Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16)

A POMDP Approach to Influence Diagram Evaluation

Eric A. Hansen, Jinchuan Shi, and **Arindam Khaled**
Dept. of Computer Science and Engineering
Mississippi State University
Mississippi State, MS 39762
hansen@cse.msstate.edu, jinchuanshi86@gmail.com, arindamkhaled@gmail.com

Abstract

We propose a node-removal/arc-reversal algorithm for influence diagram evaluation that includes reductions that allow an influence diagram to be solved by a generalization of the dynamic programming approach to solving partially observable Markov decision processes (POMDPs). Among its potential advantages, the algorithm allows a more flexible ordering of node removals, and a POMDP-inspired approach to optimizing over hidden state variables, which can improve the scalability of influence diagram evaluation in solving complex, multi-stage problems. It also finds a more compact representation of an optimal strategy.

1 Introduction

An influence diagram (ID) is a compact graphical model of a decision-making problem under uncertainty [Howard and Matheson, 1981]. Originally introduced as a front end for decision trees that can facilitate analysis of complex decision problems, IDs have since proven to be a useful framework for developing efficient solution algorithms. The class of problems represented by influence diagrams is very closely related to the class of finite-horizon partially observable Markov decision processes (POMDPs) [Monahan, 1982; Kaelbling *et al.*, 1998]. Interestingly, however, very different algorithms have been developed for these two models.

Algorithms for solving IDs can be classified as indirect and direct methods. Indirect methods convert an ID to a secondary structure before solving it, which can be a decision tree that is solved by backwards induction or branch-and-bound search [Howard and Matheson, 1981; Qi and Poole, 1995; Yuan *et al.*, 2010], a belief network that is solved by probabilistic inference techniques [Cooper, 1988; Shachter and Peot, 1992; Zhang, 1998], or a junction tree that is solved by message-passing techniques [Jensen *et al.*, 1994]. Direct methods solve an ID in the course of performing a sequence of value-preserving transformations on the ID itself, which may involve removing one node at a time from the network [Olmsted, 1983; Shachter, 1986; Tatman and Shachter, 1990], or one variable at a time from an equivalent mathematical formula [Dechter, 2000].

All of these different algorithms for ID evaluation represent a solution, or *strategy*, in the same way – as a mapping from a history of actions and observations to the choice of action, with the strategy conditioned on an initial belief in the form of a prior probability distribution over a state space. For the related model of finite-horizon POMDPs, however, a solution is often represented differently; at each stage, it is represented as a mapping from belief states to the choice of action, where a belief state is a vector of posterior probabilities over a state space. Especially for problems with a horizon that is longer than a couple of stages, the belief-based representation of a strategy can be more compact, and sometimes much more compact, than a history-based representation.

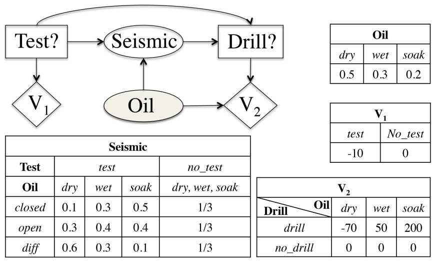
In this paper, we describe an algorithm for solving IDs that leverages this alternative representation of a strategy, and the related approach to dynamic programming originally developed for solving POMDPs. The new algorithm can be viewed as an extension of the classic node-removal/arc-reversal algorithm for ID evaluation that includes reductions that generalize the incremental pruning algorithm for POMDPs [Cassandra *et al.*, 1997]. It can also be viewed as a generalization of the incremental pruning algorithm that solves any regular and no-forgetting ID, and not just the special case of IDs that correspond to finite-horizon POMDPs.

We show that this approach to solving IDs has some potential advantages, especially in solving complex, multistage problems. Among them, it allows a more flexible order of node reductions, and it makes it possible to optimize over unobserved state variables in a POMDP-inspired way, which can improve the scalability of ID evaluation. It also finds a more compact representation of an optimal strategy, which can be easier to interpret.

The approach we develop also contributes to a better understanding of the relationship between IDs and finite-horizon POMDPs. Previous work has shown how to use techniques developed for solving IDs in order to solve POMDPs more efficiently, especially factored POMDPs [Boutilier and Poole, 1996; Hansen and Feng, 2000]. As far as we know, we are the first to propose using techniques developed for solving POMDPs in order to solve IDs more efficiently.

2 Influence diagrams and POMDPs

We begin with a brief review of relevant background about influence diagrams and finite-horizon POMDPs.



Oil		
dry	*wet*	*soak*
0.5	0.3	0.2

V1	
test	*No_test*
-10	0

Seismic				
Test	*test*			*no_test*
Oil	*dry*	*wet*	*soak*	*dry, wet, soak*
closed	0.1	0.3	0.5	1/3
open	0.3	0.4	0.4	1/3
diff	0.6	0.3	0.1	1/3

V2			
Drill \ **Oil**	*dry*	*wet*	*soak*
drill	-70	50	200
no_drill	0	0	0

Figure 1: Influence diagram for oil wildcatter problem.

2.1 Influence diagrams

An ID is defined on a directed acyclic graph with three kinds of nodes. *Chance nodes*, drawn as circles, represent random variables, $\mathbf{X} = \{X_1, \ldots, X_m\}$, as in a Bayesian network. *Decision nodes*, drawn as rectangles, represent controllable decision variables, $\mathbf{D} = \{D_1, \ldots, D_n\}$. *Value nodes*, which are drawn as diamonds and have no children, represent the preferences of the decision maker. Figure 1 shows an ID for the classic oil wildcatter decision problem [Raiffa, 1968].

Each random variable $X_i \in \mathbf{X}$ is associated with a conditional probability table, $P_i = P(X_i|pa(X_i))$, where $pa(X_i) \subseteq \mathbf{X} \cup \mathbf{D} \backslash \{X_i\}$ denotes the set of parent variables of X_i in the graph. Similarly, each decision variable $D_k \in \mathbf{D}$ has a parent set $pa(D_k) \subseteq \mathbf{X} \cup \mathbf{D} \backslash \{D_k\}$, denoting the variables whose values are observed before the decision is made. Each value node is associated with a value function $V_i \in \mathbf{V} = \{V_1, \ldots, V_q\}$ that assigns a scalar value to each instantiation of $pa(V_i) \subseteq \mathbf{X} \cup \mathbf{D}$. Given multiple value nodes, we assume the total value is their sum.

