Computing bounds on expected utilities for optimal policies based on limited information

Dennis Nilsson[*] [†] Michael Höhle[†]

October 2001

1 Introduction

This paper is concerned with agents or decision makers that use 'limited memory' policies in the sense that decisions are based on only some of their previous observations. The study of such policies is motivated in a number of ways, as detailed below.

Limited memory policies are not only simpler; they may also be optimal: In many cases previous events can be ignored because they are irrelevant given all other available information. That is indeed the case in the class of Markovian environments, in which the optimal choice of a particular decision is completely determined by the latest observation leaving all other observations irrelevant. Similarly, in influence diagrams it is well-known that optimal decisions can be made by only considering the socalled *requisite information* observed up until the decisions are made (Shachter 1998; Nielsen and Jensen 1999; Nilsson and Lauritzen 2000).

Also, 'full memory policies' can be computationally expensive. This holds, for example, for partially observable Markov decision problems (POMDPs). Here, optimal policies are based on the whole history, i.e. all previous observations and all previous actions. Even though exact algorithms have been developed for solving POMDPs, general POMDPs are intractable. Similarly, the performance of influence diagrams (IDs) in practice has been

[*]Department of Mathematical Sciences, Aalborg University, Fredrik Bajers Vej 7, DK-9220 Aalborg, Denmark.

[†]Department of Animal Science and Health, Royal Veterinary and Agricultural University, Grønnegårdsvej 3, 1870 Frb. C., Denmark.

disappointing because the existing algorithms compute policies from all requisite information; a procedure that is bound to fail for most real world applications.

Partly motivated by the insufficiency of influence diagrams, Lauritzen and Nilsson (2001) extended the notion of influence diagrams to so-called LImited Memory Influence Diagrams or simply LIMIDs. In contrast with traditional IDs, LIMIDs can represent limited memory policies and provide tools for computing them. However, when leaving the territory of 'full memory' policies, finding optimal policies may be intractable, and the LIMID algorithm presented is exact in some cases, and only approximate in other cases. Evidently, it is important to identify how far an approximate solution in LIMIDs is from being fully optimal, and consequently precise upper bounds on the expected utility are wanted.

This paper is intended to make two contributions. Firstly, we present different heuristic methods for computing near-optimal strategies. To estimate the quality of such near-optimal strategies it is important to have precise upper bounds for the expected utility of optimal strategies. Secondly, we develop algorithms for computing such upper bounds. Both contributions utilize the LIMID representation and the LIMID evaluation algorithm to compute the bounds.

Throughout we will use a simple example of a robot navigating in a small area. Whereas such decision problems are typically represented as POMDPs, our model uses the LIMID representation, which enable to model the situation where the robot only remembers its most recent observations and actions.

2 Limited Memory Influence Diagrams

The present section gives a basic description of LIMIDs as developed in Lauritzen and Nilsson (2001). For proofs, the reader is referred to this source.

2.1 LIMID representation

A LIMID is a directed acyclic graph (DAG) with no cycles. There are three types of nodes in the graph: *Chance nodes*, shown as circles, represent random variables. *Decision nodes*, shown as boxes, represent decision variables. Finally, *value nodes*, shown as diamonds, represent (local) utility functions.

Arcs in a LIMID have a different meaning depending on which node they go into. Arcs into chance nodes denote probabilistic (or functional) dependence. Associated with each chance node r is a probability distribution

p_r of r given its parents. The arcs into value nodes are functional. Associated with each value node u, there is a utility function U_u that depends on the parents of u. Arcs into a decision node are informational since they originate from nodes whose values are known for the decision maker when the decision is to be made. In contrast with traditional IDs, LIMIDs have no assumption of 'no forgetting'.

Example 1 (A Robot example) To illustrate the LIMID representation applied to a POMDP, we will use the simple Robot example of Kaelbling *et al.* (1998). Figure 1 shows the environment of the agent. It has four states, one of which is the goal state, indicated by the star. There are two possible observations: one observation is made if the agent is in the non-goal states 1,2, and 4 and another observation is made if the agent is in the goal state.

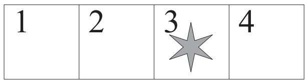


Figure 1: Agent environment of the Robot example; empty squares are indistinguishable to the agent, whereas the star indicates the distinguishable goal state.

There are two possible actions: *east* and *west*. These actions succeed with probability 0.9, and when they fail, the movement is in the opposite direction. If no movement is possible in a particular direction, the agent remains in the same location. At the beginning of the game, the agent is randomly placed in one of the non-goal states. Each time the agent is present in the goal state a reward of one is given. Thus, the agent's task has two phases. To find the goal state as quickly as possible, and — once it is found — to establish a procedure which visits the goal state as many times as possible.

To represent the above game by a LIMID we use four nodes for each time step i: We let s_i denote the chance node which indicates whether the true state of the agent is 1,2,3 or 4; o_i represents the corresponding observation *goal* or *non-goal*. The node d_i corresponds to the decision *east* or *west*. The utility node u_i represents the reward associated with the true state of the agent. For LIMIDs, the graphical representation contains a direct specification of the information sets to be used for each decision. The DAG of Figure 2 shows four slices of the LIMID for the Robot example. The assumption of no-forgetting is made explicit by drawing arcs from predecessors of decision nodes.

In contrast, the situation where the Robot does not keep record on previous observations is modelled by the LIMID shown in Figure 3.

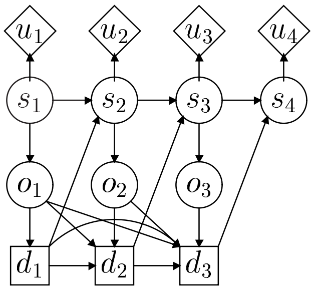


Figure 2: LIMID representing the Robot example, where all previous observations are available when decision must be taken.

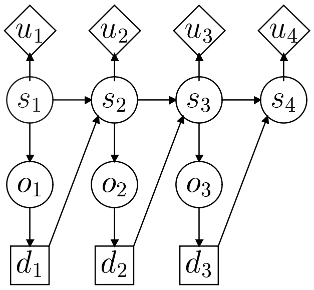


Figure 3: LIMID representing of the Robot example, where only current observations are available when decisions must be taken.

2.2 Notation and terminology

We are given a LIMID $\mathcal{L}$ with chance nodes Γ, decision nodes Δ, and utility nodes Υ. We let $V = \Gamma \cup \Delta$. Elements in V will be termed *nodes* or *variables* interchangeably. The parents of node n is denoted $\mathrm{pa}(n)$, and the *family* of d is given by $\mathrm{fa}(d) = \mathrm{pa}(d) \cup \{d\}$. For $S \subseteq V$, we let $\mathrm{fa}(S)$ be a short-hand notation for $\cup\{\mathrm{fa}(n) : n \in S\}$. Finally, $\mathrm{de}(n)$ denotes the descendants of node n.

A *policy* for $d \in \Delta$ is a function δ_d that associates with each state $x_{\mathrm{pa}(d)}$ of $\mathrm{pa}(d)$, a probability distribution $\delta_d(\cdot \,|\, x_{\mathrm{pa}(d)})$ on $\mathcal{X}_d$. A *strategy* $q = \{\delta_d : d \in \Delta\}$ determines a joint distribution of all the variables in V as

$$f_q = \prod_{r \in \Gamma} p_r \prod_{d \in \Delta} \delta_d.$$

The expected utility of q is the expectation of the total utility $U = \sum_{u \in \Upsilon} U_u$ wrt. the joint distribution of V induced by q:

$$\mathrm{EU}(q) = \sum_x f_q(x) U(x).$$

A *global maximum strategy* $\hat{q}$ in a LIMID $\mathcal{L}$ is a strategy that maximizes the expected utility. We shall also write $\mathrm{EU}(\hat{q})$ as $\mathrm{EU}(\mathcal{L})$.

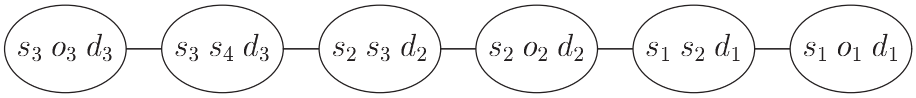


Figure 4: Junction tree for the LIMID representing of the Robot example, where only current observations are available (Figure 3).

2.3 Single Policy Updating

Single Policy Updating is an iterative procedure for evaluating general LIMIDs. It starts with an initial strategy and proceeds by modifying (updating) the policies in a random or systematically order. If the current strategy is $q = \{\delta_d : d \in \Delta\}$ and the policy for $d_i \in \Delta$ is to be updated, then the following steps are performed:

Retract: Retract the current policy for d_i from q to obtain $q_{-d_i} := q \setminus \{\delta_{d_i}\}$.

Optimize: Compute a new policy for d_i: $\tilde{\delta}_{d_i} = \arg\max_{\delta_{d_i}} \mathrm{EU}\left(q_{-d_i} \cup \{\delta_{d_i}\}\right)$

Replace: Redefine $q := q_{-d_i} \cup \{\tilde{\delta}_{d_i}\}$.

The policies are updated until they converge to a strategy in which no single policy modification can increase the expected utility. Such a strategy is termed a *local maximum strategy.*

The whole decision problem can be formulated and evaluated in terms of a *junction tree* of subsets of V. The junction tree for the LIMID in Figure 3 is shown in Figure 4.

2.4 Soluble LIMIDs

As shown in the present subsection, under general conditions the locally optimal strategies, found through Single Policy Updating, are globally optimal.

Let the symbolic expression

$$A \perp_{\mathcal{L}} B \mid S$$

denote that A and B are d-separated by S in the DAG formed by all the nodes in the LIMID $\mathcal{L}$, i.e. including the utility nodes.

A decision node d_0 in a LIMID $\mathcal{L}$ having decision nodes Δ, is said to be *extremal* in $\mathcal{L}$, if

$$u \perp_{\mathcal{L}} \mathrm{fa}(\Delta \setminus \{d_0\}) \mid \mathrm{fa}(d_0)$$

for every utility node $u \in \mathrm{de}(d_0) \cap \Upsilon$.

It is important, for the computation of an optimal strategy, that extremality is preserved under reduction:

Lemma 1 *If d is extremal in $\mathcal{L}$ then d is extremal in any reduction $\mathcal{L}'$ of $\mathcal{L}$.*

A LIMID $\mathcal{L}$ is said to have an *exact solution ordering* $d_1, \ldots, d_k$ of its decision nodes, if for all i, d_i is extremal in $\mathcal{L}$ when $d_{i+1}, \ldots, d_k$ have been converted into chance nodes. In this case, $\mathcal{L}$ is said to be *soluble*.

When Procedure Single Policy Updating is applied on a soluble LIMID $\mathcal{L}$ having exact solution ordering $d_1, \ldots, d_k$, then it always converges to a global maximum strategy when the policies initially are uniform[1], and they are updated using the order $d_k, \ldots, d_1$. In particular this leads to more efficient algorithms for solving IDs (Nilsson and Lauritzen (2000), and Madsen and Nilsson (2001)).

2.5 Redundant information in LIMIDs

In this section we show how to determine 'redundant' information for decision problems represented by LIMIDs. The results can be used to reduce the complexity of evaluating $\mathcal{L}$ during Single Policy Updating.

A node $n \in \mathrm{pa}(d)$ in $\mathcal{L}$ is said to be *non-requisite* for d, if

$$n \perp_{\mathcal{L}} (\Upsilon \cap \mathrm{de}(d)) \mid (\mathrm{fa}(d) \setminus \{n\}).$$

If n is non-requisite for d, then we say that the arc from n to d is non-requisite. A node (arc) that is not non-requisite is said to be *requisite*.

A *reduction* of $\mathcal{L}$ is obtained by successively removing non-requisite informational arcs in $\mathcal{L}$. There exists a unique reduction of $\mathcal{L}$, termed termed *minimal reduction of* $\mathcal{L}$, in which all informational arcs are requisite. The minimal reduction of $\mathcal{L}$ is denoted $\mathcal{L}_{\min}$.

It can be shown that the maximum expected utility is preserved under reduction, i.e. in particular we have that

$$\mathrm{EU}(\mathcal{L}) = \mathrm{EU}(\mathcal{L}_{\min}).$$

Furthermore, solubility is preserved under reduction, and as a consequence, Single Policy Updating applied on $\mathcal{L}_{\min}$ produces a global maximum strategy for $\mathcal{L}$.

