Journal of Artificial Intelligence Research 75 (2022) 1177-1221 Submitted 04/2022; published 11/2022

Strategy Graphs for Influence Diagrams

Eric A. Hansen HANSEN@CSE.MSSTATE.EDU
Jinchuan Shi JINCHUANSHI86@GMAIL.COM
James Kastrantas JKASTRANTAS@GMAIL.COM
Dept. of Computer Science and Engineering
Mississippi State University, MS 39762, USA

Abstract

An influence diagram is a graphical model of a Bayesian decision problem that is solved by finding a strategy that maximizes expected utility. When an influence diagram is solved by variable elimination or a related dynamic programming algorithm, it is traditional to represent a strategy as a sequence of policies, one for each decision variable, where a policy maps the relevant history for a decision to an action. We propose an alternative representation of a strategy as a graph, called a strategy graph, and show how to modify a variable elimination algorithm so that it constructs a strategy graph. We consider both a classic variable elimination algorithm for influence diagrams and a recent extension of this algorithm that has more relaxed constraints on elimination order that allow improved performance. We consider the advantages of representing a strategy as a graph and, in particular, how to simplify a strategy graph so that it is easier to interpret and analyze.

1. Introduction

An influence diagram (Howard & Matheson, 1981; Tatman & Shachter, 1990; Jensen & Nielsen, 2011) is a compact graphical model of a Bayesian decision problem that is solved by finding a strategy that maximizes expected utility, where a strategy specifies what action to take in each situation based on available information.

Variable elimination algorithms for solving influence diagrams, and closely-related node reduction algorithms, represent a strategy by a sequence of policies, one for each decision variable, where a policy maps the relevant history for a decision to an action (e.g., Shachter, 1986; Tatman & Shachter, 1990; Shenoy, 1992; Ndilikilikesha, 1994; Jensen, Jensen, & Dittmer, 1994; Dechter, 2000; Jensen & Nielsen, 2007; Luque & Díez, 2010). However, this representation of a strategy has the drawback that it can be difficult for a human user to understand, as well as unnecessarily large, since it specifies an action for every scenario, including those that are impossible or unreachable, and thus irrelevant.

In recent work, Luque, Arias, and Díez (2017) show how to modify a variable elimination algorithm so that it represents a strategy in the more compact and understandable form of a tree, called a *strategy tree*. This alternative representation of a strategy is related to classic methods for solving an influence diagram by unfolding it into an equivalent decision tree, where a strategy tree is a subtree of the decision tree that only includes branches that are reachable by following the strategy. In practice, strategy trees have been shown to be especially useful in domains where clear communication with users about the strategy found by an influence diagram solver is important, for example, in medical decision making (Segal & Shahar, 2009; Luque, Díez, & Disdier, 2016).

In this paper, we develop an approach to strategy representation that leverages the fact that a strategy tree with repeated subtrees can be represented more compactly, and usually *much* more compactly, by an equivalent graph that we call a *strategy graph*. The possibility of compressing a strategy tree into an equivalent graph has been considered before in the literature on influence diagrams, especially the early literature, where it is referred to by the term *coalescence* (Olmsted, 1983; Tatman, 1986; Smith, Holtzman, & Matheson, 1993). However, the representation of a strategy as a graph has almost never been used in practice, apparently due to difficulties in developing an algorithm that can easily perform coalescence when solving an influence diagram. To address these difficulties, the concept of a strategy graph that we develop in this paper generalizes and extends the traditional concept of coalescence in several significant ways.

Our central contribution is to show how to modify a variable elimination algorithm for influence diagrams so that it represents a strategy as a graph. We show how to do so for both a classic variable elimination algorithm and a recently-introduced generalization of this algorithm that integrates the variable elimination approach with techniques adapted from the value iteration algorithm for finite-horizon partially observable Markov decision processes (POMDPs) (Hansen, 2021). The integrated algorithm has more relaxed constraints on the order in which variables can be eliminated, which can improve the performance of the variable elimination approach to solving influence diagrams, sometimes dramatically.