We assume that IDs are regular and no-forgetting, which means there is a temporal ordering of the decision variables, denoted $D_1, D_2, \ldots, D_n$, and a decision node and its parents are parents of all subsequent decision nodes. The set of chance variables observed before the first decision is denoted I_0, the set of chance variables observed between decisions D_k and D_{k+1} is denoted I_k, and the set of chance variables that are never observed is denoted I_n. We use $dom(X)$ to denote the domain of a variable $X \in \mathbf{X} \cup \mathbf{D}$. By extension, for $\mathbf{W} \subseteq \mathbf{X} \cup \mathbf{D}$, $dom(\mathbf{W}) = \prod_{W \in \mathbf{W}} dom(W)$.

A strategy for an influence diagram is a list of decision rules $\Delta = (\delta_1, \ldots, \delta_n)$, one for each decision variable $D_i \in \mathbf{D}$, where a decision rule is a mapping, $\delta_i : dom(pa(D_i)) \rightarrow dom(D_i)$, that prescribes an action for each instantiation of the parent variables $pa(D_i)$. An ID is solved by computing a strategy with the maximum expected value, which is equal to

$$MEU = \sum_{\mathbf{I}_0} \max_{D_1} \ldots \sum_{\mathbf{I}_{n-1}} \max_{D_n} \sum_{\mathbf{I}_n} \left(\prod_{i=1}^{m} P_i \sum_{j=1}^{q} V_j \right). \quad (1)$$

Because the decision rule δ_i for the decision variable D_i is conditioned on all possible instantiations of the set $\mathbf{I}_0 \cup D_1 \cup \mathbf{I}_1 \ldots D_{i-1} \cup \mathbf{I}_{i-1}$, the complexity of ID evaluation can grow exponentially in the length of the history. (In limited cases, structural analysis of the ID can distinguish relevant from irrelevant variables in this history, which can help slow this exponential growth in complexity [Shachter, 1999; Nielsen and Jensen, 1999; Lauritzen and Nilsson, 2001].)

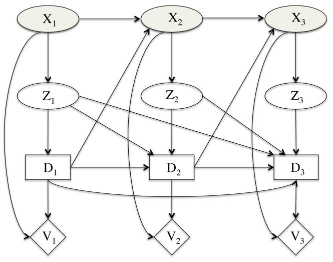


Figure 2: Three-stage POMDP as influence diagram.

2.2 Finite-horizon POMDPs

A finite-horizon POMDP can be viewed as a special case of an ID, defined as follows. For each stage t in a finite sequence of stages, there is an unobserved state variable X_t, an observation variable Z_t, a decision variable D_t, and a reward function V_t. The state and observation variables correspond to the chance nodes of an ID, the decision variables correspond to the decision nodes, and the reward functions correspond to the value nodes, as shown in Figure 2. The state variables at each stage satisfy the *Markov property*, which means that the state at stage t depends only on the state and action at the previous stage, and the observation and reward at stage t depend only on the state at stage t and the previous state and action.

Two approaches to solving POMDPs by dynamic programming have been explored in the literature. In one approach, a *decision tree* is built where each node is associated with a probability distribution over the state space, called a *belief state*, which is updated by Bayesian conditioning. In this form, the problem is solved by backwards induction or related branch-and-bound search techniques [Satia and Lave, 1973; Washington, 1997]. This approach is related to traditional algorithms for solving IDs insofar as it considers and evaluates each history beginning from the starting belief state.

In the second approach, the space of all possible belief states is considered at each stage, instead of the space of possible histories, and strategies are built in the backwards direction without considering the previous history. (Because strategies are built in the backwards direction, we will index stages in backwards order from now on, so that t denotes the number of stages remaining in the problem.) A key feature of this second dynamic programming approach is that the value function is piecewise linear and convex, or, in the case of maximization, concave [Smallwood and Sondik, 1973]. To illustrate this approach, we briefly review the incremental pruning algorithm [Cassandra *et al.*, 1997].

Consider a finite-horizon POMDP that is stage-invariant with a finite set of states X, a finite action set D, a finite observation set Z, a set of transition probabilities $Pr(x'|x,d)$ that give the probability of a transition to state x' after action d is taken in state x, a set of observation probabilities $Pr(z|x',d)$ that give the probability of observing z after action d leads to state x', and a reward function $R(x,d)$ that gives the expected reward for taking action d in state x.

When the number of actions and observations is finite, the set S_t of all possible t-stage strategies (sometimes called "policy trees") can be defined recursively, as follows:

$$S_1 = D, \tag{2}$$

$$S_t = \{\langle d_t, \beta_t \rangle | d_t \in D, \beta_t : Z \rightarrow S_{t-1}\}, \text{for } t > 1, \tag{3}$$

where $s_t^i \in S_t$ denotes a specific t-stage strategy indexed by i. A t-stage strategy s_t^i is defined as a tuple $\langle d_t^i, \beta_t^i \rangle$ consisting of an action $d_t^i \in D$ and a mapping $\beta_t^i : Z \rightarrow S_{t-1}$, such that each observation $z \in Z$ is mapped to a $(t-1)$-stage strategy.

A value vector for a t-stage strategy $s_t^i \in S_t$ can be defined recursively, as follows:

$$v_1^d(x) = R(x, d) \tag{4}$$

$$v_t^i(x) = R(x, d_t^i) + \sum_{x' \in X, z \in Z} Pr(x', z|x, d_t^i) v_{t-1}^{\beta_t^i(z)}(x'), \tag{5}$$

where $\beta_t^i(z)$ is the index of a $(t-1)$-stage strategy in S_{t-1}, and $Pr(x', z|x, d) = Pr(x'|x, d) \cdot Pr(z|x', d)$. Thus the value of a t-stage strategy $s_t^i \in S_t$ for a given belief state b is given by the dot product of the value vector v_t^i and the belief state b:

$$V_t^i(b) = \sum_{x \in X} b(x) \cdot v_t^i(x). \tag{6}$$

Given a set S_t of t-stage strategies with a corresponding set of value vectors $\mathcal{V}_t = \{v_t^i | i = 1, \ldots, |S_t|\}$, a strategy $s_t^i \in S_t$ is said to be dominated by the other strategies $S_t \backslash \{s_t^i\}$ if for all belief states b:

$$\sum_{x \in X} b(x) v_t^i(x) \leq \max_{v_t^k \in \mathcal{V}_t \backslash \{v_t^i\}} \sum_{x \in X} b(x) v_t^k(x). \tag{7}$$

The condition given by (7) can be tested by solving a linear program, and Cassandra *et al.*(1997) describe an algorithm, called PR($\mathcal{V}_t$), that removes all dominated value vectors from an input set $\mathcal{V}_t$, as well as all associated t-stage strategies from the set S^t.