3 Bounds for LIMIDs

When LIMIDs are intractable, we must rely on approximate methods for evaluating them. This section is concerned with developing methods for computing upper and lower bounds for intractable LIMIDs.

[1] The *uniform policy* for a decision d is given by $\bar{\delta}_d \equiv 1/|\mathcal{X}_d|$, where $\mathcal{X}_d$ denote the states that d can have.

3.1 Upper bounds

An upper bound for a LIMID $\mathcal{L}$ is a number that is larger (or equal) to $\mathrm{EU}(\mathcal{L})$. We compute an upper bound for $\mathcal{L}$ by adding fill-in arcs into the decision nodes of $\mathcal{L}$, and subsequently evaluate the obtained LIMID. We thus define:

Definition 1 A LIMID $\mathcal{L}^{'}$ is said to be a *covering* of $\mathcal{L}$ if it is obtained by successive additions of arcs into the decision nodes.

If $\mathcal{L}'$ is a covering of $\mathcal{L}$, then we write $\mathcal{L}' \supseteq \mathcal{L}$. The arcs added into a covering are termed *fill-in arcs*. Evidently, if $\mathcal{L}^{'} \supseteq \mathcal{L}$, then we have that

$$\mathrm{EU}(\mathcal{L}') \geq \mathrm{EU}(\mathcal{L}).$$

This property follows trivially because any strategy in $\mathcal{L}'$ is also a strategy in $\mathcal{L}$.

3.2 Lower bounds

Here, a lower bound for $\mathcal{L}$ is obtained by removing informational arcs in $\mathcal{L}$ to obtain a LIMID $\mathcal{L}' \subset \mathcal{L}$. Any (local or global maximum) strategy q' in $\mathcal{L}'$ clearly satisfies

$$\mathrm{EU}(q') \leq \mathrm{EU}(\mathcal{L}') \leq \mathrm{EU}(\mathcal{L}),$$

and thus $\mathrm{EU}(q')$ lower bounds the optimal strategy on $\mathcal{L}$. The strategy q' is termed a *lower bound strategy for* $\mathcal{L}$. A central task is to determine which informational arcs to remove in $\mathcal{L}$ to obtain $\mathcal{L}'$. Ideally, $\mathcal{L}'$ has two properties. Firstly, it is crucial that the evaluation of $\mathcal{L}'$ is feasible using Single Policy Updating, and secondly, we want to compute a strategy in $\mathcal{L}'$ with high expected utility. Note that, typically these two properties are conflicting. We shall take a sequential approach, which we briefly explain below. Details are given later.

Initially, we chose the LIMID, $\mathcal{L}_0$ say, obtained by removing all informational arcs in $\mathcal{L}$. Then, we sequentially re-insert the informational arcs into $\mathcal{L}_0$ to obtain a sequence

$$\mathcal{L}_0 \subset \mathcal{L}_1 \subset \mathcal{L}_2 \subset \cdots,$$

where $\mathcal{L}_i \subset \mathcal{L}$ contains exactly one informational arc which is not present in $\mathcal{L}_{i-1}$. For each LIMID $\mathcal{L}_i$ we generate, a local maximum strategy is computed using Single Policy Updating. We proceed in this manner, until we reach a LIMID $\mathcal{L}_n$ where evaluation is impossible. At this stage, the process is stopped, and the best strategy computed so far is returned.

To decide which informational arc to re-insert into $\mathcal{L}_{i-1}$ to obtain $\mathcal{L}_i$, we shall exploit the notion of requisiteness. In this way, no numerical LIMID-evaluation is performed. There are two reasons for this choice. Firstly, requisiteness is determined from the structure alone, and thus we avoid time-consuming numerical LIMID-evaluation. Secondly, if $\mathcal{L}_i$ is not soluble, and this is indeed often the case, then exact LIMID-evaluation of $\mathcal{L}_i$ is impossible.

Thus, we need a measure for 'how' requisite a parent x is for a decision node d. Clearly, a coarse measure is whether or not x is requisite for d. This is, however, insufficient for our purpose. As a first step in constructing a finer measure for the requisiteness of a parent x for a decision d, we shall introduce a bit notation. For a node $x \in V$ in $\mathcal{L}$ we let $\mathcal{L}_{x \to d}$ denote the LIMID given by $\mathcal{L}$ augmented with a fill-in arc from x into d. However, if the arc is present already or the arc introduces a cycle in the LIMID, then no arc is inserted, and in this case we simply define $\mathcal{L}_{x \to d} = \mathcal{L}$. Further, for a set $S = \{s_1, \ldots, s_n\} \subseteq V$, we let

$$\mathcal{L}_{S \to d} = ((\mathcal{L}_{s_1 \to d}) \cdots)_{s_n \to d}.$$

It can be seen that $\mathcal{L}_{S \to d}$ is invariant of the ordering of the nodes in S. Now, let $\mathcal{F}_{\mathcal{L}}(x, d)$ denote the set of subsets given by

$$\mathcal{F}_{\mathcal{L}}(x, d) = \{F \subseteq \mathrm{nd}(d) \mid \ x \text{ is requisite for } d \text{ in } \mathcal{L}_{F \to d}\}.$$

So, a set F belongs to $\mathcal{F}_{\mathcal{L}}(x, d)$ whenever x is requisite for d in the (hypothetical) case where the decision maker has additional knowledge on the true state of each variable in F. We now define:

Definition 2 Suppose x and y are parents of decision node d in a LIMID $\mathcal{L}$. Then, x is *dominated by y at d* if

$$\mathcal{F}_{\mathcal{L}}(x, d) \subset \mathcal{F}_{\mathcal{L}}(y, d). \tag{1}$$

If (1) is not satisfied, then we say that x is not dominated by y (at d). If x is not dominated by y, then it is clearly possible that x dominates y.

As the following lemma and corollary show, a requisite parent cannot be dominated by a non-requisite parent.

Lemma 2 *Suppose x is a parent of d in $\mathcal{L}$. For $F \subseteq \mathrm{pa}(d)$ we have*

1. *If x is requisite for d, then $F \in \mathcal{F}_{\mathcal{L}}(x, d)$.*
2. *If x is non-requisite for d, then $F \notin \mathcal{F}_{\mathcal{L}}(x, d)$.*

Proof: By definition, $\mathcal{L} = \mathcal{L}_{F \to d}$, and the result follows trivially. •

As a corollary, we immediately obtain:

Corollary 1 *Suppose x is non-requisite for d in $\mathcal{L}$, and y is requisite for d. Then y is not dominated by x at d.*

Proof: By Lemma 2, $\mathcal{F}_{\mathcal{L}}(y, d) \not\subset \mathcal{F}_{\mathcal{L}}(x, d)$, and the result follows. •

In Section 4 we show some additional properties of the notion of domination.

4 Bounds for POMDPs

A Partially Observed Markov Decision Processes (POMDPs) (Lovejoy 1991; Kaelbling *et al.* 1998) is a generalization of a Markov decision process that allows for incomplete information regarding the state of the system. So, a POMDP is a simple model of sequential decision making under uncertainty. Despite its simplicity, the significant applied potential for such processes remains largely unrealized, due to a lack of tractable solution methodologies.

In this section, we define POMDPs in terms of LIMIDs, and present algorithms for computing bounds of them. The performance of the algorithms is illustrated on the Robot example.

4.1 POMDPs

Consider a finite state, finite action POMDP. Let $t \in \{1, 2, \ldots, k\}$ represent stage t of the decision problem. The finite parameter k represents the planning horizon of the POMDP. Let s_t be the state of the system at stage t. The state must be a member of a finite state set S. The state process $\{s_t : t = 1, \ldots, k\}$ makes transitions according to the conditional probability distribution $P(s_t | s_{t-1}, a_{t-1})$, where a_t is the action selected at stage t.

In contrast to MDPs we do not have full information on the current state s_t when the action a_t is selected. In stead, we have available an observation o_t generated from the distribution $P(o_t | s_t)$.

Let $u_t(s_t)$, $s_t \in S$ be the one-stage reward accrued at stage t. At time $k+1$ a salvage value u_{k+1} is given based on only s_{k+1} in order to handle process termination. POMDPs are solved by finding the strategy, which obtains the highest expected total reward over the entire planning horizon. Note, the partial observability of the state makes the entire past of observation variables and decision variables requisite. Figure 2 shows all the above quantities for the Robot example.

4.2 Lower bounds for POMDPs

To compute lower bounds of a POMDP $\mathcal{P}$, we will use an anytime algorithm, operating on the LIMID version $\mathcal{L}$ of $\mathcal{P}$.

The procedure starts with a LIMID, denoted $\mathcal{L}_0$, obtained by removing all informational arcs in $\mathcal{L}$. A local maximum strategy is then obtained by applying SPU on $\mathcal{L}_0$. The procedure now proceeds by iteratively re-inserting the informational arcs into $\mathcal{L}_0$.

To decide which informational arc to re-insert we apply the notion of domination as described in the previous section. For this purpose we need Theorem 1 below. Here, for a parent $x \in \mathrm{pa}(d)$ of d and a set of parents $A \subseteq \mathrm{pa}(d)$ of d, we say that x *is (not) dominated by* A *at* d, if x is (not) dominated by any element of A at d.

Letting O_t and D_t be short-hand notations for $\{o_1, \ldots, o_t\}$ and $\{d_1, \ldots, d_t\}$ respectively, we now have:

Theorem 1 *Suppose $\mathcal{L}$ corresponds to a k-stage POMDP. For $t' < t$, the following statements hold in $\mathcal{L}$:*

1. *$o_{t'+1}$ is not dominated by $O_{t'} \cup D_{t'-1}$ at d_t.*
2. *$d_{t'}$ is not dominated by $O_{t'} \cup D_{t'-1}$ at d_t.*
3. *At d_t, $o_{t'+1}$ is not dominated by $d_{t'}$ and $d_{t'}$ is not dominated by $o_{t'+1}$.*

Proof: We only provide a brief sketch of the proof. From the definition of requisiteness it follows that, in $\mathcal{L}_{s_{t'} \to d_t}$, $o_{t'+1}$ and $d_{t'}$ are requisite for d_t, whereas the nodes in $O_{t'} \cup D_{t'-1}$ are non-requisite for d_t. As a consequence, we have:

$$\begin{aligned} s_{t'} &\in \mathcal{F}_{\mathcal{L}}(o_{t'+1}, d_t) && (2) \\ s_{t'} &\in \mathcal{F}_{\mathcal{L}}(d_{t'}, d_t) && (3) \\ s_{t'} &\notin \mathcal{F}_{\mathcal{L}}(o_{i+1}, d_t) \cup \mathcal{F}_{\mathcal{L}}(d_i, d_t) \quad \text{for all } i < t'. && (4) \end{aligned}$$

Now, the first property follows from (2) and (4), and the second property follows from (3) and (4). Finally, the third property follows because

$$\begin{aligned} \mathcal{F}_{\mathcal{L}}(d_{t'}, d_t) &= \mathcal{F}_{\mathcal{L}}(o_{t'+1}, d_t) \\ &= \{F \subseteq \mathrm{nd}(d_t) \mid F \cap \{s_{t'}, \ldots, s_t\} = \emptyset\}. \end{aligned}$$

This completes the proof. •

So, the above theorem states that, in a POMDP, earlier observations (decisions) do not dominate more recent observations (decisions). In fact, the stronger result stating that earlier observations (decisions) are dominated by more recent observations (decisions) can also be shown. However, this is beyond the scope of the present paper.

The full algorithm for computing lower bound strategies for a POMDP is presented below in algorithmic terms. Here, and throughout, we abbreviate Procedure Single Policy Updating into SPU.

Input: LIMID $\mathcal{L}$ corresponding to POMDP $\mathcal{P}$ with decisions $d_1, \ldots, d_k$.

Output: Lower bound strategy q for $\mathcal{L}$ and $\mathrm{EU}(q)$.

Initialization: $\mathcal{L}_0 \leftarrow$ LIMID obtained by removing all informational arcs in $\mathcal{L}$.
$q_0 \leftarrow$ local maximum strategy of $\mathcal{L}_0$ obtained from SPU starting from the uniform strategy.
$m \leftarrow 0$.