In the original description of this new algorithm, called *generalized variable elimination*, a strategy is represented by a sequence of policies, in keeping with the traditional approach to strategy representation. However, a policy constructed by generalized variable elimination can map belief states (as well as histories) to actions, a representation that it inherits from POMDPs, where a *belief state* is a probability distribution over the hidden states of a problem that is updated by Bayes' rule. Execution of a strategy that is represented in this more general form has the drawback that it requires updating a belief state by Bayes' rule after each action and observation, in order to select the next action. Thus one advantage of modifying the new algorithm so that it represents a strategy as a graph is that it makes it unnecessary to maintain a belief state in order to execute a strategy.

A more important advantage is that a strategy graph can be more compact, and often *much* more compact, than an equivalent representation of a strategy as a tree, or as a sequence of policies. As a result, it can be easier to interpret and analyze. To enhance this advantage, we show how to simplify a strategy graph as much as possible by removing redundant and unreachable nodes, and especially by merging duplicate subgraphs, that is, by coalescence. We distinguish among three forms of coalescence. First, we use the term *implicit coalescence* to refer to a form of coalescence that is performed automatically by the variable elimination algorithm itself when it eliminates variables in an order that leverages Markovian independence from prior history. We show that generalized variable elimination leads to much more implicit coalescence (and speedup) than the classic algorithm because it can solve dynamic programming recurrences over belief states. Second, we consider a form of coalescence that we call *explicit coalescence* because it explicitly checks for and merges duplicate subgraphs that are not merged implicitly. Explicit coalescence is performed by additional steps that are added to the algorithm, and so it incurs some computational overhead (unlike implicit coalescence). Finally, we consider a third form of coalescence that can simplify a strategy graph even further in exchange for bounded-error approximation.

Representing a strategy in the more compact and easier-to-understand form of a graph has advantages that are in keeping with the original motivation for introducing influence diagrams as a tool for decision analysis. An influence diagram is a graphical representation of a decision problem that is exponentially more compact than its equivalent representation as a decision tree, which makes the structure of a *problem* easier to interpret and analyze. In a similar way, the representation of a strategy as a graph can be exponentially more compact than its equivalent representation as a strategy tree, or a sequence of policies, and thus it makes the structure of a *strategy* easier to interpret and analyze.

The paper is organized as follows. Section 2 introduces the concept of a strategy graph. Section 3 shows how to modify a classic variable elimination algorithm for influence diagrams so that it represents a strategy as a graph. Section 4 reviews our recent generalization of this algorithm, which uses POMDP techniques to relax constraints on elimination order, and shows how it can also be modified to represent a strategy as a graph. Section 5 considers how to use coalescence to simplify a strategy graph even further in exchange for bounded-error approximation. Section 6 summarizes the paper and discusses related work.

2. Influence Diagrams and Strategy Representation

We begin with a brief review the influence diagram model and previous approaches to strategy representation. Then we introduce the concept of a strategy graph.

Notation We use upper-case letters such as X to denote variables and lower-case letters such as x to denote values, or states, of variables. We assume variables are discrete. For a variable X, we let $sp(X)$ denote its set of states, or state space. We use bold upper-case letters such as $\mathbf{X}$ to represent sets of variables, that is, joint variables, and we let bold lower-case letters such as $\mathbf{x}$ denote instantiations of $\mathbf{X}$. The state space of a joint variable $\mathbf{X}$ is the Cartesian product of the individual state spaces, that is, $sp(\mathbf{X}) = \times_{X \in \mathbf{X}} sp(X)$.

2.1 Graphical Model of a Decision Problem

An influence diagram is a graphical model of a Bayesian decision problem that is defined on a directed acyclic graph with three types of nodes. *Chance nodes*, drawn as ovals, represent random variables, denoted $\mathbf{C} = \{C_1, \ldots, C_m\}$. *Decision nodes*, drawn as rectangles, represent decision variables, denoted $\mathbf{D} = \{D_1, \ldots, D_n\}$. *Reward (or value) nodes*, drawn as diamonds, represent additive reward functions, denoted $\mathbf{R} = \{R_1, \ldots, R_q\}$, and are the sink nodes of the graph.

The graph of an influence diagram also has three types of arcs, distinguished by the type of node they go into. Arcs into chance nodes, called *conditional arcs*, represent probabilistic dependence, as in a Bayesian network. For each chance variable $C \in \mathbf{C}$, a conditional probability function, $P(C|pa(C))$, maps each instantiation of the variables $\{C\} \cup pa(C)$ to a probability, where $pa(X)$ denotes the set of *parent variables* of variable X in the graph. Arcs into reward nodes, called *functional arcs*, indicate the domain of the associated reward function. Thus a reward function R assigns a scalar value to each instantiation of its parent variables $pa(R)$. Arcs into decision nodes, called *informational arcs*, imply information precedence. That is, an arc from a node for variable X to a node that represents a decision variable D indicates that the state of variable X is known when decision D is made.