With this background, it is possible to compute an optimal piecewise-linear and concave value function V_t for each stage t, as well as a corresponding minimal set $\mathcal{V}_t$ of undominated t-stage strategies, as follows:

$$\mathcal{V}_t = PR(\cup_{a \in A} \mathcal{V}_t^a) \tag{8}$$

$$\mathcal{V}_t^a = PR(\oplus_{z \in Z} \mathcal{V}_t^{a,z}) \tag{9}$$

$$\mathcal{V}_t^{a,z} = PR(\{v_t^i | i = 1, \ldots, |\mathcal{V}_{t-1}|\}) \tag{10}$$

where the *cross sum* of two sets of vectors A and B is defined as $A \oplus B = \{a + b | a \in A, b \in B\}$, and

$$v_t^i(x) = R(x, a)/|Z| + \sum_{x' \in X} Pr(x', z|x, a) v_{t-1}^i(x'). \tag{11}$$

Steps (10) and (11) are referred to as the *backprojection* step of the algorithm, step (9) is referred to as the *cross-sum* step, and step (8) is referred to as the *maximization* step. The algorithm is called *incremental pruning* when the cross-sum step is performed efficiently by pruning as follows:

$$\mathcal{V}_t^a = PR(\ldots PR(PR(V_t^{a,z_1} \oplus \mathcal{V}_t^{a,z_2}) \oplus \mathcal{V}_t^{a,z_3}) \ldots \oplus \mathcal{V}_t^{a,z_k}). \tag{12}$$

The algorithm is well-described in the literature, to which we refer for further details about its efficient implementation.

In the rest of the paper, we show how to generalize this algorithm so that it solves any regular and no-forgetting ID.

3 New algorithm

The algorithm we describe next can be viewed as a synthesis of the incremental pruning algorithm and the classic node-removal/arc-reversal algorithm for ID evaluation [Olmsted, 1983; Shachter, 1986; Tatman and Shachter, 1990]. Briefly, the algorithm is created by generalizing the cross sum and maximization steps of incremental pruning to create new reductions that apply to the observed chance nodes and the decision nodes of an ID, respectively, and replacing the backprojection step of the incremental pruning algorithm with reductions used by the node-removal/arc-reversal algorithm.

3.1 Standard reductions

We start by summarizing all of the reductions of the node-removal/arc-reversal algorithm except for the reduction for decision nodes, which we will not use in our algorithm.

Barren node removal

A decision or chance node is said to be barren if it has no children, in which case it can be removed without having any effect on the value of an ID [Shachter, 1986, pp. 876].

Value node removal

Although the original node-removal/arc-reversal algorithm assumed a single value node, it was later extended to allow multiple value nodes. When there are multiple value nodes, any two value nodes can be replaced by a single value node that is equal to their sum [Tatman and Shachter, 1990, p. 374]. Let $U_{old}(pa(U))$ denote the value function associated with the value node U, and let $V_{old}(pa(V))$ denote the value function associated with the value node V before the two value nodes are merged. The value function associated with the new value node V_{new} that replaces the two merged value nodes is

$$V_{new}(\mathbf{c}) = U_{old}(\mathbf{a}) + V_{old}(\mathbf{b}), \tag{13}$$

where $\mathbf{A} = pa_{old}(U)$ is the original set of parents of U, $\mathbf{B} = pa_{old}(V)$ is the original set of parents of V, and $\mathbf{C} = pa_{new}(V) = \mathbf{A} \cup \mathbf{B}$ is the new set of parents of V.

To gain computational leverage from decomposition of the value function, it is well-understood that the merging of value nodes should be delayed as long as possible.

Chance node removal

A chance node can be removed if it has a single child value node [Shachter, 1986, pp. 876-7]. Let $P_{old}(X|pa(X))$ denote the conditional probability table associated with a chance node X, and let $V_{old}(pa(V))$ denote the value function associated with its child value node V before removing X. After removing X, the parents of X are inherited by the value node V, and the new value function associated with V is

$$V_{new}(\mathbf{c}) = \sum_{x \in X} P_{old}(x|\mathbf{a}) \cdot V_{old}(\mathbf{b}), \tag{14}$$

where $\mathbf{A} = pa_{old}(X)$ is the original set of parents of X, $\mathbf{B} = pa_{old}(V)$ is the original set of parents of V, and $\mathbf{C} = pa_{new}(V) = \mathbf{A} \cup (\mathbf{B} \backslash \{X\})$ is the new set of parents of V.

For reasons that we explain later, our algorithm only uses this reduction for chance nodes that are unobserved.

Arc reversal
Arc reversal uses Bayes' rule to reverse the direction of an arc between two chance nodes, while ensuring that the original distribution is still correctly represented [Shachter, 1986, pp. 878]. An arc from a chance node X to a chance node Y with parent sets $\mathbf{A}$ (parents of X only), $\mathbf{B}$ (parents of both X and Y), and $\mathbf{C}$ (parents of Y only) can be reversed unless there is another directed path from X to Y; in that case, reversing the arc would create a directed cycle.

When an arc (X, Y) can be reversed, the new distribution for Y is obtained by marginalizing X from the joint, as follows:

$$P_{new}(y|\mathbf{a}, \mathbf{b}, \mathbf{c}) = \sum_{x \in X} P_{old}(y|x, \mathbf{b}, \mathbf{c}) P_{old}(x|\mathbf{a}, \mathbf{b}). \quad (15)$$

Since the chance node Y inherits the parents of X, its new parents are $pa_{new}(Y) = pa_{old}(Y) \cup pa_{old}(X) = \mathbf{A} \cup \mathbf{B} \cup \mathbf{C}$.

The new distribution for Y is used to obtain the new distribution for X, as follows:

$$P_{new}(x|y, \mathbf{a}, \mathbf{b}, \mathbf{c}) = \frac{P_{old}(y|x, \mathbf{b}, \mathbf{c}) P_{old}(x|\mathbf{a}, \mathbf{b})}{P_{new}(y|\mathbf{a}, \mathbf{b}, \mathbf{c})}. \quad (16)$$

Since the chance node X is now a child of Y, and also inherits the parents of Y, its new parents are $pa_{new}(X) = \{Y\} \cup pa_{old}(X) \cup pa_{old}(Y) = \{Y\} \cup \mathbf{A} \cup \mathbf{B} \cup \mathbf{C}$.

3.2 New reductions

Besides the reductions just summarized, the classic node-removal/arc-reversal algorithm includes one more reduction, which is for decision nodes [Shachter, 1986, pp. 877-8]. It can be applied to a decision node if and only if it has a single child node that is a value node, and all other parents of the child value node are also parents of the decision node.

We do not use this reduction because it presupposes a history-based strategy representation. Instead, we replace it with two new reductions that represent strategies in a different way. One is for decision nodes and the other is for chance nodes that are observable, which we call *observation nodes*.