Repeat (until stopping rule are met or $m = k$)

1. $\mathcal{L}' \leftarrow \mathcal{L}_m$.
2. For $t = 1 \ldots k - m$ do

   $\mathcal{L}' \leftarrow (\mathcal{L}')_{\{o_t, d_t\} \to d_{t+m}}$.
3. (a) $\mathcal{L}_{m+1} \leftarrow \mathcal{L}'$;
   (b) $q' \leftarrow$ local maximum strategy of $\mathcal{L}_{m+1}$ obtained from SPU starting from the uniform strategy;
   (c) $q_{m+1} = \arg\max \mathrm{EU}(q', q_m)$.
4. $m \leftarrow m + 1$.

Procedure 1 *An anytime procedure for finding a 'lower bound' strategy for a POMDP $\mathcal{P}$.*

A few comments are in place at this stage:

1. It can readily be seen that Procedure 1 has the following properties:
   (a) **(Iterative Improvement)** $\mathrm{EU}(q_m) \geq \mathrm{EU}(q_{m-1})$ for $m = 1, \ldots, k$.
   (b) **(Convergence)** $\mathrm{EU}(q_k) = \mathrm{EU}(\hat{q})$, where $\mathrm{EU}(\hat{q}) = \mathrm{EU}(\mathcal{L})$.
2. Note that all computations apart from step 3(b) are simple graph manipulations on a LIMID.

3. As indicated in Figure 5, the complexity of $\mathcal{L}_m$ is an increasing function of m. It is therefore important that Procedure 1 can be interrupted at any point, and return the last strategy q_m. For instance, the user can specify a certain time-frame (s)he is willing to wait for an executable strategy – once this time-frame is exceeded, the currently available strategy is returned.

4. Note, Procedure 1 starts by inserting fill-in arcs from the latest observations and decisions because they dominate earlier observations and decisions. However, intuitively, this approach also makes sense: More recent observations tend to be more informative than distant observations.

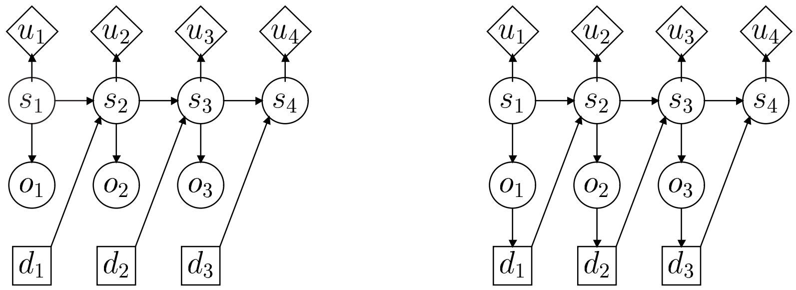


Figure 5: The LIMIDs $\mathcal{L}_0$ and $\mathcal{L}_1$ obtained by applying Procedure 1 on $\mathcal{L}$ in Figure 2, and yielding lower bounds strategies for $\mathcal{L}$.

4.3 Upper bounds for POMDPs

The initial upper bound of a LIMID $\mathcal{L}$ is achieved by adding a fill-in arc, for each stage t, from node s_t into decision d_t. The obtained LIMID, denoted $\mathcal{L}^0$, then corresponds to a traditional MDP. Starting from $\mathcal{L}^0$, the following anytime algorithm iteratively improves this initial upper bound.

Input: LIMID $\mathcal{L}$ corresponding to $\mathcal{P}$ with decisions $d_1, \ldots, d_k$.

Output: Soluble cover $\mathcal{L}^m$ of $\mathcal{L}$ and $\mathrm{EU}(\mathcal{L}^m)$.

Initialization: $m \leftarrow 0$.

Repeat (until stopping rule are met or $m = k$)

1. $\mathcal{L}' \leftarrow \mathcal{L}$.
2. For $t = 1 \ldots k - m$ do

   $\mathcal{L}' \leftarrow (\mathcal{L}')_{s_t \to d_{t+m}}$
3. $\mathcal{L}^m \leftarrow \mathcal{L}'$; $\mathrm{EU}(\mathcal{L}^m) \leftarrow$ maximum expected utility of $\mathcal{L}^m$ obtained by performing SPU on $(\mathcal{L}^m)_{\min}$.
4. $m \leftarrow m + 1$.

Procedure 2 *An anytime procedure for finding an 'upper bound' strategy for a POMDP $\mathcal{P}$.*

Procedure 2 has the following important properties:

Theorem 2 *Suppose we apply Procedure 2 on a LIMID $\mathcal{L}$. Then the following statements hold for all $m = 0, \ldots, k$:*

1. **(Soluble Covering)** $\mathcal{L}^m$ *is a soluble cover of* $\mathcal{L}$.
2. **(Iterative Improvement)** $\mathrm{EU}(\mathcal{L}^{m+1}) \leq \mathrm{EU}(\mathcal{L}^m)$ *for* $m < k$.
3. **(Convergence)** $\mathrm{EU}(\mathcal{L}^k) = \mathrm{EU}(\mathcal{L})$.

Proof: To prove the first statement, note that $\mathcal{L}^m$ is trivial a covering of $\mathcal{L}$. Further, it can be seen that $\mathcal{L}^m$ obeys the exact solution ordering $d_1, \ldots, d_k$, which proves 1.

To see the second statement, it may be advantageous to regard Figure 6 and 7 while reading the arguments below. For $i \in \{m, m+1\}$, we note that $\mathcal{L}^i$ can be written as:

$$\mathcal{L}^i \;=\; ((\mathcal{L}_{s_1 \to d_{1+i}}) \cdots)_{s_{k-i} \to d_k}.$$

Let $\mathcal{L}'$ be the LIMID $\mathcal{L}^m$ augmented with the fill-in arcs in $\mathcal{L}^{m+1}$, i. e.

$$\mathcal{L}' \;=\; (((\mathcal{L}_{s_1 \to d_{1+m}}))_{\{s_1, s_2\} \to d_{2+m}} \cdots)_{\{s_{k-m-1}, s_{k-m}\} \to d_k}$$

By construction, $\mathcal{L}' \supseteq \mathcal{L}^m$ and $\mathcal{L}' \supseteq \mathcal{L}^{m+1}$ and consequently,

$$\mathrm{EU}(\mathcal{L}') \;\geq\; \max\{\mathrm{EU}(\mathcal{L}^m), \mathrm{EU}(\mathcal{L}^{m+1})\}. \tag{5}$$

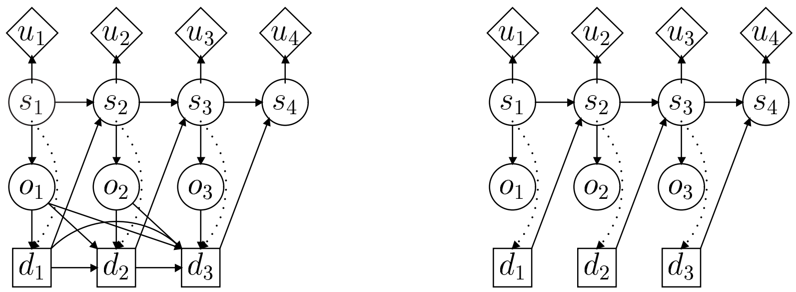


Figure 6: The soluble cover $\mathcal{L}^0$, and its minimal reduction obtained by applying Procedure 2 on $\mathcal{L}$ in Figure 2.. Dotted arcs are fill-in arcs.

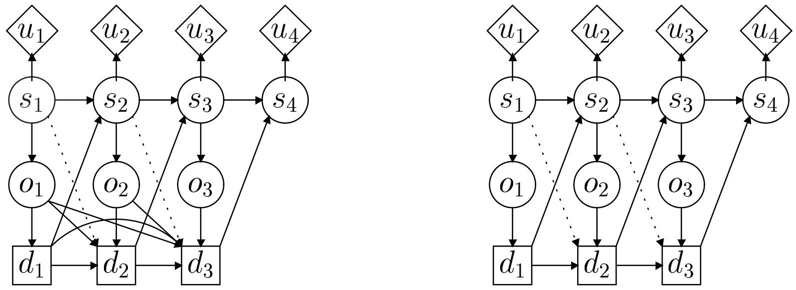


Figure 7: The soluble cover $\mathcal{L}^1$, and its minimal reduction obtained by applying Procedure 2 on $\mathcal{L}$ in Figure 2. Dotted arcs are fill-in arcs.

In $\mathcal{L}'$, the arcs from s_t ($t \in \{1, \ldots, k-m-1\}$) into d_{t+m+1} are non-requisite, because every trail from s_t to $(\Upsilon \cap \mathrm{de}(d_{t+m+1}))$ are blocked by $\{d_{t+1}, s_{t+1}\}$. Thus, the arcs from s_t to d_{t+m+1} can be removed without loss of maximal expected utility. So, $\mathrm{EU}(\mathcal{L}') = \mathrm{EU}(\mathcal{L}^m)$ and from (5) it follows that $\mathrm{EU}(\mathcal{L}^m) \geq \mathrm{EU}(\mathcal{L}^{m+1})$ as desired.

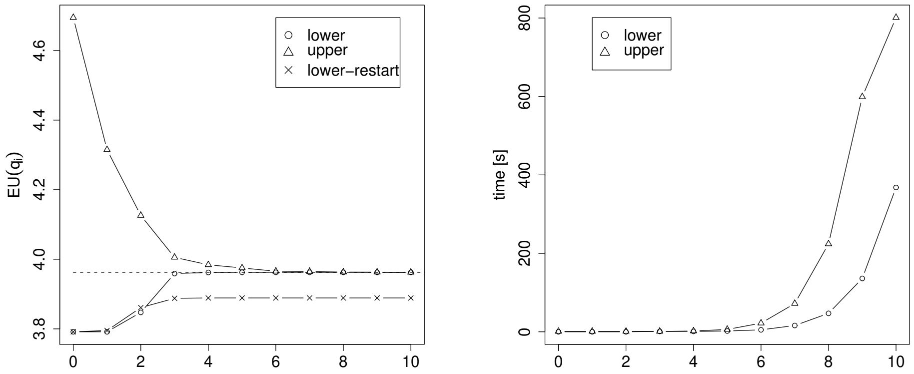
Finally, the third statement is trivial since $\mathcal{L}^k = \mathcal{L}$. •

Example 2 Figure 6 and 7 show two steps of Procedure 2 applied to the robot example.

4.4 Bounds for the Robot example

Now we return to the Robot example. Instead of the rather trivial case with 3 decisions, shown in Figure 2, versions with 10 and 99 decisions are considered.

In Figure 8(a), the lower and upper bounds obtained by the above procedures in the case of 10 decisions are given. Here, the 'x' indicates the



(a) (b)

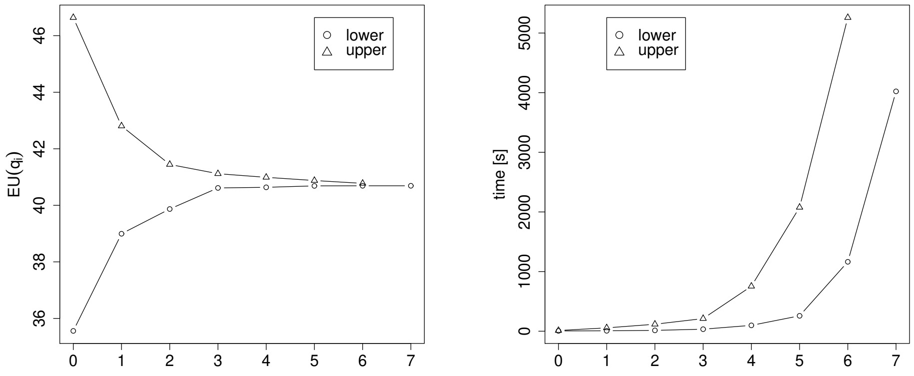
Figure 8: Running Procedure 1 and 2 for the Robot example with 10 decisions. The curve of "lower-restart" denotes Procedure 1, where we start the SPU iteration from the last strategy obtained. The dotted line shows the expected value of the global maximum strategy of the POMDP – which in the case of only 10 decisions is tractable. Figure (b) depicts time-usages.

modification of Procedure 1 in which the strategy q_i is computed by performing SPU on $\mathcal{L}_i$ starting from the previous strategy q_{i-1}. It can be seen that this approach rather quickly is trapped in local maximum strategy. In Figure 8(b), the running times are on a 600 MHz Pentium III machine with 256MB Ram. Some speed optimizations could be achieved by tuning the code with respect to efficiency.