We consider influence diagrams under the classic assumptions of (i) a total ordering of decisions and (ii) no-forgetting. The first assumption is equivalent to the assumption that there is a directed path in the graph that includes all of the decision nodes, which means decision variables are ordered in time: $D_1, ..., D_n$. Let $\mathbf{Y}$ denote the subset of chance variables that are observed at some point in the decision problem, and let $\mathbf{X}$ denote the subset of chance variables that are never observed, so that $\mathbf{C} = \mathbf{Y} \cup \mathbf{X}$. Given a total ordering of decision variables, we have a partition of the chance variables, $\{\mathbf{Y}_1, \mathbf{Y}_2, ..., \mathbf{Y}_n, \mathbf{X}\}$, where $\mathbf{Y}_1$ is the set of chance variables whose state is known before the first decision D_1 is made, $\mathbf{Y}_{i+1}$ is the set of chance variables observed after the decision D_i is made and before the decision D_{i+1} is made, and $\mathbf{X}$ is the set of chance variables that are never observed. Given this partition, there is a partial temporal ordering of variables,

$$\mathbf{Y}_1 \prec D_1 \prec \mathbf{Y}_2 \prec D_2 \prec \ldots \prec \mathbf{Y}_n \prec D_n \prec \mathbf{X}, \tag{1}$$

where each decision variable D_i, or set of chance variables $\mathbf{Y}_i$, is instantiated before all subsequent variables in this partial order.

The *no-forgetting* assumption means that if the state of a variable Y is known before a decision D_i is made, it is also known before any posterior decision D_j is made, where $i < j$, even if there is not an explicit arc from Y to D_j in the graph. The set of variables for which the state is known to the decision maker before a decision D_i is made is called the set of *informational predecessors* of D_i, denoted $Pred(D_i)$, and defined as:

$$\begin{aligned} Pred(D_i) &= \mathbf{Y}_1 \cup \{D_1\} \cup \mathbf{Y}_2 \cup \ldots \cup \mathbf{Y}_{i-1} \cup \{D_{i-1}\} \cup \mathbf{Y}_i \\ &= Pred(D_{i-1}) \cup \{D_{i-1}\} \cup \mathbf{Y}_i. \end{aligned} \tag{2}$$

The total ordering of decisions and the no-forgetting assumption reflect the perspective that the decision problem is solved by a single decision maker who makes a sequence of decisions based on perfect recall of all past decisions and observations.

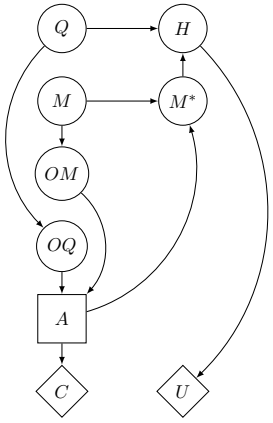
Example: Mildew treatment. Figure 1 shows an influence diagram for a problem of fungicide treatment of mildew in a wheat field. The example is described by Jensen and Nielsen (2007, pp. 282-283) and the probabilities and rewards are from the HUGIN website.[1]

The influence diagram has four unobserved chance variables. The initial crop state (Q) can be *fair* (f), *average* (a), *good* (g), or *very good* (v). The crop state at harvest (H) can be in any one of these four states or three others: *poor* (p), *bad* (b), or *rotten* (r). Both the degree of mildew present before treatment (M) and after treatment (M^*) can be in any one of four possible states: *none* (no), *little* (l), *moderate* (m), or *severe* (s).

There are two observed chance variables: observation of the crop state (OQ), with the same four possible values as Q, and observation of the mildew situation (OM), with the same four possible values as M. Although the possible values are the same, the observations are noisy and imperfectly correlated with the underlying state.

There is one decision variable (A) with four options for fungicide treatment: *none* (no), *light* (l), *moderate* (m), or *heavy* (h). One reward node (C) gives the cost of fungicide treatment. The other (U) gives the value of the final harvest as a function of crop state.

