAG of $\mathcal{I}$ ($C_1$ – root cluster of $\mathcal{M}$)
 2: for all clusters $C_j \in \mathcal{M}$ from leaves to the root do
 3:   if $C_j$ is a SUM cluster $(\sum_S, \times, N)$ then
 4:     for all variables $X \in S$ do
 5:       Let $N^{+x} = \{\phi | \phi \in N, X \in vars(\phi)\}$
 6:       $\sigma_X$ = APPROX-SUM($\underline{i}$, $(\sum_X, \times, N^{+x})$)
 7:       Update $N = (N \setminus N^{+x}) \cup \{\sigma_X\}$
 8:       if $X$ is last variable in $S$ then
 9:         Set cluster value $val(C_j) = \sigma_X$
10:   else if $C_j$ is a MAX cluster $(\max_S, +, N)$ then
11:     for all variables $X \in S$ do
12:       Let $N^{+x} = \{\phi | \phi \in N, X \in vars(\phi)\}$
13:       $\sigma_X$ = APPROX-MAX($\underline{i}$, $(\max_X, +, N^{+x})$)
14:       Update $N = (N \setminus N^{+x}) \cup \{\sigma_X\}$
15:       if $X$ is last variable in $S$ then
16:         Set cluster value $val(C_j) = \sigma_X$
17:   else
18:     $\sigma$ = APPROX-EMPTY($(\emptyset, \times, N)$)
19:     Set cluster value $val(C_j) = \sigma$
20:   Send message $val(C_j)$ to $C_j$'s parents in $\mathcal{M}$
21: return $val(C_1)$
```

$(\sum_A h_2(A,C) \cdot g_2(A,C)) + (\sum_A f_1(A,B) \cdot h_1(A,B) \cdot g_3(A,B)) \cdot (\sum_A \cdot h_2(A,C) \cdot g_4(A,C)) = \lambda_1(B) \cdot \lambda_2(C) + \lambda_3(B) \cdot \lambda_4(C)$. *Therefore, the σ-message is $\sigma = \{\pi_1, \pi_2\}$ where $\pi_1 = \{\lambda_1(B), \lambda_2(C)\}$ and $\pi_2 = \{\lambda_3(B), \lambda_4(C)\}$.*

Processing a MAX Cluster: Consider next a MAX cluster $n = (\max_X, +, N)$ with $N = \{f_1, \dots, f_n, \pi_1, \dots, \pi_m, \sigma_1, \dots, \sigma_p\}$ as before. In this case, we have that $\mathcal{F} = \sum_{i=1}^n f_i + \sum_{j=1}^m \pi_j + \sum_{k=1}^p \sigma_k = \sum_{i=1}^n f_i + \sum_{k=0}^p \sigma_k$, where $\sigma_0 = \sum_{j=1}^m \pi_j$. Given an i-bound $\underline{i}$, we can approximate $val(n)$ by applying a multi-stage partitioning that pushes max inside summation as well as multiplication, thus generating a σ-message as follows:

$$\begin{aligned} val(n) = \max_X \mathcal{F} &\le (\max_X \sum_{i=1}^n f_i) + (\max_X \sum_{k=0}^p \sigma_k) \\ &\le (\sum_{q=1}^M \max_X \sum_{f \in Q_q} f) + (\sum_{k=0}^p \max_X \sigma_k) \qquad (3) \\ &\le \sum_{q=1}^M \max_X \sum_{f \in Q_q} f + \sum_{k=0}^p \prod_{l=1}^{L_k} \max_X \prod_{f \in Q_{kl}} f \end{aligned}$$

where $\mathcal{Q} = \{Q_1, \dots Q_M\}$ is a mini-bucket partitioning of the functions $\{f_1, \dots, f_n\}$, and $\mathcal{Q}_k = \{Q_{k1}, \dots, Q_{kL_k}\}$ is a mini-bucket partitioning of the functions in σ_k, respectively.