Strategy representation
Our new algorithm represents strategies in a way that is similar to how they are represented in solving POMDPs. Let T denote the total number of decision and observation nodes, and let $N_T, N_{T-1}, \ldots, N_1$ denote an ordering of the decision and observation nodes that respects the temporal ordering of the decision nodes of the ID, as well as a topological ordering of the observation nodes. In this ordering, the subscript t of N_t indicates the number of stages until the end of the problem. Thus any instantiation of the sequence $N_T, N_{T-1}, \ldots, N_1$ corresponds to a valid history.

We use the symbol N_t for both decision and observation nodes because it simplifies our notation when specifying strategies. For each node N_t, let $Act(N_t)$ denote the set of associated actions, and let $Obs(N_t)$ denote the set of associated observations. Of course, actions are only associated with decision nodes, and observations are only associated with observation nodes. But if N_t is a decision node, we can let $Obs(N_t)$ equal a singleton set containing a dummy observation that provides no information. Similarly, if N_t is an observation node, we can let $Act(N_t)$ equal a singleton set containing a dummy action that has no effect. This convention does not change the meaning of the ID, and using the same notation for decision and observation nodes lets us recursively define a set S_t of t-stage strategies, as follows,

$$S_1 = Act(N_1), \quad (17)$$
$$S_t = \{\langle d_t, \beta_t \rangle | d_t \in Act(N_t), \beta_t : Obs(N_t) \to S_{t-1}\}, (18)$$

where a t-stage strategy $s_t \in S_t$ specifies a strategy for the last t stages of the problem, beginning from node N_t, in much the same way that strategies for a finite-horizon POMDP are recursively defined by Equations (2) and (3).

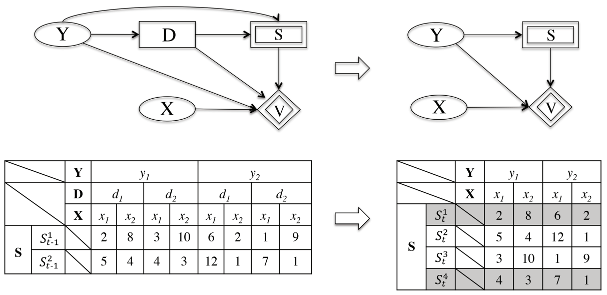
Strategy node and strategy value node
Our new algorithm first solves an ID for the last stage of the problem, then for the last two stages, then for the last three stages, and so on, in a way that generalizes the incremental pruning algorithm. It uses two nodes of the ID to hold the solution as it is gradually constructed. The first node, which we call the *strategy node*, is the last decision node in the ID. It is used to hold a set S_t of strategies, as defined by Equations (17) and (18). The second node, which we call the *strategy value node*, is a child value node of the strategy node. It serves as a piecewise-linear and concave value function for the set of strategies, as we show below.

The two nodes are originally established in a simple initialization step. First, the last decision node in the ID is identified. Then the standard reductions described above are used to eliminate all of its descendants except for one value node (which must exist if the decision node cannot be removed by a series of barren node removals). Because the strategy node is just a decision node used in a special way, and the strategy value node is just its child value node, an ID with these two nodes is well-defined.

Let D denote the strategy node, let V denote the strategy value node, let $\mathbf{X}$ denote the set of parents of the strategy value node that are unobserved chance nodes, and let $\mathbf{Y}$ denote the set of parents of the strategy value node that are observation nodes or decision nodes, excluding the strategy node. For each action $d \in Act(D)$, and for any instantiation $\mathbf{x} \in \mathbf{X}$ of the hidden state variables, and for any instantiation $\mathbf{y} \in \mathbf{Y}$ of the related decision and observation variables, the value function $V(d, \mathbf{x}, \mathbf{y})$ associated with the strategy value node gives the value of taking the action d. Moreover, for any belief state b over instantiations of the hidden state variables $\mathbf{X}$ (that is, for any probability distribution over instantiations $\mathbf{x} \in \mathbf{X}$), and for any instantiation $\mathbf{y} \in \mathbf{Y}$ of the related decision and observation variables, the expected value of decision d is given by: $\sum_{\mathbf{x}} b(\mathbf{x}) V(d, \mathbf{x}, \mathbf{y})$. Finally, for any belief state b over instantiations $\mathbf{x} \in \mathbf{X}$ of the hidden state variables, and for any instantiation $\mathbf{y} \in \mathbf{Y}$ of the decision and observation variables, the optimal value function is

$$V(b, \mathbf{y}) = \max_{d \in Act(D)} \sum_{\mathbf{x} \in \mathbf{X}} b(\mathbf{x}) V(d, \mathbf{x}, \mathbf{y}). \quad (19)$$

As this analysis shows, the value function associated with the strategy value node is a piecewise linear and concave value function, similar to the value function for a POMDP, and we leverage this observation in the algorithm we describe next.



	Y	y_1				y_2			
	D	d_1		d_2		d_1		d_2	
	X	x_1	x_2	x_1	x_2	x_1	x_2	x_1	x_2
S	S^1_{t-1}	2	8	3	10	6	2	1	9
	S^2_{t-1}	5	4	4	3	12	1	7	1

	Y	y_1		y_2	
	X	x_1	x_2	x_1	x_2
S	S^1_t	2	8	6	2
	S^2_t	5	4	12	1
	S^3_t	3	10	1	9
	S^4_t	4	3	7	1

Figure 3: Example of ID before (left) and after (right) a decision node reduction. Tables show value function for value node. Shaded strategies in the revised table are dominated.

Both the strategy node and the strategy value node play an integral role in the following two new reductions.

Decision node reduction

We first describe a reduction for decision nodes. It can be applied to a decision node if and only if its only children are the strategy node and (possibly) the strategy value node. (It is a parent of both, usually, because if it is not a parent of the strategy value node, it has no effect on value.) Importantly, and by contrast to the decision node reduction for the classic node-removal/arc-reversal algorithm, it allows the child value node of the decision node being reduced to have unobserved chance nodes as parents.

The reduction removes the decision node N_t, it modifies the action set associated with the strategy node, and it modifies the value function associated with the strategy value node. Figure 3 shows a simple example of the decision node reduction. To indicate the special role played by the strategy node and strategy value node, the strategy node is shown as a double square labeled S, and the strategy value node is shown as a double diamond labeled V.

Let S_{t-1} denote the set of $(t-1)$-stage strategies originally associated with the strategy node (in its action set), and let V_{old} denote the value function originally associated with the strategy value node. The reduction replaces the set S_{t-1} with a set S_t of t-stage strategies based on Equation (18). That is, for each strategy $s_{t-1} \in S_{t-1}$ and for each action $d_t \in Act(N_t)$ associated with the removed decision node, the new set S_t includes a strategy $s_t = \langle d_t, \beta_t \rangle$ where β_t maps the dummy observation to the strategy $s_{t-1} \in S_{t-1}$.