1. See http://camvac.hugin.com/index.php/Mildew.



$P(Q)$			
f	a	g	v
0.2	0.4	0.3	0.1

Q	$P(OQ\|Q)$ f	a	g	v
f	0.8	0.15	0.05	0.0
a	0.3	0.6	0.1	0.0
g	0.1	0.2	0.6	0.1
v	0.0	0.1	0.4	0.5

$P(M)$			
no	l	m	s
0.4	0.3	0.2	0.1

M	$P(OM\|M)$ no	l	m	s
no	0.9	0.1	0.0	0.0
l	0.2	0.5	0.2	0.1
m	0.1	0.2	0.5	0.2
s	0.0	0.1	0.3	0.6

A, M	$P(M^*\|A, M)$ no	l	m	s
no, no	1.0	0.0	0.0	0.0
no, l	0.0	1.0	0.0	0.0
no, m	0.0	0.0	1.0	0.0
no, s	0.0	0.0	0.0	1.0
l, no	1.0	0.0	0.0	0.0
l, l	0.8	0.2	0.0	0.0
l, m	0.0	0.8	0.2	0.0
l, s	0.0	0.0	0.8	0.2
m, no	1.0	0.0	0.0	0.0
m, l	1.0	0.0	0.0	0.0
m, m	0.8	0.2	0.0	0.0
m, s	0.0	0.8	0.2	0.0
h, no	1.0	0.0	0.0	0.0
h, l	1.0	0.0	0.0	0.0
h, m	1.0	0.0	0.0	0.0
h, s	0.8	0.2	0.0	0.0

M^*, Q	$P(H\|M^*, Q)$ r	b	p	f	a	g	v
no, f	0.00	0.05	0.10	0.70	0.10	0.05	0.00
no, a	0.00	0.00	0.05	0.10	0.70	0.10	0.05
no, g	0.00	0.00	0.00	0.05	0.10	0.70	0.15
no, v	0.00	0.00	0.00	0.00	0.10	0.20	0.70
l, f	0.05	0.10	0.70	0.10	0.05	0.00	0.00
l, a	0.00	0.00	0.05	0.10	0.70	0.10	0.05
l, g	0.00	0.00	0.05	0.10	0.70	0.15	0.00
l, v	0.00	0.00	0.00	0.05	0.15	0.70	0.10
m, f	0.15	0.70	0.10	0.05	0.00	0.00	0.00
m, a	0.05	0.10	0.70	0.10	0.05	0.00	0.00
m, g	0.00	0.05	0.10	0.70	0.10	0.05	0.00
m, v	0.00	0.00	0.05	0.10	0.70	0.15	0.00
s, f	0.90	0.10	0.00	0.00	0.00	0.00	0.00
s, a	0.15	0.70	0.10	0.05	0.00	0.00	0.00
s, g	0.05	0.10	0.70	0.10	0.05	0.00	0.00
s, v	0.00	0.05	0.10	0.70	0.10	0.05	0.00

$C(A)$			
h	m	l	no
-4	-3	-2	0

$U(H)$						
r	b	p	f	a	g	v
-1	1	5	8	10	12	13

Figure 1: Influence diagram for the mildew treatment problem.

2.2 Traditional Representation of a Strategy

To solve, or evaluate, an influence diagram means to compute an optimal strategy and its expected utility. For a problem faced by a single decision maker with perfect recall, it is well-known that an optimal deterministic strategy exists. It is traditional to represent such a strategy by a sequence of policies, $\Delta = (\delta_{D_1}, \ldots, \delta_{D_n})$, with one policy for each decision variable $D_i \in \mathbf{D}$, where a policy is a mapping, $\delta_{D_i} : sp(Pred(D_i)) \rightarrow sp(D_i)$. That is, a policy maps each instantiation of the informational predecessors of the decision variable to an action. A straightforward way to represent a strategy defined in this way is by a sequence of tables, where each table represents a policy. Note that the table may not need to have a dimension for every variable that is an informational predecessor of D_i. It only needs to consider variables that are relevant, which means they affect the choice of action.