In Figure 9, the case with 99 decisions is shown. Surprisingly, we achieve a similar pattern as the case with 10 decisions: The upper bound and lower bound curves quickly converge, and thus, gives us a guarantee that the obtained lower bound strategy is close to the global maximum strategy for $\mathcal{L}$.

5 Lower bounds for general LIMIDs

In the next two sections we present methods for computing bounds for general LIMIDs. The methods are similar, though more complex, to those presented for the special case of POMDPs.



(a) (b)

Figure 9: Running Procedure 1 and 2 for the Robot example with 99 decisions. (a) shows the expected utility of $\mathrm{EU}(q_i)$. Figure (b) shows time usages.

5.1 The Maze Problem

We will illustrate the algorithms for general LIMIDs on a more complicated version of the robot navigation example used by (Horsch and Poole 1998; Littman *et al.* 1995). Here, the environment is given by the maze shown in Figure 10.

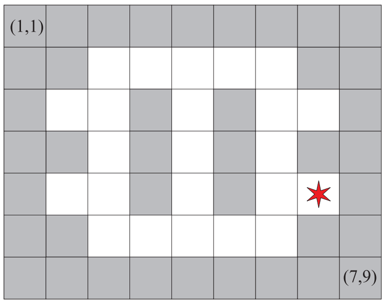


Figure 10: Maze environment of the extended example. Shaded tiles indicate walls and the star denotes the goal state.

To begin with, the agent is placed at random in one of the 22 valid non-goal states and the task is to navigate to the goal state in at most 10 steps. Four sensors, one in each compass direction, inform the agent about presence of walls in the neighboring tiles. Based on this information the agent decides whether to move `n,s,e,w` or `stay` in its current position. After 10 of

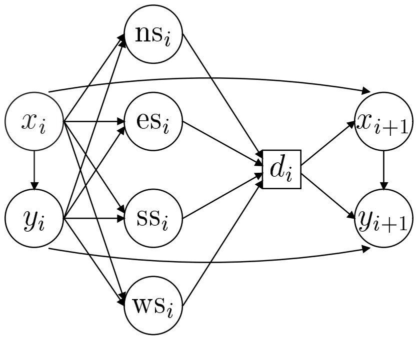


Figure 11: Basic structure of the LIMID representing the maze example. The structure is repeated 10 times, and based on the state of x_{11} and y_{11} a utility node u gives the terminal reward. In the above LIMID, all no-forgetting arcs are implicit.

these decisions the game ends and a reward of 1 is given if the current state and the goal state coincide, otherwise no reward is obtained. The described navigation task may be represented as a POMDP, but instead we will exploit the factoring nature of sensors and position.

Figure 11 shows the LIMID representation of the task. Three different models are examined:

Maze1:	perfect sensors and perfect actor.
Maze2:	noisy sensors and perfect actor.
Maze3:	noisy sensors and noisy actor.

With perfect actors, moving in a direction where there is a wall always fails. Otherwise, the desired moves always succeeds. Noisy actors on the other hand have a 0.089 probability of not moving, a 0.001 probability of moving in the opposite direction, a probability of 0.01 for moving in the $+90$ degree direction and finally a 0.01 probability for the -90 degree direction. Except for stay actions, which always succeed, the probability of moving in the desired direction is thus 0.89. In case of walls, all directions containing a wall are assigned zero probability and the remaining probabilities are normalized.

With perfect sensors, a wall is recognized if and only if there is a wall. With noisy-sensors, a sensor fails to recognize a wall with probability 0.1, and with probability 0.05 a wall is reported even though no existed.

5.2 Procedure Information Augmentation

Procedure Information Augmentation is an iterative procedure for computing lower bound strategies for general LIMIDs

We start by introducing a bit notation: For $a \in \text{pa}(d)$, we let $\mathcal{L}_{a \not\to d}$ denote the LIMID $\mathcal{L}$, in which the arc from a into d is removed. We use a similar notation for sets of nodes. For a given LIMID $\mathcal{L}$, the procedure starts with a LIMID $\mathcal{L}_0$ obtained by removing all informational arcs from $\mathcal{L}$. Then, it proceeds by iteratively re-inserting the informational arcs into $\mathcal{L}_0$. As in the POMDP case, we use the notion of domination to choose which informational arcs to re-insert.

At some stage in Procedure Information Augmentation, assume the current LIMID is $\mathcal{L}_i \subseteq \mathcal{L}$, and an informational arc is to be inserted into d. In deciding which informational arc to insert into d, we need to define the *candidate nodes*, i.e. the nodes that should be taken into consideration as being a new parent of d. The potential class of nodes consists of nodes that are parents of d in $\mathcal{L}$, but not parents of d in $\mathcal{L}_i$. However, evidently there is no effect in choosing a node n which would be non-requisite for d in $(\mathcal{L}_i)_{n \to d}$: While not affecting the maximal expected utility, such non-requisite arcs introduce unnecessary computation. Thus, the set of candidate nodes becomes:

$$C \;=\; \{n \in \text{pa}_{\mathcal{L}}(d) \setminus \text{pa}_{\mathcal{L}_i}(d) \mid \; n \text{ is requisite for } d \text{ in } (\mathcal{L}_i)_{n \to d}\ \}.$$

We now consider the two possible cases:

If $C = \emptyset$, then no informational arc is inserted into $\mathcal{L}_i$, and we simply let $\mathcal{L}_{i+1} = \mathcal{L}_i$.

If $C \neq \emptyset$, then we chose one of the candidates in C. As in the POMDP case, we shall use the notion of domination to guide us in this choice. Consider, for the moment the LIMID $(\mathcal{L}_i)_{C \to d}$ obtained by adding fill-in arcs from all candidate nodes into d. In $(\mathcal{L}_i)_{C \to d}$, we may proceed by searching for a candidate node in C that dominates all other candidate nodes (at d). However, in many cases such a node does not exist, and even if it existed, it may be difficult to find. Instead, in $(\mathcal{L}_i)_{C \to d}$, we search for a set $C' \subseteq C$ with the property that each node in C' is *not* dominated by $C \setminus C'$. Clearly, $C' = C$ has this property, but as we want the cardinality of C' to be as small as possible, we proceed by performing the following steps:

1. Let $A = V \setminus \text{fa}_{(\mathcal{L}_i)_{C \to d}}(d)$,
2. Define $\mathcal{L}' = ((\mathcal{L}_i)_{C \to d})_{A \to d}$.
3. Define $R = \{n \mid n \text{ requisite for } d \text{ in } \mathcal{L}'\ \}$.
4. If $R \cap C = \emptyset$, then

   $\mathcal{L}' \to (\mathcal{L}')_{B \not\to d}$, where $B = A \cap R$, and goto 3.

5. $\mathcal{L}_{i+1} = (\mathcal{L}_i)_{n\to d}$, where n is an arbitrary node in $C' = R \cap C$.

So, in step 1 and 2, fill-in arcs are inserted in $(\mathcal{L}_i)_{C\to d}$ from all possible nodes into d (such that no cycles are created). In step 3 and 4, we test whether some of the candidate nodes are requisite. If this is not the case, then we successively remove fill-in arcs that are requisite, until some of the candidate nodes become requisite. After performing these steps, $C' = C \cap R \neq \emptyset$, and it can be seen that, in $\mathcal{L}'$, the nodes in C' are requisite for d whereas the nodes in $C \setminus C'$ are non-requisite for d. Consequently, each node in C' is not dominated by $C \setminus C'$ (at d) as wanted.

The full Procedure Information Augmentation is shown below.

Input: LIMID $\mathcal{L}$ with decision nodes $d_1, d_2, \ldots, d_k$.

Output: LIMID $\mathcal{L}_j \subseteq \mathcal{L}$; local maximum strategy q_j for $\mathcal{L}_j$, and $\mathrm{EU}(q_j)$.

Initialization: $\mathcal{L}_0 \leftarrow$ LIMID obtained by removing all informational arcs in $\mathcal{L}$; $j \leftarrow 0$.

Repeat (until stopping criteria are met or $\mathcal{L}_j = \mathcal{L}$)

For $i = k \ldots 1$ do:

1. (Find candidate set)
   (a) $d \leftarrow d_i$;
   (b) $C = \{n \in \mathrm{pa}_{\mathcal{L}}(d) \setminus \mathrm{pa}_{\mathcal{L}_j}(d) \mid n \text{ is req. for } d \text{ in } (\mathcal{L}_j)_{n\to d}\}$.
2. (Find and evaluate new LIMID)
   If $C = \emptyset$, then $\mathcal{L}_{j+1} \leftarrow \mathcal{L}_j$ and goto 3.
   If $C \neq \emptyset$, then do
   (a) $A = V \setminus \mathrm{fa}_{(\mathcal{L}_j)_{C\to d}}(d)$;
   $\mathcal{L}' \leftarrow ((\mathcal{L}_j)_{C\to d})_{A\to d}$;
   $R \leftarrow \{n \mid n \text{ requisite for } d \text{ in } \mathcal{L}'\ \}$.
   (b) While $R \cap C = \emptyset$ do
   $\mathcal{L}' \leftarrow (\mathcal{L}')_{B\not\to d}$, where $B = A \cap R$.
   $R \leftarrow \{n \mid n \text{ requisite for } d \text{ in } \mathcal{L}'\}$.
   (c) $n \leftarrow$ arbitrary node in $R \cap C$.
   $\mathcal{L}_{j+1} \leftarrow (\mathcal{L}_j)_{n\to d}$.
   (d) $q' \leftarrow$ local maximum strategy of $\mathcal{L}_{j+1}$ obtained by performing SPU on $\mathcal{L}_{j+1}$ starting from the uniform strategy.
   (e) $q_{j+1} = \arg\max\{\mathrm{EU}(q) : q \in \{q', q_j\}\}$.
3. $j \leftarrow j + 1$

Procedure 3 *Procedure Information Augmentation for finding a lower bound strategy in $\mathcal{L}$.*

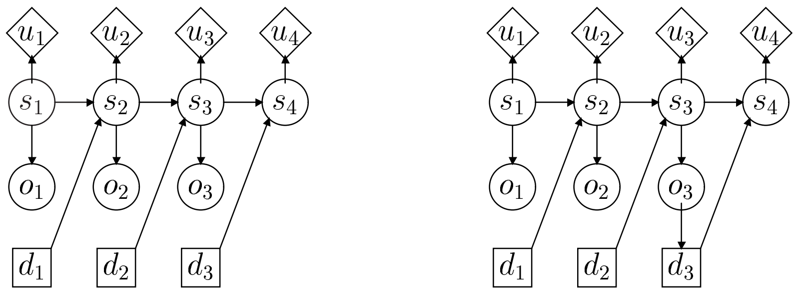


Figure 12: The LIMIDs $\mathcal{L}_0$ and $\mathcal{L}_1$ obtained by performing Procedure Information Augmentation on $\mathcal{L}$.

Note that all computations apart from step 2(d) are simple graph manipulations on $\mathcal{L}$.

To perform Step 1(b) in Procedure Information Augmentation efficiently, we will need Theorem 3 below. It says that, under suitable conditions, non-requisiteness of a decision node d in $\mathcal{L}$ is equivalent to non-requisiteness of d in $\mathcal{L}_{n \to d}$. The theorem is proved in Appendix A.