Let $\mathbf{X}$ denote the set of parents of the strategy value node that are unobserved chance nodes, and let $\mathbf{Y}$ denote the set of parents of the strategy value node that are either observation nodes or decision nodes, excluding the strategy node and (removed) decision node. When the decision node is removed, the new value function for the strategy value node is:

$$V_{new}(\mathbf{x}, \mathbf{y}, s_t) = V_{old}(\mathbf{x}, \mathbf{y}, d_t, s_{t-1}). \tag{20}$$

Note that the sets $\mathbf{X}$ and $\mathbf{Y}$ are not changed by removing the decision node. Adopting a notation similar to the one we used in the POMDP case, we also let $v^i_t(\mathbf{x}, \mathbf{y})$ denote the value vector associated with a strategy $s^i_t \in S_t$.

After the decision node is removed, the last step of the reduction is to eliminate dominated strategies in the new set S_t corresponding to the action set of the revised strategy node. We consider two cases. In the first, $\mathbf{X}$ is empty, that is, the strategy value node does not have any parents that are unobserved chance nodes. This case is relatively easy: a strategy $s^i_t \in S_t$ with corresponding value vector v^i_t is dominated if

$$v^i_t(\mathbf{y}) \leq \max_{v^k_t \in \mathcal{V}^t \setminus \{v^i_t\}} v^k_t(\mathbf{y}), \forall \mathbf{y} \in \mathbf{Y}, \tag{21}$$

which is easily tested by enumerating all instantiations of $\mathbf{Y}$.

In the second case, $\mathbf{X}$ is non-empty, that is, the strategy value node has parents that are unobserved chance nodes. This case requires reasoning about hidden state in a way that generalizes the pruning step of the incremental pruning algorithm for POMDPs. Given a set S_t of t-stage strategies with corresponding value vectors $\mathcal{V}_t$, a strategy $s^i_t \in S_t$ is dominated by the other strategies $S_t \setminus \{s^i_t\}$ if for all belief states b:

$$\sum_{\mathbf{x} \in \mathbf{X}} b(\mathbf{x}) v^i_t(\mathbf{x}, \mathbf{y}) \leq \max_{v^k_t \in \mathcal{V}_t \setminus \{v^i_t\}} \sum_{\mathbf{x} \in \mathbf{X}} b(\mathbf{x}) v^k_t(\mathbf{x}, \mathbf{y}), \forall \mathbf{y} \in \mathbf{Y}. \tag{22}$$

Because this test considers all possible belief states, that is, all probability distributions over the hidden states, it requires use of linear programming. In the worst case, it solves a linear program for each strategy $s^i_t \in S_t$ and instantiation $\mathbf{y} \in \mathbf{Y}$.

Observation node reduction

The observation node reduction we describe next can be applied to an observation node if and only if its only children are the strategy node and (possibly) the strategy value node.

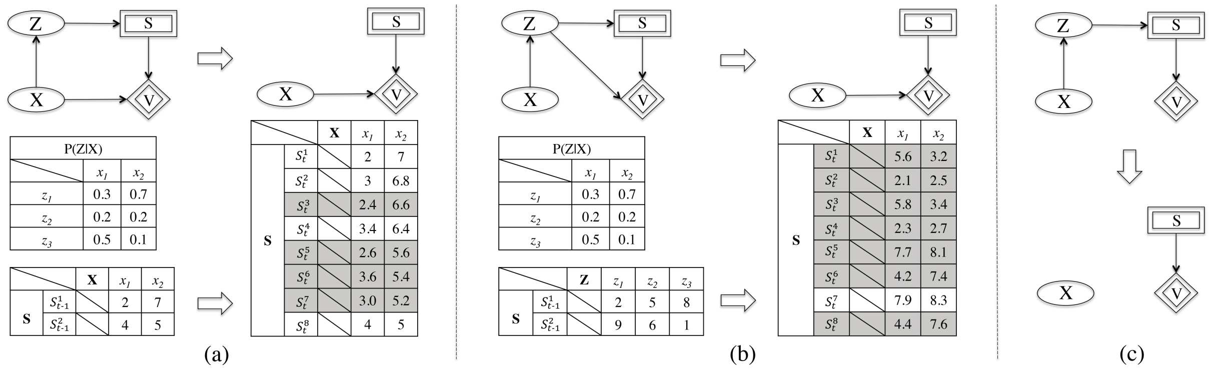
The reduction removes the observation node N_t, it modifies the action set associated with the strategy node, and it modifies the value function associated with the strategy value node. Again, the algorithm replaces the set S_{t-1} of $(t-1)$-stage strategies originally associated with the strategy node with a new set S_t of t-stage strategies based on Equation (18). For each new strategy $s_t \in S_t$, where $s_t = \langle d_t, \beta_t \rangle$, d_t is now a dummy action that has no effect, and β_t maps each observation $z \in Obs(N_t)$ to a strategy $s_{t-1} \in S_{t-1}$.

We must consider how the parent set of the strategy value node is changed by this reduction, and how its new value function is defined. Let $\mathbf{A}$ denote the parents of the strategy value node before the reduction that are unobserved chance nodes, and let $\mathbf{B}$ denote the parents of the strategy value node before the reduction that are observation or decision nodes, excluding the strategy node and (removed) observation node. Let $\mathbf{C}$ denote the parents of the removed observation node that are unobservable chance nodes, and let $\mathbf{D}$ denote the parents of the removed observation node that are observation or decision nodes. The new parent set for the strategy value node consists of the unobservable chance nodes $\mathbf{X} = \mathbf{A} \cup \mathbf{C}$ and the observation and decision nodes $\mathbf{Y} = \mathbf{B} \cup \mathbf{D}$. The new value function for the strategy value node is,

$$V_{new}(\mathbf{x}, \mathbf{y}, s_t) = \sum_{z \in dom(N_t)} P(z|\mathbf{c}, \mathbf{d}) \cdot V_{old}(\mathbf{a}, \mathbf{b}, z, \beta_t(z)), \tag{23}$$

where $z \in dom(N_t)$ is an observation, and $\beta_t(z)$ maps the observation z to a strategy in S_{t-1}.