Example 4. *Consider the MAX cluster $n = (\max_A, +, N)$ such that $N = \{f_1(A,B), f_2(A,C), \pi(A,B,C,D), \sigma(A,B,C,D)\}$, $\pi(A,B,C,D) = h_1(A,B) \cdot h_2(C,D)$ and $\sigma(A,B,C,D) = g_1(A,B) \cdot g_2(A,C) + g_3(A,B) \cdot g_4(C,D)$, respectively. First, we have $\mathcal{F} = f_1(A,B) + f_2(A,C) + h_1(A,B) \cdot h_2(C,D) + g_1(A,B) \cdot g_2(A,C) + g_3(A,B) \cdot g_4(C,D)$). Then, assuming a mini-bucket i-bound of 3, we can apply the multi-stage mini-bucket partitioning from Eq. 3 and generate the following approximation: $val(n) \le \max_A(f_1(A,B) + f_2(A,C)) + \max_A h_1(A,B) \cdot h_2(C,D) + \max_A g_1(A,B) \cdot g_2(A,C) + \max_A g_1(A,B) \cdot g_4(C,D) = \lambda_1^1(B,C) + \lambda_2^1(B) \cdot h_2(C,D) + \lambda_3^1(B,C) + \lambda_4^1(B) \cdot g_4(C,D)$, where $\lambda_1^1(B,C) = \max_A f_1(A,B) + f_2(A,C)$, $\lambda_2^1(B) = \max_A h_1(A,B)$, $\lambda_3^1(B,C) = \max_A g_1(A,B) \cdot g_2(A,C)$ and $g_4^1(B) = \max_A g_3(A,B)$, respectively.*

Processing an Empty Cluster: Let $n = (\emptyset, \times, N)$ be an empty cluster with N as before. In this case, we just use the distributivity property to create a σ-message $\sigma = (\mathcal{P} \cdot \pi_1^1 \cdot \pi_1^2 \cdots \pi_1^p) + \cdots + (\mathcal{P} \cdot \pi_{l_1}^1 \cdot \pi_{l_2}^2 \cdots \pi_{l_p}^p)$, where $\mathcal{P} = \prod_{i=1}^n f_i \cdot \prod_{j=1}^m$ and $\sigma_k = (\pi_1^k + \cdots + \pi_{l_k}^k)$, for $1 \le k \le p$.

The Mini-Bucket Approximation for MCDAGs

Algorithm 1 presents the mini-bucket approximation for MCDAGs, called MCDAG-MBE($\underline{i}$). Let $\mathcal{I} = \langle \mathbf{X}, \mathbf{D}, \mathbf{P}, \mathbf{U} \rangle$ be the input ID and $\mathcal{M}$ be an MCDAG decomposition. Given an i-bound $\underline{i}$, the mini-bucket algorithm processes each cluster $C_j \in \mathcal{M}$ in a bottom up manner, from leaves to the root, by a variable elimination procedure that computes new *compound messages* and sends them to C_j's parents in $\mathcal{M}$. Specifically, if C_j is a SUM cluster labeled $(\sum_S, \times, N)$ then N contains all the messages received from C_j's children and consists in general of input probability and utility functions, π- and σ-messages, respectively. The variables in S are eliminated one by one as follows. Let X be the current variable. We denote by N^{+x} the set of N's elements that mention X in their scope. Function APPROX-SUM implements Eq. 2 and eliminates X using the i-bound $\underline{i}$ to generate the σ-message σ_X that is added back to N. If X is the last variable in S then σ_X is sent to all of C_j's parents in $\mathcal{M}$. Alternatively, if C_j is a MAX cluster labeled $(\max_S, +, N)$ then all variables in S are eliminated one by one using function APPROX-MAX that implements Eq. 3 and generates the corresponding σ-messages. Furthermore, if C_j is an empty cluster, then function APPROX-EMPTY simply uses the distributivity property of $\times$ over $+$ to compute the corresponding σ-message that is subsequently passed to C_j's parents. Finally, after the root cluster C_1 is processed, its value $val(C_1)$ represents an upper bound on the exact MEU value of $\mathcal{I}$ and is returned.