Theorem 3 *Suppose d is a decision node in $\mathcal{L}$, and let $\mathcal{L}' = \mathcal{L}_{n \to d}$, where $n \in \mathrm{nd}(d)$. Then, n is non-requisite for d in $\mathcal{L}'$ if and only if*

$$n \perp_{\mathcal{L}} (\Upsilon \cap \mathrm{de}(d)) \mid \mathrm{fa}(d). \tag{6}$$

Recall that the procedure presented in the previous section starts by inserting informational arcs into the decision nodes from nodes that are closer in time. The above procedure, for the special case of POMDPs, produces similar LIMIDs. This is illustrated in the following example:

Example 3 Now, we re-visit the simple POMDP $\mathcal{L}$ in Figure 2. Let $\mathcal{L}_0$ be the LIMID in which every information arc in $\mathcal{L}$ is removed as shown in Figure 12. Suppose we want to refine the informational set for the last decision d_3. The candidate sets C is given by:

$$C = \{o_1, o_2, o_3, d_1, d_2\}.$$

Since $C \neq \emptyset$, we define A, $\mathcal{L}'$, and R by (letting $S_i \equiv \{s_1, \ldots, s_i\}$):

$$\begin{aligned}
A &= V \setminus \mathrm{fa}_{(\mathcal{L}_0)_{C \to d_3}}(d_3) \\
&= S_4 \\
\mathcal{L}' &= ((\mathcal{L}_0)_{C \to d_3})_{A \to d_3} \\
&= \mathcal{L}_{S_3 \to d_3} \quad \text{(because } s_4 \text{ is a descendant of } d_3\text{)} \\
R &= \{n \mid \ n \text{ is requisite for } d_3 \text{ in } \mathcal{L}' \ \} \\
&= \{s_3\}
\end{aligned}$$

Because $R \cap C = \emptyset$, we redefine $\mathcal{L}'$ and R as

$$\begin{aligned} \mathcal{L}' &\leftarrow (\mathcal{L}')_{(A\cap R)\not\to d_3} \\ &= \mathcal{L}_{S_2 \to d_3} \quad \text{(because } A \cap R = \{s_3\}\text{)}. \\ R &\leftarrow \{n \mid n \text{ is requisite for } d_3 \text{ in } \mathcal{L}'\} \\ &= \{s_2, d_2, o_3\}. \end{aligned}$$

Now, $R \cap C = \{d_2, o_3\} \neq \emptyset$, and thus we arbitrarily pick a node from $R \cap C$, say o_3, to obtain $\mathcal{L}_1 = (\mathcal{L}_0)_{o_3 \to d_3}$ (see Figure 12).

Example 4 (Lower bounds for the Maze Problem) Here, we indicate how Procedure Information Augmentation compute lower bounds for the Maze problem. The procedure starts with the LIMID $\mathcal{L}_0$ containing no informational arcs. During the first cycle[2], the procedure inserts an informational into each decision node d_i $(i > 1)$[3] from an arbitrarily node chosen from the set

$$\{\text{ns}_i, \text{es}_i, \text{ws}_i, \text{ss}_i, d_{i-1}\},$$

i.e. the latest sensor nodes and the previous decision node. During the second cycle, another node from the above set is chosen. This continues for five cycles. In cycle 6–10, the informational arc inserted into d_i $(i > 2)$ is chosen from the sensor nodes $\text{ns}_{i-1}, \text{es}_{i-1}, \text{ws}_{i-1}, \text{ss}_{i-1}$ and decision d_{i-2}.

In **Maze1** (perfect sensors and perfect actors), the following strategy is optimal:

Decision	**Action(s)**
d_1:	Go south if possible; else, go east if possible; else go west.
$d_2, \ldots, d_8$:	go south if possible; else go east.
d_{10}:	Go east.

Note, the optimal strategy uses only current sensor input, i. e. no knowledge about the past is necessary. From the above discussion it can be seen that Procedure Information Augmentation always finds the optimal strategy after performing five cycles (and possible within fewer cycles).

In **Maze2** and **Maze3**, the optimal strategy is unknown. Figure 13 shows the results by applying Procedure Information Augmentation for the different setups of the maze problem. Using our implementation, it takes approximately 2400 seconds to perform 10 cycles. After 10 cycles, the following results are obtained:

	Maze1	**Maze2**	**Maze3**
Expected utility	1.00	0.998685	0.897123

[2]Here, a cycle means that an informational arc has been inserted into each decision node (unless the candidate set is empty).

[3]For the case $i = 1$, an arbitrary node is chosen from the set $\{\text{ns}_1, \text{es}_1, \text{ws}_1, \text{ss}_1\}$

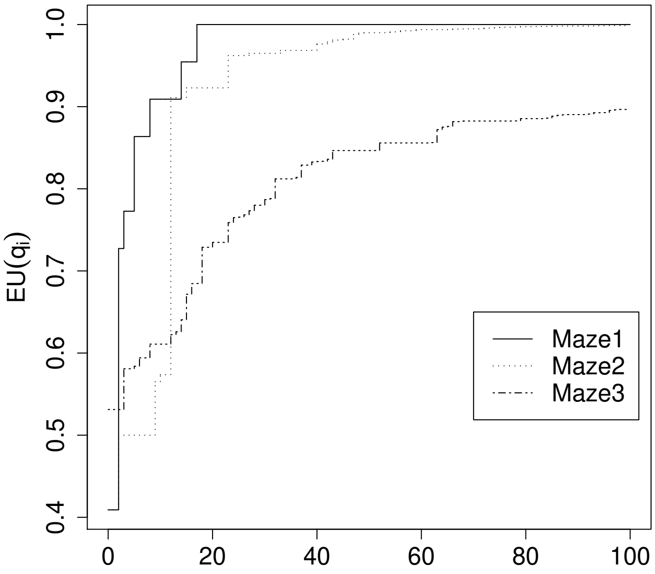


Figure 13: The obtained expected utilities after performing Procedure Information Augmentation with the three different setups.

6 Upper bounds for general LIMIDs

The approach we shall take for computing upper bounds of a general LIMID $\mathcal{L}$ consists of the following steps:

1. Construct a 'low complexity' upper bound, say $\mathcal{L}^0$, of $\mathcal{L}$ with few requisite informational arcs.

2. Start from $\mathcal{L}^0$, and iteratively generate upper bounds $\mathcal{L}^1, \mathcal{L}^2 \ldots$ of $\mathcal{L}$ with decreasing expected utility:

$$\mathrm{EU}(\mathcal{L}^0) \geq \mathrm{EU}(\mathcal{L}^1) \geq \mathrm{EU}(\mathcal{L}^2) \ldots$$

These tasks are described in subsequent subsections.

6.1 A 'low complexity' upper bound

Suppose we are given a LIMID $\mathcal{L}^{k+1}$ having decision nodes $d_1, \ldots, d_k$ and want to compute an upper bound for $\mathrm{EU}(\mathcal{L}^{k+1})$. We will perform this task successively: We start by 'making' decision node d_k extremal by adding arcs into d_k. Then we convert d_k into a chance node, and 'make' d_{k-1} extremal by adding arcs into d_{k-1}. Proceeding in this way, we eventually construct a soluble covering of $\mathcal{L}$. At each stage of this procedure, a subset of the decision nodes are made extremal. We thus define:

Definition 3 A *partial solution ordering* $d_i, \ldots, d_k$ with the property that, for all $j \geq i$, d_j is extremal in $\mathcal{L}$ when $d_{j+1}, \ldots, d_k$ have been converted into chance nodes.

As shown in Lauritzen and Nilsson (2001), an exact solution ordering of $\mathcal{L}$ is preserved under reduction. Similarly, a partial solution ordering is preserved under reduction:

Theorem 4 *If $\mathcal{L}'$ is a reduction of $\mathcal{L}$, and $\mathcal{L}$ has partial solution ordering $d_i, \ldots, d_k$, then $\mathcal{L}'$ has partial solution ordering $d_i, \ldots, d_k$.*

Proof: Suppose $\mathcal{L}$ is soluble and let $d_i, \ldots, d_k$ be an partial solution ordering. Let $\mathcal{L}'$ be a reduction of $\mathcal{L}$ obtained by removing an arc from some non-requisite parent a_l into some decision node d_l. We will show that $\mathcal{L}'$ is soluble with partial solution ordering $d_i, \ldots, d_k$. So, for $j \geq i$, we want to show that d_j is extremal in $\mathcal{L}'$ where $d_{j+1}, \ldots, d_k$ have been converted into chance nodes. We consider two cases.

1. $j \geq l$. In this case it follows because extremality is preserved under reduction (Lemma 1).
2. $j < l$. In this case it follows because d-separation is preserved under reduction.

This completes the proof. •

In general, there is no guarantee that a decision node d can be 'made' extremal by inserting arcs into it. However, if no other decision node descends d, then d can always be made extremal. Letting $\Delta_i \equiv \{d_1, \ldots, d_i\}$, we have:

Theorem 5 *Suppose $\mathcal{L}$ has decision nodes $\Delta = \{d_1, \ldots, d_k\}$. If*

$$\mathrm{de}(d_{i+1}) \cap \Delta_i = \emptyset, \tag{7}$$

then

$$\mathcal{L}' \quad = \quad ((\mathcal{L}_{\mathrm{fa}(\Delta_1) \to d_1}) \cdots)_{\mathrm{fa}(\Delta_{k-1}) \to d_k},$$

is a soluble covering of $\mathcal{L}$ with exact solution ordering $d_1, \ldots, d_k$.

Proof: By construction $\mathrm{fa}_{\mathcal{L}'}(d_i) \supseteq \mathrm{fa}_{\mathcal{L}'}(\Delta_{i-1})$ for all i, and the result follows trivially from the definition of requisiteness. •

Because $\mathcal{L}$ is acyclic, we can always order the decision nodes in $\mathcal{L}$ so that (7) is satisfied.

Now, we are able to sketch the procedure. For the initial LIMID $\mathcal{L}^{k+1}$, we assume that its decision nodes $\Delta = \{d_1, \ldots, d_k\}$ satisfy (7). Starting from $\mathcal{L}^{k+1}$, we successively generate LIMIDs $\mathcal{L}^k, \ldots, \mathcal{L}^1$ such that $\mathcal{L}^{j+1}$ has partial solution ordering $d_{j+1}, \ldots, d_k$. From $\mathcal{L}^{j+1}$ we generate $\mathcal{L}^j$ by performing the following steps (abbreviating $\mathcal{L} = \mathcal{L}^{j+1}$):

1. Compute N_j by

$$N_j = \arg\min_N \left\{ |N| \;\middle|\; \begin{array}{c} N \text{ are requisite parents of } d_j \text{ in } \mathcal{L}_{N \to d_j} \\ \wedge \\ \mathcal{L}_{N \to d_j} \text{ has partial solution ordering} \\ d_j, \ldots, d_k. \end{array} \right\}. \quad (8)$$

2. Let $\mathcal{L}^j = \mathcal{L}_{N_j \to d_j}$;

3. Remove non-requisite arcs: $\mathcal{L}^j \leftarrow (\mathcal{L}^j)_{\min}$.

Because of step 3, the final LIMID $\mathcal{L}^1$ may not be a covering of $\mathcal{L}^{k+1}$. However, we have that

$$\mathrm{EU}(\mathcal{L}^1) \geq \mathrm{EU}(\mathcal{L}^{k+1}),$$

because during the construction of $\mathcal{L}^1$ we only remove non-requisite informational arcs.

A non-trivial step in the above procedure, is to find the subset N_j in (8). The following theorems shows how this can be done. The first theorem states conditions under which a partial solution ordering $d_{j+1}, \ldots, d_k$ in $\mathcal{L}$ is preserved in a LIMID of the form $\mathcal{L}_{B \to d_j}$:

Theorem 6 *Suppose $\mathcal{L}$ has partial solution ordering $d_{j+1}, \ldots, d_k$, and assume the decision nodes satisfy (7). Let*

$$B_j = \begin{cases} V & j = k \\ \cap_{i=j+1}^k \{n \in V \mid n \perp_{\mathcal{L}} (\Upsilon \cap \mathrm{de}(d_i)) \mid \mathrm{fa}(d_i)\} & j < k. \end{cases} \quad (9)$$

Assume $B \subseteq \mathrm{nd}(d_j)$. Then $\mathcal{L}_{B \to d_j}$ has partial solution ordering $d_{j+1}, \ldots, d_k$ if and only if $B \subseteq B_j$.

The following theorem gives a similar, though different, result. With the same assumptions as above, it gives conditions under which $\mathcal{L}_{B \to d_j}$ has partial solution ordering $d_j, \ldots, d_k$.

Theorem 7 *Suppose $\mathcal{L}$ has partial solution ordering $d_{j+1}, \ldots, d_k$, and assume the decision nodes satisfy (7). Let B_j be given in (9), and assume $B \subseteq \mathrm{nd}(d_j)$. If $B \subseteq B_j$ satisfies*

$$\mathrm{fa}(\Delta_j) \perp_{\mathcal{L}} (\Upsilon \cap \mathrm{de}(d_j)) \mid (B \cup \{d_j\}), \tag{10}$$

then the following statements hold:

1. *$\mathcal{L}_{B \to d_j}$ has partial solution ordering $d_j, \ldots, d_k$.*
2. *B contains the set of requisite parents of d_j in $\mathcal{L}_{B \to d_j}$.*

On the other hand, if (10) does not hold, then at least one of the above statements does not hold.