Expected utility of a strategy. The same strategy can also be represented by an ordered set of degenerate conditional probability distributions, $P_\Delta = (P_{\delta_{D_1}}, \ldots, P_{\delta_{D_n}})$, one for each decision variable D_i, where each conditional probability distribution is defined as follows:

$$P_{\delta_{D_i}}(D_i = d_i | Pred(D_i)) = \begin{cases} 1 \text{ if } d_i = \delta_{D_i}(Pred(D_i)), \\ 0 \text{ otherwise.} \end{cases} \tag{3}$$

A strategy Δ defined in this alternative way induces a joint probability distribution over the variables $\mathbf{C} \cup \mathbf{D}$ of the problem, defined as

$$P_\Delta(\mathbf{C}, \mathbf{D}) = \prod_{D \in \mathbf{D}} P_{\delta_D}(D | Pred(D)) \prod_{C \in \mathbf{C}} P(C | pa(C)), \tag{4}$$

which allows the expected utility of a strategy Δ to be defined as

$$EU(\Delta) = \sum_{\mathbf{C} \cup \mathbf{D}} \Big[P_\Delta(\mathbf{C}, \mathbf{D}) U(\mathbf{C}, \mathbf{D}) \Big], \tag{5}$$

where $U(\mathbf{C}, \mathbf{D}) = \sum_{R \in \mathbf{R}} R(pa(R))$ is the global utility function of the decision problem, that is, it is the sum of the local reward functions. In words, Equation (5) means that the expected utility of a strategy is the sum of the probability of each joint assignment to the variables in $\mathbf{C} \cup \mathbf{D}$ multiplied by its utility.

Let $\mathbf{\Delta}$ denote the set of all deterministic strategies for an influence diagram. An optimal strategy is defined as $\Delta^* = \arg\max_{\Delta \in \mathbf{\Delta}} EU(\Delta)$, and the maximum expected utility (MEU) for the decision problem is $EU(\Delta^*)$. In Sections 3 and 4, we describe variable elimination algorithms that compute an optimal strategy and its expected utility.

2.3 Representation of a Strategy as a Graph

In this paper, we introduce an alternative representation of a strategy as a special kind of directed acyclic graph. As pointed out in the introduction, this representation can be more compact, since it allows redundancies and unreachable scenarios to be more easily removed, and thus it "compresses" the graph and makes it easier to interpret. Of course, the following definition of a strategy graph includes a strategy tree as a special case.

Definition 1. *A* strategy graph *represents a strategy for an influence diagram by a directed acyclic graph with two kinds of nodes, observation nodes and decision nodes, where the following conditions hold.*

- *An observation node is associated with an observed chance variable and has one or (usually) more outgoing arcs that are labeled by states of the variable. Each state labels at most one arc, and corresponds to an observation. An outgoing arc leads to either another observation node or a decision node.*
- *A decision node is associated with a decision variable, and has a single outgoing arc that is labeled by a state of the decision variable, which corresponds to an action. The outgoing arc leads to either an observation node, another decision node, or* nil*. Its successor is* nil *if the decision node is a "terminal node" of the graph, which means it is the last decision node on a path originating from a source node of the graph.*

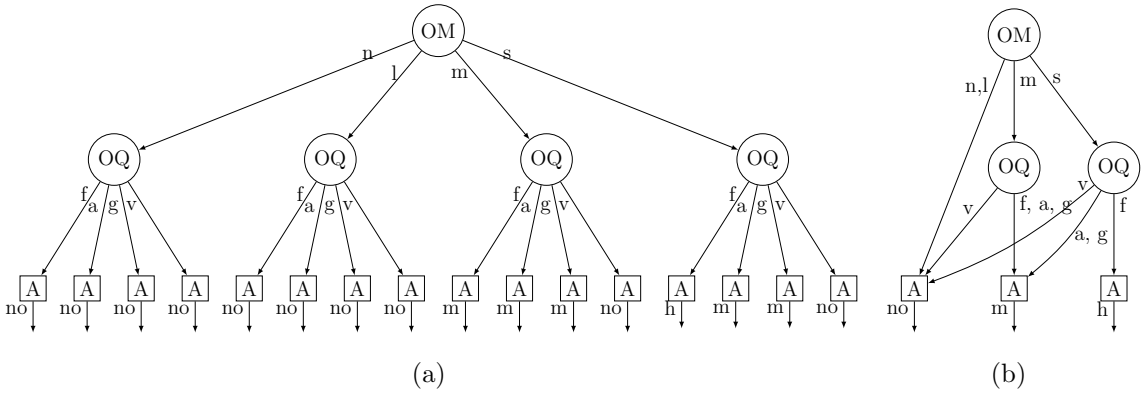


Figure 2: (a) Optimal strategy tree for mildew problem and (b) equivalent strategy graph.