Figure 4 show some simple examples of this reduction. There is a subtle complication in determining the parent set of



P(Z|X)	x_1	x_2
z_1	0.3	0.7
z_2	0.2	0.2
z_3	0.5	0.1

(a)

S	X	x_1	x_2
$S_{t\text{-}1}^1$		2	7
$S_{t\text{-}1}^2$		4	5

S	X	x_1	x_2
S_t^1		2	7
S_t^2		3	6.8
S_t^3	▨	2.4	6.6
S_t^4		3.4	6.4
S_t^5	▨	2.6	5.6
S_t^6	▨	3.6	5.4
S_t^7	▨	3.0	5.2
S_t^8		4	5

(b)

P(Z|X)	x_1	x_2
z_1	0.3	0.7
z_2	0.2	0.2
z_3	0.5	0.1

S	Z	z_1	z_2	z_3
$S_{t\text{-}1}^1$		2	5	8
$S_{t\text{-}1}^2$		9	6	1

S	X	x_1	x_2
S_t^1	▨	5.6	3.2
S_t^2	▨	2.1	2.5
S_t^3	▨	5.8	3.4
S_t^4	▨	2.3	2.7
S_t^5	▨	7.7	8.1
S_t^6	▨	4.2	7.4
S_t^7		7.9	8.3
S_t^8	▨	4.4	7.6

Figure 4: Examples of observation node reduction. Dominated strategies are shaded in the table for the value function.

the strategy value node after the reduction that we mention to qualify the discussion in the previous paragraph. If a parent node of the observation node is not connected to the strategy value node by a path that does not go through the strategy node, then it is *irrelevant* in a sense defined by Nielsen [2002] and Jensen and Nielsen [2007, pp. 413-420], which means it has no effect on expected value. Thus it can be excluded from the parent set of the strategy value node after the reduction. In the example shown in Figure 4(c), for example, the node X is irrelevant and becomes barren after the reduction.

After the observation node has been removed, the last step of the reduction is to eliminate dominated strategies in the new set S_t corresponding to the action set of the revised strategy node. Adopting a notation similar to the one used in the POMDP case, we let $v_t^i(\mathbf{x}, \mathbf{y})$ denote the value vector associated with a strategy $s_t^i \in S_t$. As already explained in the discussion of eliminating dominated strategies after the reduction of a decision node, there are two cases to consider. In the first case, $\mathbf{X}$ is empty, that is, the strategy value node does not have any parents that are unobserved chance nodes. In this case, a strategy $s_t^i \in S_t$ is dominated if it satisfies the condition of Equation (21), which can be tested without linear programming. In the second case, $\mathbf{X}$ is non-empty, that is, the strategy value node has parents that are unobserved chance nodes. In this case, a strategy $s_t^i \in S_t$ is dominated by the other strategies $S_t \backslash \{s_t^i\}$ if it satisfies the test of Equation (22) for all belief states b over the hidden state variables $\mathbf{X}$.

Efficient implementation of this pruning step is more challenging for an observation node reduction than a decision node reduction because the cardinality of the set of strategies grows *much* more explosively when an observation node is removed than when a decision node is removed. In the incremental pruning algorithm, this step is referred to as the *cross sum* step, and it can be performed efficiently by interleaving generation and pruning of the new strategies. First, the algorithm considers a partial mapping $\beta_t : Obs(N_t) \rightarrow S_{t-1}$ with just one value $z \in Obs(N_t)$, generates all partial strategies based on this mapping, and prunes dominated strategies. Then it considers a more complex mapping with two values of the observation variable, generates all strategies, and prunes those that are dominated. It continues to generate and prune partial strategies in this way until all values of the observation variable have been considered. This "incremental" approach to generating and pruning observation-based strategies is the idea for which the incremental pruning algorithm is named. We refer to Cassandra *et al.* [1997] for further details.

3.3 Correctness and order of reductions

The algorithm terminates when only the strategy node and strategy value node remain, at which point the strategy node holds an optimal strategy represented as an acyclic graph we call a *strategy graph*. The correctness of the algorithm follows from the fact that all reductions are *value-preserving*, in the sense defined by Shachter [1986]. The new decision and observation node reductions are value-preserving because they transform the ID into a smaller ID without changing the value of the strategies, and elimination of dominated strategies leaves the optimal strategy unaffected.

In contrast to the classic node-removal/arc-reversal algorithm, our algorithm allows more flexibility in the order in which nodes are removed from the ID. This flexibility makes it more effective in solving some problems, as we will see, although it also means the algorithm's efficiency depends critically on choosing an appropriate order of reductions.

4 Examples and analysis

To illustrate the algorithm, and especially the performance of different heuristics for determining the order of reductions, we consider three IDs from the literature.

4.1 Oil Wildcatter

In this problem, shown in Figure 1, a wildcatter must decide whether or not to drill for oil. The amount of available oil is a hidden state variable with three possible states: dry, wet, and soak. Before deciding whether to drill, the wildcatter can perform a seismic test with three possible outcomes (closed, open, and diffuse), to estimate how much oil is present.

Figure 5 shows how our algorithm solves this ID using two different orders of reductions. In Figure 5(a), the unobservable chance node for *Oil* is removed before any other nodes are removed. As a result, there is no need to reason about hidden state or use linear programming to prune dominated strategies. The classic node-removal/arc-reversal algorithm removes nodes in exactly the same order.

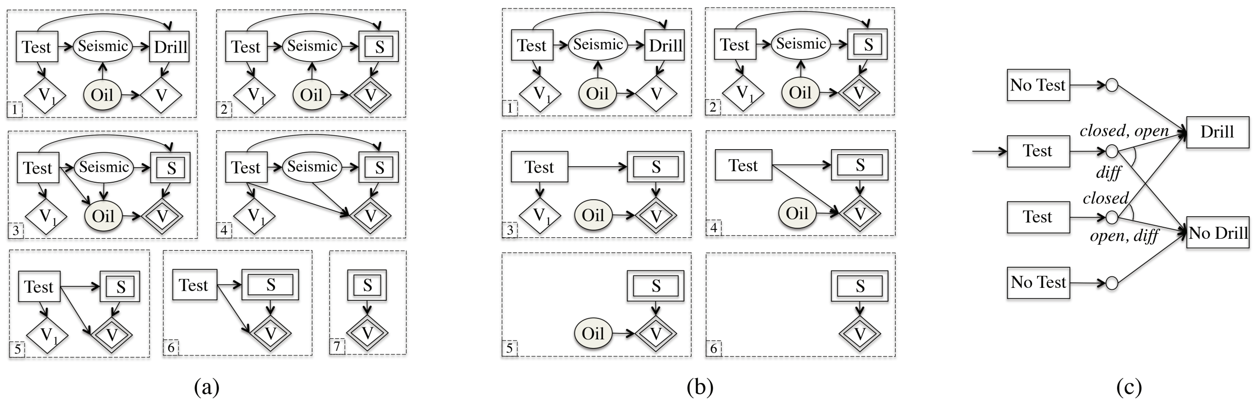


Figure 5: For oil wildcatter problem: (a) sequence of reductions with unobservable node removed *first*, (b) sequence of reductions with unobservable node removed *last*, and (c) optimal strategy graph for all belief states (arrow for initial belief state).