In Appendix B, the correctness of Theorem 6 and Theorem 7 is given. As a corollary to Theorem 6 and 7 we obtain:

Corollary 2 *The set N_j given by*

$$N_j = \operatorname*{arg\,min}_{B : B \subseteq (B_j \cap \mathrm{nd}(d_j))} \{|B| \mid \mathrm{fa}(\Delta_j) \perp_{\mathcal{L}} (\Upsilon \cap \mathrm{de}(d_j)) \mid (B \cup \{d_j\})\}. \tag{11}$$

satisfies the equation in (8).

As shown below, the subset N_j in (11) is identical to a a minimum separating set in a suitable undirected graph. Here, for an the undirected graph $\mathcal{G}$, we let the symbolic expression $A \perp_{\mathcal{G}} B \mid S$ denote that A and B are separated by S in $\mathcal{G}$.

Theorem 8 *Let A, B, C and N be subsets of nodes in $\mathcal{L}$, and let $\mathcal{G} = \left(\mathcal{L}_{\mathrm{An}(A \cup B \cup C)}\right)^m$. Then,*

$$\begin{aligned} N' &= \operatorname*{arg\,min}_{M : M \subseteq N} \left\{|M| \mid A \perp_{\mathcal{L}} B \mid (M \cup C)\right\} \quad \Leftrightarrow & (12) \\ N' &= \operatorname*{arg\,min}_{M : M \subseteq N} \left\{|M| \mid A \perp_{\mathcal{G}} B \mid (M \cup C)\right\}. & (13) \end{aligned}$$

Theorem 8 is proven in Appendix C. By transforming the problem of finding the subset in (11) into a problem of finding a minimum separating set in an undirected graph, we can exploit the well-known and efficient max-flow min-cut algorithm for this task. The interested reader is referred to Chartrand and Oellermann (1993).

Finally, the full procedure is shown below.

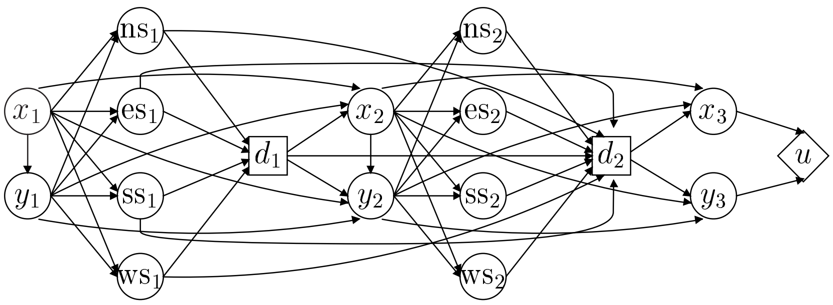


Figure 14: LIMID representing the maze problem with only two decisions. In the LIMID, all informational arcs are requisite.

> **Input:** LIMID $\mathcal{L}^{k+1}$ with decisions $d_1, \ldots, d_k$ whose ordering satisfies (7).
>
> **Output:** LIMID $\mathcal{L}^1$ with exact solution ordering $d_1, \ldots, d_k$; upper bound $\mathrm{EU}(\mathcal{L}^1)$ of $\mathrm{EU}(\mathcal{L}^{k+1})$.
>
> For $j = k \ldots 1$ do:
>
> 1. $\mathcal{L} \leftarrow \mathcal{L}^{j+1}$.
> 2. Compute the set B_j as in (9).
> 3. Compute the set N_j as in (11).
> 4. $\mathcal{L}^j \leftarrow (\mathcal{L}^{j+1})_{N_j \to d_j}$.
> 5. $\mathcal{L}^j \leftarrow (\mathcal{L}^j)_{\min}$.
>
> Evaluate $\mathcal{L}^1$ using SPU to obtain $\mathrm{EU}(\mathcal{L}^1)$.

Procedure 4 *A procedure for finding a 'low complexity' upper bound of $\mathcal{L}$.*

Note that all computations apart from last step are simple graph manipulations on the initial LIMID.

Example 5 (Procedure 4 on Maze Problem) To illustrate Procedure 4, consider the LIMID in Figure 14, representing the maze problem with only two decisions. We denote the LIMID $\mathcal{L}^3$.

The procedure starts by inserting fill-in arcs into the last decision d_2 in the following way: First, we find the sets B_2 and N_2 as defined in (9) and (11). By definition $B_2 = V$, and N_2 is the subset of $B_2 \cap \mathrm{nd}(d_2)$ with smallest cardinality, so that $N_2 \cup \{d_2\}$ d-separates u from $\mathrm{fa}(\Delta_2)$. It can be seen that $N_2 = \{x_2, y_2\}$, and therefore,

$$\mathcal{L}^2 \quad = \quad (\mathcal{L}^3)_{\{x_2,y_2\} \to d_2}$$

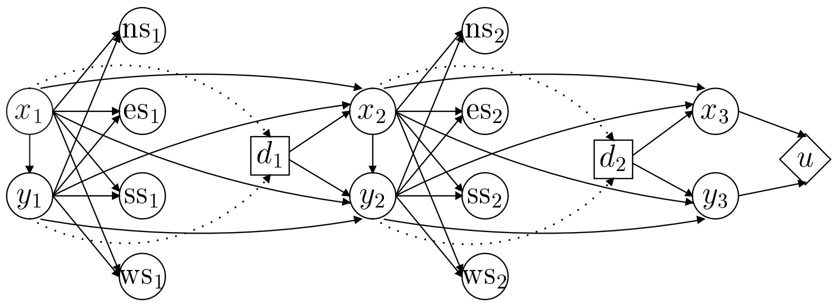


Figure 15: Minimal reduction of the LIMID $\mathcal{L}^1$. Dotted arcs are fill-in arcs.

Then, non-requisite arcs in $\mathcal{L}^2$ are removed. In $\mathcal{L}^2$, all parents of d_1 are requisite, and all parents of d_2, except $\{x_2, y_2\}$, are non-requisite. Abbreviating $O_i = \{\mathrm{ns}_i, \mathrm{ss}_i \mathrm{ws}_i \mathrm{es}_i\}$ we thus obtain:

$$\begin{aligned} \mathcal{L}^2 &\leftarrow (\mathcal{L}^2)_{\min} \\ &= (\mathcal{L}^2)_{(O_1 \cup O_2 \cup \{d_1\}) \not\to d_2}. \end{aligned}$$

Next, fill-in arcs into d_1 are inserted. The sets B_1 and N_1 can be found to be:

$$\begin{aligned} B_1 &= O_1 \cup O_2 \cup \{d_1, x_1, y_1\} \\ &\qquad \text{because } (O_1 \cup O_2 \cup \{d_1, x_1, y_1\}) \text{ is d-sep. from } u \text{ given } \mathrm{fa}(d_2) \\ N_1 &= \{x_1, y_1\}. \end{aligned}$$

Therefore,

$$\mathcal{L}^1 = (\mathcal{L}^2)_{\{x_1, y_1\} \to d_1}.$$

The minimal reduction $(\mathcal{L}^1)_{\min}$ is shown in Figure 15. Note, in $(\mathcal{L}^1)_{\min}$, there are only 2 informational arcs into each of the decision nodes d_1 and d_2. By comparison, in the original LIMID $\mathcal{L}^3$ (which is minimal), there are 4 and 9 informational arcs respectively.

In the maze problem with 10 decision nodes, the number of requisite parents for decision d_i is $5 \cdot (i-1) + 4$. However, by applying Procedure 4, we obtain a soluble cover of the LIMID where each decision node d_i only has 2 requisite parents (x_i and y_i).

Example 6 (Upper bounds on Maze Problem) We now apply Procedure 4, to compute an upper bound of the Maze-navigation problem with 10 stages. As indicated in the previous example, the LIMID $\mathcal{L}^1$, obtained by applying Procedure 4 on $\mathcal{L}$, has the form:

$$\mathcal{L}^1 = (((\mathcal{L}_0)_{\{x_1, y_1\} \to d_1}) \cdots)_{\{x_{10}, y_{10}\} \to d_{10}},$$

where $\mathcal{L}_0$ is the LIMID obtained by removing all informational arcs from $\mathcal{L}$.

So, $\mathcal{L}^1$ corresponds to the situation in which the Robot always knows its true position. This LIMID is tractable, and the following expected utility values are obtained for the different setups:

	Maze1	**Maze2**	**Maze3**
$\mathrm{EU}(\mathcal{L}^1)$	1	1	0.94078

The 0.94078 bound for the **Maze3** setup, shows that even though the robot knows its true position at all stages (which it does in $\mathcal{L}^1$), the noisy actor makes it impossible to reach the goal state with certainty. To compare, in Example 4, we found, for **Maze3**, a lower bound strategy with expected utility 0.897.

6.2 Improving upper bounds

Suppose we apply Procedure 4 on a LIMID $\mathcal{L}$ to obtain $\mathcal{L}^1$. Thus, $\mathrm{EU}(\mathcal{L}^1)$ upper bounds $\mathrm{EU}(\mathcal{L})$. Typically, $\mathrm{EU}(\mathcal{L}^1)$ is easy to compute because, by construction, $\mathcal{L}^1$ has few requisite informational arcs. Indeed, this is the intention of the procedure. However, because we only focus on constructing a LIMID $\mathcal{L}^1$ having few requisite parents, $\mathrm{EU}(\mathcal{L}^1)$ may be a poor upper bound for $\mathrm{EU}(\mathcal{L})$.

In this subsection, we provide a method for improving the upper bound $\mathrm{EU}(\mathcal{L}^1)$. The procedure starts from $\mathcal{L}^1$, and generate LIMIDs $\mathcal{L}^2, \mathcal{L}^3, \ldots$, so that each $\mathrm{EU}(\mathcal{L}^i)$ upper bounds $\mathrm{EU}(\mathcal{L})$ and

$$\mathrm{EU}(\mathcal{L}^1) = \mathrm{EU}(\mathcal{L}^2) \geq \mathrm{EU}(\mathcal{L}^3) \geq \mathrm{EU}(\mathcal{L}^4) \cdots .$$

Initially, we construct a LIMID $\mathcal{L}^2$ that is a soluble covering of $\mathcal{L}$. This is achieved in two steps:

Firstly, we augment $\mathcal{L}^1$ with the informational arcs in $\mathcal{L}$ that has been removed during the construction of $\mathcal{L}^1$ because they were non-requisite. So, the obtained LIMID $\tilde{\mathcal{L}}^1$ has the form:

$$\tilde{\mathcal{L}}^1 = ((\mathcal{L}^1)_{\mathrm{pa}_{\mathcal{L}}(d_1)\to d_1} \cdots)_{\mathrm{pa}_{\mathcal{L}}(d_k)\to d_k}.$$

Secondly, we obtain $\mathcal{L}^2$ by augmenting $\tilde{\mathcal{L}}^1$ with fill-in arcs into each decision node from previous families. So,

$$\mathcal{L}^2 = ((\tilde{\mathcal{L}}^1)_{\mathrm{fa}(\Delta_1)\to d_2} \cdots)_{\mathrm{fa}(\Delta_{k-1})\to d_k}.$$

None of the inserted informational arcs affect the maximal expected utility, i.e. $\mathrm{EU}(\mathcal{L}^1) = \mathrm{EU}(\mathcal{L}^2)$. In short, the first step does not affect the expected utility because of the way $\mathcal{L}^1$ is constructed and Lemma 3 in Appendix A. The following theorem shows why the second step does not affect the expected utility. The theorem is proved in Appendix D.

Theorem 9 *Suppose $\mathcal{L}$ is soluble with exact solution order $d_1, \ldots, d_k$, and assume the ordering of the decision nodes satisfies (7). If*

$$\mathcal{L}' = ((\mathcal{L}_{\mathrm{fa}(\Delta_1)\to d_2})\cdots)_{\mathrm{fa}(\Delta_{k-1})\to d_k},$$

then $\mathrm{EU}(\mathcal{L}) = \mathrm{EU}(\mathcal{L}')$.

In words, the theorem says that, under suitable conditions, the insertion of fill-in arcs in $\mathcal{L}$ from previous families does not affect the expected utility of the LIMID.

The full procedure is shown below. Recall, the notation $\mathcal{L}_{a\not\to d}$ denotes the LIMID $\mathcal{L}$, in which the arc from $a \in \mathrm{pa}(d)$ into d is removed.