THEOREM **1** (correctness, complexity)**.** *Given an influence diagram $\mathcal{I}$, algorithm MCDAG-MBE($\underline{i}$) is correct, namely it computes an upper bound on the MEU of $\mathcal{I}$. Its complexity is time and space exponential in the i-bound $\underline{i}$ only.*

Tightening the Upper Bound

Weighted mini-buckets (Liu and Ihler 2011; Ihler et al. 2012; Marinescu, Dechter, and Ihler 2014) improve the naïve mini-bucket bound by using Hölder's inequality as well as cost-shifting (or reparameterization) via moment-matching operations between mini-buckets. We show next how to extend these ideas to MCDAG-MBE($\underline{i}$) yielding a new scheme

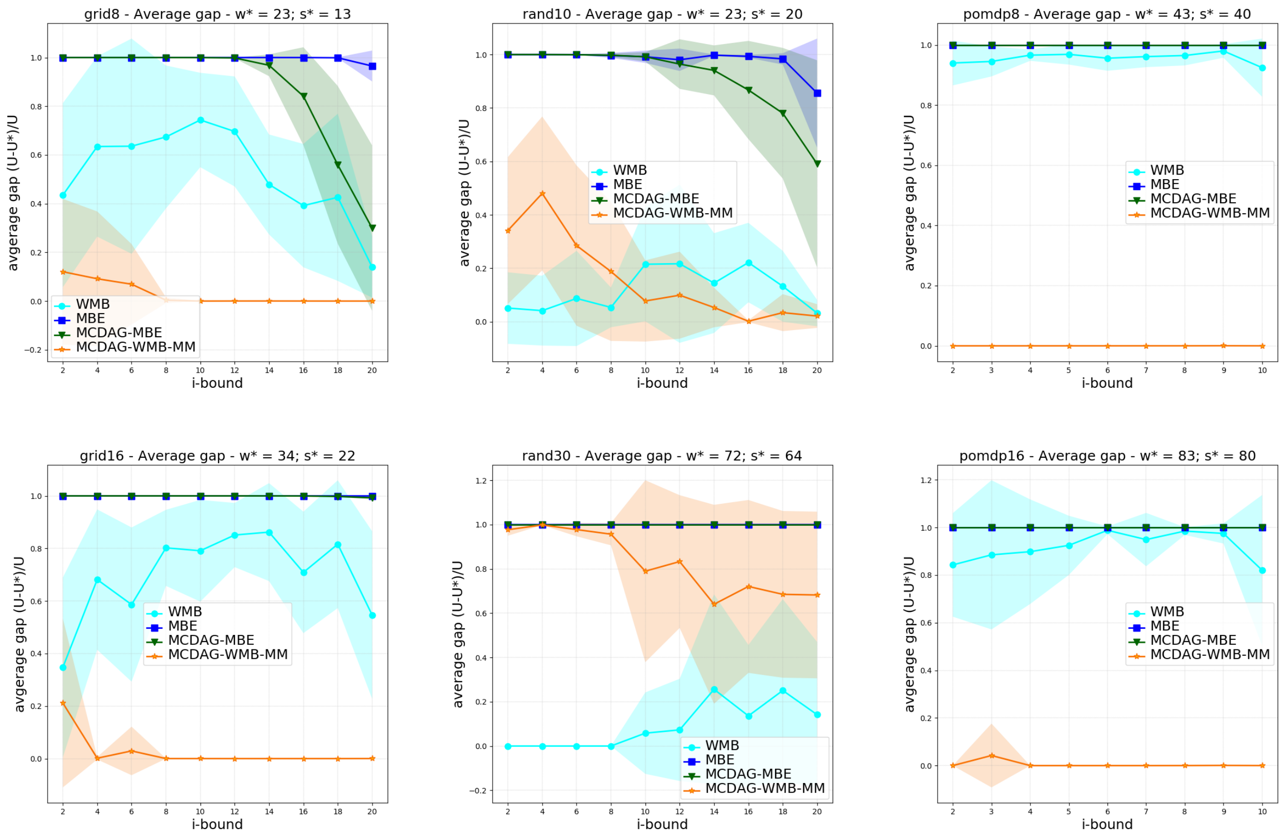


Figure 2: Average relative gap to the tightest upper bound for `grid`, `random` and `pomdp` benchmarks.

called MCDAG-WMB-MM($\underline{i}$) (weighted mini-buckets with moment-matching) that can produce improved upper bounds.

Cost-Shifting for SUM Clusters Let $n = (\sum_X, \times, N)$ be a SUM cluster. Recall from Eq. 2 that we rewrite $\mathcal{F}$, the combination of functions and messages in N, as a sum of products of functions, and subsequently approximate each product separately. Let $P_k = \prod_j f_j^k$ be the k-th product of functions in $\mathcal{F}$ and let $\mathcal{Q} = \{Q_{k1}, \ldots, Q_{kR}\}$ be its mini-bucket partitioning. Assuming that $\psi_{kr} = \prod_{f \in Q_{kr}} f$ is the function of the r-th mini-bucket Q_{kr}, we assign weight $w_{kr} > 0$ to each mini-bucket Q_{kr} such that $\sum_r w_{kr} = 1$ and perform a regular moment-matching reparameterization, namely $\psi_{kr} = \psi_{kr} \cdot \left(\frac{\mu}{\mu_r}\right)^{w_{kr}}$, where $\mu_r = \sum_{Y_r} (\psi_{kr})^{1/w_{kr}}$, $\mu = \prod_r (\mu_r)^{w_{kr}}$ and $Y_r = vars(Q_{kr}) \setminus \{X\}$, as is typically done in the case of sum-product elimination (Marinescu, Dechter, and Ihler 2014). After reparameterization, each mini-bucket is subsequently processed by weighted elimination to compute $\lambda_{kr} = (\sum_X (\psi_{kr})^{1/w_{kr}})^{w_{kr}}$ which is used to generate the final σ-message that upper bounds $val(n)$.