Figure 5(b) shows an order of reductions that is not possible using the classic approach: the unobservable chance node for *Oil* is removed *after* all the other nodes are removed. As a result, removing the observation node (*Seismic*) and decision node (*Test*) requires adopting the POMDP approach to reason about hidden state and prune dominated strategies. Our algorithm solves this ID in a small fraction of a second regardless of which of these two orders of reductions is used.

It is interesting to consider panel 5 of Figure 5(b), which shows the ID at the point where all of the nodes have been removed except for the unobservable chance node for *Oil* (as well as the strategy node and strategy value node). Figure 5(c) shows the set of undominated strategies at this point, which is represented compactly as an acyclic graph we call a *strategy graph*. The solution shown in Figure 5(c) includes an optimal strategy for every possible prior probability distribution for the unobserved state variable for Oil, which means it is a more general solution than usually found by ID algorithms. The decision about which of the four nodes on the left to start with depends on the prior probability distribution. When the prior state probability distribution is 0.5 for *dry*, 0.3 for *wet*, and 0.2 for *soak*, it is best to start in the node marked by an arrow. The final reduction, which removes the hidden state variable for *Oil*, uses this prior probability distribution to select this optimal strategy, and discards the unreachable parts of the strategy graph.

4.2 Maze navigation

We next consider a partially observable maze navigation problem introduced in previous work on limited-memory influence diagrams [Nilsson and Hohle, 2001]. Figure 6(a) shows the maze. The shaded tiles represent walls, the white tiles represent movable space, and the white tile with a star is the goal state. A robot in the maze has four available actions; it can move a single step in any of the four compass directions. The robot successfully moves in the intended direction with probability 0.89. It fails to move with probability 0.089, it moves sideways with probability 0.02 (0.01 for each side), and it moves backward with probability 0.001. If movement in some direction would take it into a wall, that movement has probability zero, and the probability of not moving is increased. The robot has four sensors, one for each direction, which accurately sense whether the neighboring tile in that direction is a wall. Because different states can result in the same observation, the problem is partially observable.

A robot is randomly placed in a non-goal state (i.e., the prior probability distribution is uniform), and performs an action at each of a sequence of ten stages. If it reaches the goal state by the final stage, it receives a reward of 1; otherwise, it receives a reward of 0. Thus the objective is to maximize the probability of reaching the goal state within ten stages.

Although Nilsson and Hohle [2001] model the problem as an ID, it is a very difficult problem for traditional ID algorithms to solve. (This fact motivates their use of limited-memory IDs.) At each stage, the robot receives one of 16 observations and performs one of 4 actions, which means that 64^{10} different histories are possible over ten stages. Given so many histories, traditional algorithms for ID evaluation cannot solve this problem. Even representation of a strategy as a simple mapping from histories to actions is infeasible with so many histories.

Surprisingly, the problem can be solved in less than a minute by our new node-removal/arc-reversal algorithm, if it adopts an order of reductions that reflects the POMDP approach and allows hidden state at each stage. We use the following heuristic to order reductions. If the last decision node before the strategy node is not yet eligible to be removed, we consider its descendants. We remove value node descendants first (except the strategy value node, of course). Then we remove unobservable chance nodes (which may require arc reversals), then observation nodes, and finally the decision node itself. The process then repeats with the next decision node before the strategy node. With this order of reductions, the algorithm performs the same computations as the incremental pruning algorithm. The maze problem is easily solved by this approach because an optimal strategy for this problem can be represented compactly by a strategy graph with only 123 action nodes.

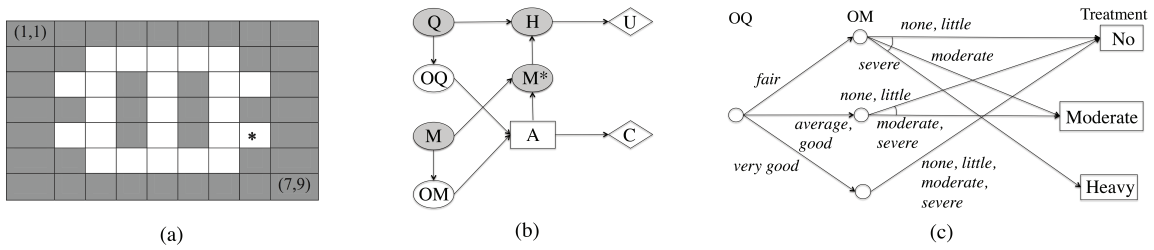


Figure 6: (a) Maze problem. (b) Mildew influence diagram. (c) Optimal strategy graph for mildew problem.

4.3 Mildew treatment

Finally, we consider a simple ID that models the problem of a farmer deciding how to eliminate mildew in a wheat field by selecting among fungicide treatments. The example is originally described by Jensen and Nielsen [2007, pp. 282-283], with more details given at the HUGIN website.[1]

As shown in Figure 6(b), the ID has one decision node (A) with four options for fungicide treatment (none, light, moderate, and heavy). It has two observation nodes: observation of the crop state (OQ), with 4 possible values (fair, average, good, and very good), and observation of the mildew situation (OM), with 4 possible values (none, little, moderate, and severe). It has four unobservable chance nodes. The initial crop state (Q) has 4 values. The crop state at harvest (H) has 7 values. Both the mildew situation before treatment (M) and the mildew situation after treatment (M*) have 4 values.

Two of the unobserved chance nodes (H and M*) must be removed before the decision node and observations nodes are removed. The other two unobserved chance nodes (Q and M) can be removed either before the two observation nodes are removed or after they are removed. If they are removed before the observation nodes are removed, the problem is solved in a fraction of a second by our algorithm. (Note that this order of node removals is the same order in which the nodes would be removed by the traditional node-removal/arc-reversal algorithm.) If the two unobserved chance nodes are not removed until all of the other nodes are removed, including the observation nodes, the problem cannot be solved within 30 minutes by our algorithm because of the large number of undominated strategies.

For this problem, it is better to remove all unobservable chance nodes before the two observation nodes because there are only 64 histories, while the number of possible strategies is 4^{16}, and many are undominated. Thus our algorithm is no faster than (though it is as fast as) the classic node-removal/arc-reversal algorithm in solving this problem. But it does have the advantage that it represents the optimal strategy more compactly, as shown in Figure 6(c).

4.4 Heuristics for ordering reductions

A key difference between our algorithm and the traditional node-removal/arc-reversal algorithm for IDs is that our algorithm allows unobservable chance nodes to be removed *after* decision and observation nodes are removed, which requires reasoning about hidden state and use of linear programming to eliminate dominated strategies. This approach is effective for problems where the number of histories is extremely large but the number of undominated strategies is relatively small, as in the last stages of complex, multistage IDs, such as the maze problem. In fact, the maze problem provides both a best-case example for the POMDP approach and a worst-case example for the approach of traditional ID algorithms, which struggle to solve IDs with a large number of histories.