Input: LIMID $\mathcal{L}$ with decision nodes $d_1, \ldots, d_k$ whose ordering obey (7).

Output: Soluble cover $\mathcal{L}^j$ of $\mathcal{L}$; $\mathrm{EU}(\mathcal{L}^j)$.

Initialization: $j \leftarrow 1$.

$\mathcal{L}^1$ obtained by applying Procedure 4 on $\mathcal{L}$;

$\tilde{\mathcal{L}}^1 \leftarrow ((\mathcal{L}^1)_{\mathrm{pa}_{\mathcal{L}}(\Delta_1)\to d_2}\cdots)_{\mathrm{pa}_{\mathcal{L}}(\Delta_{k-1})\to d_k)}$;

$\mathcal{L}^2 \leftarrow ((\tilde{\mathcal{L}}^1)_{\mathrm{fa}(\Delta_1)\to d_2}\cdots)_{\mathrm{fa}(\Delta_{k-1})\to d_k)}$.

Repeat (until stopping criteria are met)

For $i = k$ to 1 do

1. (Define candidates) $C_i \leftarrow \mathrm{pa}_{\mathcal{L}^j}(d_i) \setminus (\mathrm{fa}_{\mathcal{L}^j}(\Delta_{i-1}) \cup \mathrm{pa}_{\mathcal{L}}(d_i))$
2. If $C_i \neq \emptyset$ do:
   $a \leftarrow$arbitrary node in C_i.
   $\mathcal{L}^{j+1} \leftarrow (\mathcal{L}^j)_{a\not\to d_i}$
   $\mathrm{EU}(\mathcal{L}^{j+1}) \leftarrow$maximal expected utility of $(\mathcal{L}^{j+1})_{\min}$ obtained by SPU.
3. If $C_i = \emptyset$ do: $\mathcal{L}^{j+1} \leftarrow \mathcal{L}^j$.
4. $j \leftarrow j + 1$.

Return $\mathcal{L}^j$; $\mathrm{EU}(\mathcal{L}^j)$.

Procedure 5 *An anytime algorithm for generating upper bounds with higher quality.*

The above procedure has the following properties:

Theorem 10 *Suppose we apply Procedure 5 on a LIMID $\mathcal{L}$. Then, the following statements hold for $j = 2, 3, \ldots$:*

1. **(Iterative Improvement)** $\mathrm{EU}(\mathcal{L}^j) \geq \mathrm{EU}(\mathcal{L}^{j+1})$.
2. **(Soluble)** *$\mathcal{L}^j$ is soluble with exact solution ordering $d_1, \ldots, d_k$.*
3. **(Covering)** *$\mathcal{L}^j$ is a covering of $\mathcal{L}$.*

Proof: The first property follows trivially since $\mathcal{L}^j \supseteq \mathcal{L}^{j+1}$. To prove the second property, we note

$$\mathrm{fa}_{\mathcal{L}^j}(\Delta_{i-1}) \subseteq \mathrm{fa}_{\mathcal{L}^j}(d_i),$$

and the result immediately follows from the definition of solubility.

Finally, for the last property, we note that $\mathcal{L}^2 \supseteq \mathcal{L}$, and the procedure does not remove any arcs that are present in $\mathcal{L}$, i.e. $\mathcal{L}^j \supseteq \mathcal{L}$. •

Example 7 (Improve the upper bounds on Maze Problem) Returning to the maze problem, we use Procedure 5 to iteratively improve the upper bound obtained in Example 6. As shown in Example 4, the optimal strategy in the case of **Maze1** has an expected utility of 1, which coincides with the found upper bound. Therefore no iterative improvement is necessary in this case. Figure 16 shows the results for **Maze3**.

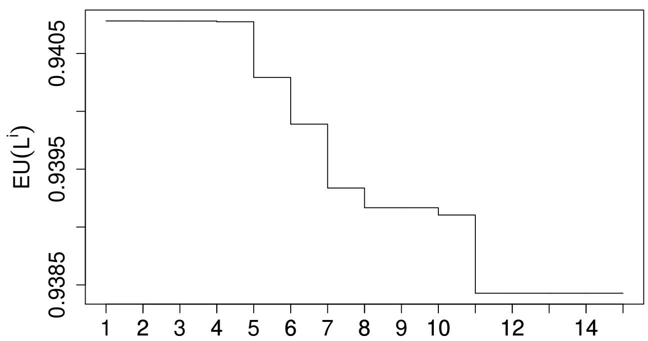


Figure 16: Results obtained by running the iterative improvement procedure on the **Maze3** problem.

For each of the three different setups, the upper bounds associated with the final LIMID is presented below:

	Maze1	**Maze2**	**Maze3**
EU($\mathcal{L}^{14}$)	1	0.999999	0.938428

From this we can deduce that for the case of **Maze 2**, the optimal strategy lies between 0.998685 and 0.999999, i.e. our lower bound strategy is a very good approximation to the optimal strategy. In **Maze 3**, the range is 0.897123 - 0.938428, i.e. in this case there is still room for improvement.

7 Conclusion

Because calculating optimal strategies is intractable for most real-world decision problems, the algorithms must make some approximations and assumptions. The approach we take in this paper is based on LIMIDs, and the evaluation of limited memory strategies. As described in the present paper and in Lauritzen and Nilsson (2001), there are good arguments for using limited memory strategies.

In this paper, we provide heuristic algorithms for searching in the huge space of limited memory strategies. A special focus is on POMDPs. There are two reasons for this. Firstly, despite its simplicity, POMDPs are highly intractable, and thus we must rely on approximate solutions. Secondly, limited memory strategies fit naturally into the POMDP framework because of the sequential nature of POMDPs: The computation of lower and upper bounds for the POMDP case are straightforward and intuitive. In addition, as illustrated in the Robot example, the results are very promising.

7.1 Performance of the algorithms

The limited memory strategies we compute can be compared with the associated upper bounds computed for the same problem. We assess the performance of our algorithms by running them on two multistage decision problems: A Robot problem and a Maze problem. Here, we briefly present the results for the latter.

In the Maze problem, the agent (robot) should reach a goal state within ten steps. Three different setups, with increasing difficulty, were evaluated:

Maze1	perfect sensors/perfect actor
Maze2	noisy sensors/perfect actor
Maze3	noisy sensors/noisy actor

In **Maze1**, the obtained strategy reaches the goal state always. For the **Maze2** problem, the computed limited memory strategy reaches the goal with probability 99.987%. The **Maze3** problem is the most challenging problem. In this case, the obtained limited memory strategy reaches the goal state with probability 89.71%. Again, the optimal strategy is intractable, but by running our upper bound strategies on the problem, we know that the optimal strategy is bounded from above with 93.84%.

To summarize, the performance of our our lower bound strategies is as follows:

Maze1	**Maze2**	**Maze3**
100%	99.987%	89.71%

The Maze problem (and variants thereof) is well studied in the literature. The model we use is borrowed from Horsch and Poole (1998). In this paper, the following results were obtained:

Maze1	**Maze2**	**Maze3**
86.96%	77.67%	70.45%

Even though, direct comparison is not possible because the obtained results depends on criteria such as implementation efficiency and evaluation time, it indicates that our lower bound strategies perform well. This conclusion is supported by the obtained upper bounds:

Maze1	**Maze2**	**Maze3**
1	0.999999	0.938428

7.2 Future work

This work opens the door to a variety of possible extensions. The general algorithm for computing lower and upper bounds works well on some problems. However, because the algorithm are designed for general LIMIDs, it should be possible to improve the performance of the algorithms on special domains. Our algorithms for POMDP problems demonstrate this.

In game theory, a new representation termed Multi Agent Influence Diagrams (MAIDs) have been proposed by Koller and Brian (2001). The computation of a Nash equilibrium is arguably the key computational task in game theory. The notion of MAIDs is very similar to LIMIDs, and by using the algorithms presented in this paper, it may be possible to compute bounds on Nash equilibria.

7.3 Implementation issues

A C++ implementation is created for the lower and upper bound algorithms. The implementation uses the Hugin API (Hugin Expert A/S 2000) for Bayesian network inference and the LEDA Platform (Mehlhorn and Naher 2000) for graph theoretic calculations.

Acknowledgements

This research was supported by DINA (Danish Informatics Network in the Agricultural Sciences), funded by the Danish Research Councils through their PIFT programme.

Appendix A

To prove Theorem 3, we need the following two results.

The first result we shall need was shown by Lauritzen *et al.* (1990), and says that the d-separation criterion for a DAG is equivalent to separation in a suitable undirected graph:[4]

Proposition 1 *Let A, B and S be subsets of nodes in $\mathcal{L}$. Then $A \perp_{\mathcal{L}} B \mid S$ if and only if S separates A from B in $(\mathcal{L}_{\mathrm{An}(A\cup B\cup S)})^m$.*

Here, $(\mathcal{L}_{\mathrm{An}(A\cup B\cup S)})^m$ denotes the moral graph obtained by first restricting $\mathcal{L}$ to involve only nodes in $A \cup B \cup S$ and their ancestors.

The second result says that, under suitable conditions, d-separation of two node-sets A and B by a third node-set S, continues to apply in a LIMID of the form $\mathcal{L}' = \mathcal{L}_{N\to b}$. Here, we let $V_{\mathcal{G}}$ and $E_{\mathcal{G}}$ denote the node-set and edge-set respectively for an undirected graph $\mathcal{G}$.

Lemma 3 *Suppose $A \perp_{\mathcal{L}} B \mid S$ for three subsets A, B, S in a LIMID $\mathcal{L}$. Assume N is a subset of nodes in $\mathcal{L}$, and b is a node in $\mathcal{L}$. Let $\mathcal{L}' = \mathcal{L}_{N\to b}$. If $(N \cup \{b\}) \subseteq A \cup S$, then $A \perp_{\mathcal{L}'} B \mid S$.*

Proof: Without loss of generality, we assume that $N \subseteq \mathrm{nd}(b)$. Let $\mathcal{G}$ and $\mathcal{G}'$ be two undirected graphs given by

$$\begin{aligned}\mathcal{G} &= (\mathcal{L}_{\mathrm{An}(A\cup B\cup S)})^m \\ \mathcal{G}' &= (\mathcal{L}'_{\mathrm{An}(A\cup B\cup S)})^m.\end{aligned}$$

[4] In the original proposition of Lauritzen *et al.* (1990), it is assumed that the subsets A, B and S are disjoint. This restriction is not needed.

Further, let $\mathcal{G}''$ be the undirected graph obtained by augmenting $\mathcal{G}$ with edges between any two edges in $A \cup S$. Note that $V_{\mathcal{G}} = V_{\mathcal{G}'} = V_{\mathcal{G}''}$, and $E_{\mathcal{G}} \subseteq E_{\mathcal{G}'} \subseteq E_{\mathcal{G}''}$.

By Proposition 1, A and B are separated by S in $\mathcal{G}$. It can be seen that this separation continues to apply in $\mathcal{G}''$, since the addition of edges between any two nodes in $A \cup S$ cannot create a path from A to B that is not separated by S. Since $E_{\mathcal{G}'} \subseteq E_{\mathcal{G}''}$, it follows that A and B are separated by S in $\mathcal{G}'$. The result now follows by Proposition 1. •

Finally, we have

Proof (Theorem 3): Suppose (6) holds. By Lemma 3,

$$n \perp_{\mathcal{L}'} (\Upsilon \cap \mathrm{de}(d)) \mid (\mathrm{fa}(d) \setminus \{n\}), \tag{14}$$

i.e. n is non-requisite for d in $\mathcal{L}'$.