Cost-Shifting for MAX Clusters In this case, we perform a multi-stage mini-bucket partitioning that involves both max-sum and max-product eliminations (see Eq. 3) and therefore the cost-shifting scheme attempts to match the max-marginals across the mini-buckets of each of the partitionings. Specifically, for the max-product stage, let $P_k = \prod_j f_j$ be a product of functions and let $\mathcal{Q}_k = \{Q_{k1}, \ldots, Q_{kR}\}$ its mini-bucket partitioning where $\psi_{kr} = \prod_{f \in Q_{kr}} f$. We reparameterize ψ_{kr} as $\psi_{kr} = \psi_{kr} \cdot \left(\frac{\mu}{\mu_r}\right)$, where $\mu_r = \max_{Y_r} \psi_{kr}$ and $\mu = (\prod_r \mu_r)^{1/R}$ and $Y_r = vars(Q_{kr}) \setminus \{X\}$. Similarly, for the max-sum stage, the mini-bucket function is $\psi_{kr} = \sum_{\psi \in Q_{kr}}$ and we reparameterize ψ_{kr} by $\psi_{kr} = \psi_{kr} - \mu_r + \frac{1}{R}\mu$, where $\mu_r = \max_{Y_r} \psi_{kr}$ and $\mu = \sum_r \mu_r$, respectively.

Experiments

We evaluate empirically the performance of our bounding scheme on a variety of difficult benchmarks for IDs. All experiments were run on a 2.6GHz CPU with 64GB of RAM.

Benchmarks and Algorithms For our purpose, we considered three random problem domains: (1) `grid` which consists of random m-by-m grid IDs with d decisions and $m^2 - d$ chance variables, (2) `random` which consists of random IDs with c chance variables and d decisions, and (3) `pomdp` which consists of random POMDPs with s state

algorithm	instance	i=2	i=6	i=10	instance	i=2	i=10	i=18
MBE	**maze_a_d2**	1.23E+20	8.86E+02	4.86	**sys1_s=10_t=3**	2.09E+34	2.38E+18	7.70E+13
MCDAG-MBE	c=14,d=2	6.30E+20	1.14E+03	9.21	c=79,d=30	2.82E+24	2.88E+09	2.60E+06
WMB	w*=11	5813.91	5.37	0.35	w*=60	1.02E+10	8.37E+07	4.34E+06
MCDAG-WMB-MM	s*=10,k=9	16.13	1.12	**0.21**	s*=58,k=3	3.44E+07	8.79E+03	**4.31E+02**
MBE	**maze_a_d4**	6.96E+40	6.47E+06	2.59E+03	**sys1_s=10_t=4**	1.72E+46	3.01E+26	1.11E+19
MCDAG-MBE	c=26,d=4	2.92E+45	4.04E+09	3.42E+03	c=102,d=40	1.93E+35	3.79E+14	1.39E+10
WMB	w*=21	2.73E+08	2.30E+03	8.38	w*=80	6.59E+13	2.87E+11	3.04E+08
MCDAG-WMB-MM	s*=20,k=9	4362.69	11.23	**1.07**	s*=78,k=3	1.18E+11	2.68E+06	**3.98E+04**
MBE	**maze_a_d6**	6.92E+60	1.27E+14	6.27E+06	**sys1_s=10_t=5**	1.38E+58	5.43E+30	6.27E+22
MCDAG-MBE	c=38,d=6	1.35E+73	5.41E+18	7.32E+08	c=125,d=50	1.33E+46	3.18E+19	9.12E+13
WMB	w*=31	1.96E+13	2.79E+05	365.84	w*=100	1.80E+17	7.67E+13	4.34E+11
MCDAG-WMB-MM	s*=30,k=9	3.10E+05	241.10	**7.60**	s*=98,k=3	4.09E+14	1.36E+09	**1.46E+07**
MBE	**maze_a_d8**	2.14E+82	3.34E+19	8.28E+12	**sys1_s=10_t=6**	1.01E+70	4.72E+36	1.71E+26
MCDAG-MBE	c=50,d=8	1.66E+106	6.80E+28	1.93E+16	c=148,d=60	9.14E+56	8.80E+23	1.37E+17
WMB	w*=41	6.19E+17	1.03E+08	1.25E+04	w*=120	3.86E+20	7.16E+16	3.91E+13
MCDAG-WMB-MM	s*=40,k=9	7.94E+08	5.68E+03	**193.86**	s*=118,k=3	2.49E+18	1.04E+12	**2.23E+09**

Table 1: Results on the `maze` (left) and `sysadmin` (right) problem instances.