An attractive property of our algorithm is that it can adopt whichever order of reductions is best for a given ID. When the number of histories is much larger than the number of undominated strategies, as it is for the maze problem, it should order reductions in a way that favors the POMDP approach. When the number of undominated strategies is much larger than the number of histories, as it is for the mildew problem, it should order reductions in the traditional way, where unobservable chance nodes are removed first. By allowing flexibility in the order of reductions, our algorithm combines the advantages of POMDP algorithms and traditional ID algorithms.

5 Conclusion

We have described a node-removal/arc-reversal algorithm for ID evaluation that includes reductions that allow an ID to be solved by a generalization of the dynamic programming approach to solving POMDPs. The algorithm allows a more flexible ordering of node removals, and a POMDP-inspired approach to optimizing over hidden state variables, that can improve the scalability of ID evaluation in solving complex, multi-stage problems. Because the algorithm can remove nodes in the same order as they are removed by the traditional node-removal/arc-reversal algorithm, it can perform at least as well as the traditional algorithm, and potentially better using a different order of reductions that reflects the POMDP approach. It relies on heuristics to select the order of reductions that leads to the best performance for a given problem.

Even when the new algorithm removes nodes in the same order as they are removed by the traditional algorithm, it has the advantage of finding a more compact representation of an optimal strategy. Instead of representing a strategy as a simple mapping from histories to actions, a strategy is represented as an acyclic graph, which can be easier to interpret.

We have sketched the algorithm in broad outline in this paper. In future work, we will refine its details and consider ways to improve its performance, including the development of more sophisticated heuristics for ordering reductions.

[1]See http://www.hugin.com/technology/samples/mildew.

Acknowledgments

This research was partially supported by NSF grant IIS-1219114.

References

[Boutilier and Poole, 1996] C. Boutilier and D. Poole. Computing optimal policies for partially observable Markov decision processes using compact representations. In *Proceedings of the 13th National Conference on Artificial Intelligence*, pages 1168–1175, 1996.

[Cassandra *et al.*, 1997] A. Cassandra, M. Littman, and N. Zhang. Incremental pruning: A simple, fast, exact method for partially observable Markov decision processes. In *Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence*, pages 54–61, 1997.

[Cooper, 1988] G. Cooper. A method for using belief networks as influence diagrams. In *Proc. of the 4th Conf. on Uncertainty in Artificial Intelligence*, pages 55–63, 1988.

[Dechter, 2000] R. Dechter. A new perspective on algorithms for optimizing policies under uncertainty. In Steve Chien, Subbarao Kambhampati, and Craig A. Knoblock, editors, *AIPS*, pages 72–81. AAAI, 2000.

[Hansen and Feng, 2000] E. Hansen and Z. Feng. Dynamic programming for POMDPs using a factored state representation. In *Proc. of the 5th International Conf. on AI Planning Systems (AIPS-2000)*, pages 130–139, 2000.

[Howard and Matheson, 1981] R. Howard and J. Matheson. Influence diagrams. In Ronald Howard and James Matheson., editors, *The Principles and Applications of Decision Analysis*, pages 719–762, Menlo Park, CA, 1981.

[Jensen and Nielsen, 2007] F. Jensen and T. Nielsen. *Bayesian Networks and Decision Graphs*. Springer, New York, 2nd edition, 2007.

[Jensen *et al.*, 1994] F. Jensen, F. V. Jensen, and S. Dittmer. From influence diagrams to junction trees. In *Proc. of the 10th Conference on Uncertainty in Artificial Intelligence*, pages 367–373, 1994.

[Kaelbling *et al.*, 1998] L. Kaelbling, M. Littman, and A. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

[Lauritzen and Nilsson, 2001] S. Lauritzen and D. Nilsson. Representing and solving decision problems with limited information. *Management Science*, 47(9):1235–1251, 2001.

[Monahan, 1982] G. Monahan. A survey of partially observable Markov decision processes: Theory, models, and algorithms. *Management Science*, 28:1–16, 1982.

[Nielsen and Jensen, 1999] T. Nielsen and F. V. Jensen. Well-defined decision scenarios. In *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI-99)*, pages 502–511, 1999.

[Nielsen, 2002] T. Nielsen. Decomposition of influence diagrams. *Journal of Applied Non-Classical Logics*, 12(2):135–150, 2002.

[Nilsson and Hohle, 2001] D. Nilsson and M. Hohle. Computing bounds on expected utilities for optimal policies based on limited information. Technical Report 94, Danish Informatics Network in the Agricultural Sciences, 2001.

[Olmsted, 1983] S. Olmsted. *On representing and solving decision problems*. PhD thesis, Stanford University, 1983.

[Qi and Poole, 1995] R. Qi and D. Poole. A new method for influence diagram evaluation. *Computational Intelligence*, 11:498–528, 1995.

[Raiffa, 1968] H. Raiffa. *Decision analysis*. Addison-Wesley, Reading, MA, 1968.

[Satia and Lave, 1973] J. Satia and R. Lave. Markovian decision processes with probabilistic observation of states. *Management Science*, 20(1):1–13, 1973.

[Shachter and Peot, 1992] R. Shachter and M. Peot. Decision making using probabilistic inference methods. In *Proc. of the 8th Conference on Uncertainty in Artificial Intelligence (UAI-92)*, pages 276–283, 1992.

[Shachter, 1986] R. Shachter. Evaluating influence diagrams. *Operations Research*, 34:871–882, 1986.

[Shachter, 1999] R. Shachter. Efficient value of information computation. In *Proc. of the 15th Conf. on Uncertainty in Artificial Intelligence*, pages 594–601, 1999.

[Smallwood and Sondik, 1973] R. Smallwood and E. Sondik. The optimal control of partially observable Markov processes over a finite horizon. *Operations Research*, 21:1071–1088, 1973.

[Tatman and Shachter, 1990] J. Tatman and R. Shachter. Dynamic programming and influence diagrams. *IEEE Trans. on Systems, Man and Cybernetics*, 20(2):365–379, 1990.

[Washington, 1997] R. Washington. BI-POMDP: bounded, incremental partially observable markov model planning. In *Proceedings of the 4th European Conference on Planning*, pages 440–451, 1997.

[Yuan *et al.*, 2010] C. Yuan, X. Wu, and E. Hansen. Solving multistage influence diagrams using branch-and-bound search. In *Proc. of the 26th Conference on Uncertainty in Artificial Intelligence (UAI-10)*, pages 691–700, 2010.

[Zhang, 1998] N. Zhang. Probabilistic inference in influence diagrams. *Computational Intelligence*, 14(4):475–497, 1998.