To prove the converse statement, suppose n is non-requisite for d in $\mathcal{L}'$. So, by definition, (14) holds. Now, (6) follows because d-separation is preserved under arc-removal. •

Appendix B

Proof (Theorem 6) Let i be an arbitrary element in $\{j+1, \ldots, k\}$, and let $\mathcal{L}' = \mathcal{L}_{B \to d_j}$. If $\mathcal{L}$ has partial solution ordering $d_{j+1}, \ldots, d_k$, then we have

$$\mathrm{fa}(\Delta_{i-1}) \perp_{\mathcal{L}} (\Upsilon \cap \mathrm{de}(d_i)) \mid \mathrm{fa}(d_i). \tag{15}$$

Suppose $B \subseteq B_j$. By (9), we have that

$$B \perp_{\mathcal{L}} (\Upsilon \cap \mathrm{de}(d_i)) \mid \mathrm{fa}(d_i).$$

Thus,

$$(\mathrm{fa}(\Delta_{i-1}) \cup B) \perp_{\mathcal{L}} (\Upsilon \cap \mathrm{de}(d_i)) \mid \mathrm{fa}(d_i).$$

By (7), $\mathrm{de}_{\mathcal{L}}(d_i) = \mathrm{de}_{\mathcal{L}'}(d_i)$. Further, $(B \cup \mathrm{fa}_{\mathcal{L}}(\Delta_{i-1})) = \mathrm{fa}_{\mathcal{L}'}(\Delta_{i-1})$ and $\mathrm{fa}_{\mathcal{L}}(d_i) = \mathrm{fa}_{\mathcal{L}'}(d_i)$, i. e.

$$\mathrm{fa}_{\mathcal{L}'}(\Delta_{i-1}) \perp_{\mathcal{L}} (\Upsilon \cap \mathrm{de}_{\mathcal{L}'}(d_i)) \mid \mathrm{fa}_{\mathcal{L}'}(d_i).$$

From Lemma 3, it now follows that

$$\mathrm{fa}_{\mathcal{L}'}(\Delta_{i-1}) \perp_{\mathcal{L}'} (\Upsilon \cap \mathrm{de}_{\mathcal{L}'}(d_i)) \mid \mathrm{fa}_{\mathcal{L}'}(d_i),$$

i. e. $\mathcal{L}'$ has partial solution ordering $d_{j+1}, \ldots, d_k$.

Conversely, suppose $B \not\subseteq B_j$. By (9), we have that

$$B \not\perp_{\mathcal{L}} (\Upsilon \cap \mathrm{de}(d_i)) \mid \mathrm{fa}(d_i).$$

Because d-separation is preserved under arc-removal, d-connection is preserved under arc-insertion, and thus

$$B \not\perp_{\mathcal{L}'} (\Upsilon \cap \mathrm{de}(d_i)) \mid \mathrm{fa}(d_i).$$

Because $B \subseteq \mathrm{fa}_{\mathcal{L}'}(d_j) \subseteq \mathrm{fa}_{\mathcal{L}'}(\Delta_{i-1})$, it follows that, in $\mathcal{L}'$, d_i is not extremal when $d_{i+1}, \ldots, d_k$ are converted into chance nodes. It follows that $\mathcal{L}'$ has not partial solution ordering $d_{j+1}, \ldots, d_k$. This completes thte proof. •

Proof (Theorem 7) Suppose $B \subseteq B_j$, and let $\mathcal{L}' = \mathcal{L}_{B \to d_j}$. Further, assume (10) holds. By (10), we have

$$\mathrm{fa}(\Delta_{j-1}) \perp_{\mathcal{L}} (\Upsilon \cap \mathrm{de}(d_j)) \mid (B \cup \{d_j\}).$$

Because $\mathrm{fa}_{\mathcal{L}}(\Delta_{j-1}) = \mathrm{fa}_{\mathcal{L}'}(\Delta_{j-1})$, and $\mathrm{de}_{\mathcal{L}}(d_j) = \mathrm{de}_{\mathcal{L}'}(d_j)$ we have from Lemma 3

$$\mathrm{fa}(\Delta_{j-1}) \perp_{\mathcal{L}'} (\Upsilon \cap \mathrm{de}(d_j)) \mid (B \cup \{d_j\}). \tag{16}$$

By (16), d_j is extremal in $\mathcal{L}'$ when $d_{j+1}, \ldots, d_k$ are converted into chance nodes. Further, by Theorem 6, $\mathcal{L}'$ has partial solution ordering $d_{j+1}, \ldots, d_k$. We conclude that $\mathcal{L}'$ has partial solution ordering $d_j, \ldots, d_k$ which proved the first statement. To prove the second statement, we note that (10) and Lemma 3 imply that

$$(\mathrm{fa}(d_j) \setminus B) \perp_{\mathcal{L}'} (\Upsilon \cap \mathrm{de}(d_j)) \mid (B \cup \{d_j\}), \tag{17}$$

i.e. $\mathrm{pa}_{\mathcal{L}'}(d_j) \setminus B$ are non-requisite for d_j in $\mathcal{L}'$ which proved the second statement.

Conversely, suppose the two statements in the theorem hold, i.e. we have (respectively)

$$\begin{gathered}\mathrm{fa}(\Delta_{j-1}) \perp_{\mathcal{L}'} (\Upsilon \cap \mathrm{de}(d_j)) \mid (B \cup \{d_j\}) \\ (\mathrm{fa}(d_j) \setminus B) \perp_{\mathcal{L}'} (\Upsilon \cap \mathrm{de}(d_j)) \mid (B \cup \{d_j\}).\end{gathered}$$

Clearly, $B \perp_{\mathcal{L}'} (\Upsilon \cap \mathrm{de}(d_j)) \mid (B \cup \{d_j\})$ and by a simple application of the graphoid axioms, it can be seen that

$$\mathrm{fa}(\Delta_j) \perp_{\mathcal{L}'} (\Upsilon \cap \mathrm{de}(d_j)) \mid (B \cup \{d_j\})$$

Now, (10) follows because $\mathrm{fa}_{\mathcal{L}'}(d_j) = \mathrm{fa}_{\mathcal{L}}(d_j)$, $\mathrm{de}_{\mathcal{L}'}(d_j) = \mathrm{de}_{\mathcal{L}}(d_j)$, and the fact that d-separation is preserved under arc-removal (since $\mathcal{L} \subseteq \mathcal{L}'$).

•

Appendix C

For two undirected graphs, we shall say that $\mathcal{G}'$ is a subgraph of $\mathcal{G}$, whenever $V_{\mathcal{G}} = V_{\mathcal{G}'}$ and $E_{\mathcal{G}} \supseteq E_{\mathcal{G}'}$, i.e. $\mathcal{G}$ and $\mathcal{G}'$ have the same node-sets, and every edge in $\mathcal{G}'$ is also present in $\mathcal{G}$.

Proof: (Theorem 8) Let N' be given as in (12), and let

$$\mathcal{G}' = \left(\mathcal{L}_{\mathrm{An}(A\cup B\cup C\cup N')}\right)^{\mathrm{m}}.$$

By Proposition 1, $A \perp_{\mathcal{G}'} B \mid (N' \cup C)$. Since $\mathcal{G}$, as defined in the theorem, is a subgraph of $\mathcal{G}'$, $V_{\mathcal{G}} \cap (N' \cup C)$ separates A from B in $\mathcal{G}$. Thus, by Proposition 1,

$$A \perp_{\mathcal{L}} B \mid (V_{\mathcal{G}} \cap (N' \cup C)).$$

Because N' is a minimal subset that satisfies (12), it follows that $N' \subseteq V_{\mathcal{G}}$. Now, (13) follows from Proposition 1.

To prove the converse statement, let N' be given as in (13). By Proposition 1, $A \perp_{\mathcal{L}} B \mid (N' \cup C)$. Suppose $A \perp_{\mathcal{L}} B \mid (N^* \cup C)$, for some subset N^* of nodes in $\mathcal{L}$. By Proposition 1, $A \perp_{\mathcal{G}^*} B \mid N^* \cup C$, where $\mathcal{G}^* = \left(\mathcal{L}_{\mathrm{An}(A\cup B\cup C\cup N^*)}\right)^{\mathrm{m}}$. Since $\mathcal{G}$, as defined in the theorem, is a subgraph of $\mathcal{G}^*$, $(C \cup N^*) \cap V_{\mathcal{G}}$ separates A from B in $\mathcal{G}$. Thus, by (13),

$$\mid N' \mid \quad \leq \quad \mid N^* \cap V_{\mathcal{G}} \mid \quad \leq \quad \mid N^* \mid,$$

which proves that N' satisfies (12). •

Appendix D

Proof (Theorem 9) Since $\mathcal{L}$ is soluble with exact solution order $d_1, \ldots, d_k$, we have for all $j = 2, \ldots, k$:

$$\mathrm{fa}(\Delta_{j-1}) \perp_{\mathcal{L}} (\Upsilon \cap \mathrm{de}(d_j)) \mid \mathrm{fa}(d_j). \tag{18}$$

Let A_j be the parents of d_j in $\mathcal{L}'$ that are not parents of d_j in $\mathcal{L}$, i.e.

$$A_j = \mathrm{pa}_{\mathcal{L}'}(d_j) \setminus \mathrm{pa}_{\mathcal{L}}(d_j).$$

Consider the LIMID $\mathcal{L}^j$ defined by

$$\mathcal{L}^j = ((\mathcal{L}_{A_2 \to d_2}) \cdots)_{A_j \to d_j}.$$

The result follows if we can show that for all $j = k, \ldots, 2$, the nodes in A_j are non-requisite for d_j in $\mathcal{L}^j$. The proof is by induction. Note that

$$A_1 \cup \cdots \cup A_j \subseteq \mathrm{fa}_{\mathcal{L}}(\Delta_{j-1}), \tag{19}$$

and by (7),

$$\mathrm{de}_{\mathcal{L}}(d_j) = \mathrm{de}_{\mathcal{L}^j}(d_j). \tag{20}$$

and consider the cases:

$j = k$: From (18), (19), (20) and $k-1$ successive use of Lemma 3, it follows that, in $\mathcal{L}^k$, A_k is d-separated from $(\Upsilon \cap \mathrm{de}(d_k))$ given $\mathrm{fa}(d_k)$. Now, the result follows.

$j < k$: Similarly, the result follows from (18), (19), (20), and $j-1$ successive use of Lemma 3.

This completes the proof. •

References

Chartrand, G. and Oellermann, O. R. (1993). *Applied and Algorithmic Graph Theory*. McGraw-Hill Book Co., Singapore.

Horsch, M. C. and Poole, D. (1998). An anytime algorithm for decision making under uncertainty. In *Proceedings of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–98)*, pp. 246–55. Morgan Kaufmann Publishers, San Francisco, CA.

Hugin Expert A/S (2000). *Hugin API 4.2 Reference Manual*. http://www.hugin.dk/ftp/doc/api-manual.ps.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, **101**, (1–2), 99–134.

Koller, D. and Brian, M. (2001). Multi-agent diagrams for representing and solving games. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-2001)*, pp. 1027–34. Morgan Kaufmann Publishers, San Francisco, CA.

Lauritzen, S. L., Dawid, A. P., Larsen, B. N., and Leimer, H.-G. (1990). Independence properties of directed Markov fields. *Networks*, **20**, 491–505.

Lauritzen, S. L. and Nilsson, D. (2001). LIMIDs of Decision Problems. *To appear in Management Science.*

Littman, M. L., Cassandra, A. R., and Kaelbling, L. P. (1995). Learning policies for partially observable environments: Scaling up. In *Proceedings of the Twelfth International Conference on Machine Learning*, (ed. A. Prieditis and S. Russell), pp. 362–70. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, San Francisco, CA.

Lovejoy, W. S. (1991). A survey of algorithmic methods for partially observed Markov decision processes. *Annals of Operations Research*, **28**, (1–4), 47–66.

Madsen, A. and Nilsson, D. (2001). Solving Influence Diagrams using HUGIN, shafer-shenoy, and lazy propagation. In *Proceedings of the 17th Annual Conference on Uncertainty in Artificial Inteligence*, (ed. J. Breese and D. Koller), pp. 337–45. Morgan Kaufmann Publishers, San Francisco, California.

Mehlhorn, K. and Naher, S. (2000). *Leda : A Platform for Combinatorial and Geometric Computing*. Cambridge University Press.

Nielsen, T. D. and Jensen, F. V. (1999). Welldefined decision scenarios. In *Proceedings of the Fifteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–99)*, pp. 502–11. Morgan Kaufmann Publishers, San Francisco, CA.

Nilsson, D. and Lauritzen, S. (2000). Evaluating influence diagrams using LIMIDs. In *Uncertainty in Artificial Intelligence: Proceedings of the Sixteenth Conference (UAI-2000)*, pp. 436–45. Morgan Kaufmann Publishers, San Francisco, CA.

Shachter, R. (1998). Bayes-ball: The rational pasttime (for determining irrelevance and requisite information in belief networks and influence diagrams). In *Proceedings of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–98)*, pp. 48–487. Morgan Kaufmann Publishers, San Francisco, CA.