variables per time step and d time steps (or decisions). We generated problem instances for each domain as follows. For `grid`, we generated a 10-by-10 grid DAG ($m = 10$) and then randomly selected $d \in \{8, 12, 16\}$ variables to act as decisions. For `random`, we generated random DAGs with $c \in \{50, 100, 150\}$ chance variables and $d \in \{10, 20, 30\}$ decisions, respectively. For `pomdp`, we set the number of state variables s to 10 per stage (so that we had 4 observed and 6 hidden variables, respectively) and varied the number of decisions from 8 to 16, respectively. In all cases, the variables had 2 values in their domains and we generated d random binary utility functions such that each decision variable was included in the scope of at least one utility function. The utility values were randomly sampled between 0 and 1. We generated a total of 90 instances (30 instances per domain).

In addition, we also experimented with two realistic planning domains called `maze` and `sysadmin`, respectively. The `maze` problems are influence diagrams that model the action selection of an agent traversing a maze containing walls and open spaces (Horsch and Poole 1998). The `sysadmin` problems optimize the decisions of a system administrator maintaining a network of computers (Guestrin et al. 2003).

We evaluated algorithms MCDAG-MBE(i) and MCDAG-WMB-MM(i) and compared them with the standard mini-buckets MBE(i) (Dechter 2000a) and the recent weighted mini-buckets WMB(i) (Lee et al. 2019). The latter was shown to outperform the join-graph dual decomposition scheme JGD-ID(i) from (Lee, Ihler, and Dechter 2018). All algorithms are parameterized by the mini-bucket i-bound and use a *min-fill* based elimination ordering (Kjaerulff 1990).

Results In Figure 2 we plot the average relative gap $\rho = (U - U^*)/U$ with respect to the tightest upper bound (U^*) as a function of the mini-bucket i-bound for the `grid`, `random` and `pomdp` domains with {8,16}, {10,30} and {8,16} decisions, respectively. Each data point is an average over 10 random instances generated for that particular domain and the shaded areas represent ± 1 standard deviations from the mean. We also record the average constrained induced width (w^*) of the min-fill ordering and the average induced width (s^*) of the corresponding MCDAG decomposition. We see that on `grids` and `pomdp` problems MCDAG-WMB-MM(i) was able to compute the tightest upper bounds at all reported i-bounds which shows the benefit of exploiting the MCDAG instead of the traditional tree-decomposition. However, on `random` problems which contain a large number of decision variables, we see that the picture is reversed with WMB(i) producing tighter bounds than MCDAG-WMB-MM(i). In this case, the multi-stage relaxation defined by Eqs. 2 and 3 leads to poor quality bounds and the moment-matching reparameterization is not powerful enough to tighten them.

Table 1 shows the upper bounds obtained on `maze` and `sysadmin` problem instances for different values of the i-bound. For each instance we also report the number of chance variables (c), the number of decisions (d) as well as the maximum domain size (k). We see again that in both domains MCDAG-WMB-MM(i) produced the tightest bounds. In many cases these bounds were several orders of magnitude tighter than those computed by the other competitors. This demonstrates conclusively the benefit of partitioning based approximations and cost-shifting over MCDAGs instead of traditional strong join-tree decompositions for IDs.

Conclusion

The paper revisits the MCDAG decomposition for influence diagrams and proposes a new mini-bucket approximation scheme for bounding the MEU. MCDAGs are more sensitive to the underlying problem structure than join-trees often yielding smaller induced width decompositions which in turn may lead to partitionings that yield more accurate bounds. We also show how to tighten further the proposed mini-bucket bounds using a simple moment-matching reparameterization scheme. Our experiments on a variety of difficult benchmark problems demonstrate the effectiveness of the proposed bounds compared with existing state-of-the-art approximation schemes for influence diagrams.

Acknowledgements

We thank the reviewers for their valuable feedback. This work was supported in part by the NSF grant IIS-2008516.

References

Dechter, R. 2000a. An anytime approximation for optimizing policies under uncertainty. In *Workshop of Decision Theoretic Planning in (AIPS-2000).*

Dechter, R. 2000b. A new perspective on algorithms for optimizing policies under uncertainty. In *Artificial Intelligence Planning Systems (AIPS)*, 72–81.

Dechter, R.; and Rish, I. 2003. Mini-buckets: A general scheme of approximating inference. *Journal of ACM* 50(2): 107–153.

Guestrin, C.; Koller, D.; Parr, R.; and Venkataraman, S. 2003. Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research* 19(1): 399–468.

Horsch, M. C.; and Poole, D. 1998. An anytime algorithm for decision making under uncertainty. In *Uncertainty in Artificial Intelligence (UAI)*, 246–255.

Howard, R.; and Matheson, J. 1984. Influence diagrams. In *Readings on the Principles and Applications of Decision Analyis*, 721–762.

Ihler, A.; Flerova, N.; Dechter, R.; and Otten, L. 2012. Join-graph based cost-shifting schemes. In *Uncertainty in Artificial Intelligence (UAI)*, 397–406.

Jensen, F.; Jensen, V.; and Dittmer, S. 1994. From influence diagrams to junction trees. In *Uncertainty in Artificial Intelligence (UAI)*, 367–363.

Kjaerulff, U. 1990. Triangulation of graph-based algorithms giving small total space. *Technical Report, University of Aalborg, Denmark* .

Lee, J.; Ihler, A.; and Dechter, R. 2018. Join Graph Decomposition Bounds for Influence Diagrams. In *Uncertainty in Artificial Intelligece (UAI)*, 1053–1062.

Lee, J.; Marinescu, R.; Dechter, R.; and Ihler, A. 2019. A Weighted Mini-Bucket Bound for Solving Influence Diagrams. In *Uncertainty in Artificial Intelligece (UAI)*, 393–400.

Liu, Q.; and Ihler, A. 2011. Bounding the partition function using Holder's inequality. In *International Conference on Machine Learning (ICML)*, 849–856.

Liu, Q.; and Ihler, A. 2012. Belief propagation for structured decision making. In 523–532, ed., *Uncertainty in Artificial Intelligence (UAI).*

Marinescu, R.; Dechter, R.; and Ihler, A. 2014. AND/OR search for marginal MAP. In *Uncertainty in Artificial Intelligence (UAI)*, 563–572.

Maua, D.; de Campos, C.; and Zaffalon, M. 2012. Solving Limited Memory Influence Diagrams. *Journal of Artificial Intelligence Research (JAIR)* 44: 211–229.

Moral, S. 2018. Divergence measures and approximate algorithms for valuation based systems. In *Information Processing and Management of Uncertainty in Knowledge-Based Systems*, 591–602.

Nilson, D.; and Holhe, M. 2011. Computing bounds on expected utilties for optimal policies based on limited information. In *Dinar Research Report*.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. ISBN 0934613737.

Ping, W.; Liu, Q.; and Ihler, A. 2015. Decomposition bounds for Marginal MAP. In *Advances in Neural Information Processing Systems (NIPS)*, 3267—-3275.

Pralet, C.; Schiex, T.; and Verfaillie, G. 2006. From Influence Diagrams to Multi-Operator Cluster DAGs. In *Uncertainty in Artificial Intelligece (UAI)*, 393–400.

Pralet, C.; Schiex, T.; and Verfaillie, G. 2009. *Sequential Decision-Making Problems—Representation and Solution*. Wiley. ISBN 978-1-84821-174-2.

Shachter, R. 1986. Evaluating influence diagrams. *Operations Research* 34(6): 871–882.

Shenoy, P. 1992. Valuation-based systems for Bayesian decision analysis. *Operations Research* 40(1): 463–484.

Shenoy, P.; and Shafer, G. 1990. Axioms for probability and belief-function propagation. In *Uncertainty in Artificial Intelligence (UAI)*, 169–178.

Tatman, J.; and Shachter, R. 1990. Dynamic programming and influence diagrams. *IEEE Transactions on Systems, Man, and Cybernetics* 20(1): 365–379.