- *All ancestor nodes of a decision node must be associated with a variable that is an informational predecessor of the decision variable associated with the decision node. That is, the order in which variables are considered on any path from a source node to a terminal node of the strategy graph must respect the information precedence constraints of the problem.*
- *More than one node of a strategy graph can be associated with the same variable of the influence diagram, in which case the nodes are distinguished by their histories, that is, their ancestors in the graph.*
- *A source node of a strategy graph can be a decision or an observation node, and there can be more than one source node, in which case there is a rule for selecting the source node from which to start execution of the strategy. For example, the starting source node can be selected based on the start state of the problem, or a probability distribution over the possible start states.*

Because every terminal decision node has a single outgoing arc corresponding to the choice of an action, with *nil* as a successor, a strategy graph defined in this way is not quite a graph. Therefore, to ensure it is a graph, we assume that the outgoing arc from a terminal decision node leads to a global sink node for *nil* that represents termination of the strategy.

Obviously, a strategy graph can be displayed in different ways that affect how easily a user can interpret the strategy it represents. When we display strategy graphs in this paper, we distinguish between decision and observation nodes by shape. By convention, we represent a decision node by a rectangle and an observation node by an oval. In addition, for simplicity, the global sink node of the graph is not shown.

Example: Mildew treatment. Figure 2a shows an optimal strategy tree for the mildew problem, which has one terminal decision node for each instantiation of the two observed chance variables. Each node is labeled by the variable it is associated with and each arc is labeled by a state of the variable.

Figure 2b shows an equivalent strategy graph. In the rest of the paper, we show how to modify a variable elimination algorithm so that it constructs a strategy graph, and how to compress a strategy graph so that it is as compact and easy to understand as possible.

3. Variable Elimination

The representation of a strategy as a graph can be adopted by any algorithm for solving influence diagrams. In this section, we show how to modify the classic variable elimination approach so that it constructs a strategy graph. We consider the simple variable elimination algorithm described by Jensen and Nielsen (2007, pp. 353–5). Closely-related algorithms can be modified in a similar way (e.g., Tatman & Shachter, 1990; Shenoy, 1992; Ndilikilikesha, 1994; Jensen et al., 1994; Dechter, 2000; Luque & Díez, 2010).

Notation. Recall that we use bold upper-case letters such as $\mathbf{X}$ to represent sets of variables, that is, joint variables. In the special case of an empty set of variables, we adopt the standard convention that its state space consists of a single special state, denoted λ. Thus $sp(\emptyset) = \{\lambda\}$ and $|sp(\emptyset)| = 1$. In addition, when $\mathbf{X} \cap \mathbf{Y} = \emptyset$, we let $(\mathbf{X}, \mathbf{Y})$ denote a joint variable, and $(\mathbf{x}, \mathbf{y})$ a state of this joint variable. Note that $(\lambda, \mathbf{y}) = \mathbf{y}$.

If $\mathbf{X} \subseteq \mathbf{Y}$, then $\mathbf{y}^{\downarrow \mathbf{X}}$ denotes the *projection* of the state $\mathbf{y}$ onto the state space for $\mathbf{X}$, which means that values of variables in $\mathbf{Y}$ that are not also in $\mathbf{X}$ are ignored. Thus $\mathbf{y}^{\downarrow \mathbf{X}}$ is a state of $\mathbf{X}$. If $\mathbf{X} = \emptyset$, then $\mathbf{y}^{\downarrow \mathbf{X}} = \lambda$.

A *potential* over a set of variables $\mathbf{X}$ is a function that maps each instantiation $\mathbf{x}$ of $\mathbf{X}$ to a real number. A *probability potential*, denoted $\phi(\mathbf{X})$, is further restricted to be a non-negative function that is not identically zero. It generalizes the concept of a probability distribution to contexts where it is not necessarily normalized. A *conditional probability potential* for $\mathbf{X}$ given $\mathbf{Y}$, denoted $\phi(\mathbf{X}|\mathbf{Y})$, is a probability potential over $\mathbf{X}$ conditional on disjoint $\mathbf{Y}$. A *utility potential*, denoted $\psi(\mathbf{X})$, generalizes the concept of an expected utility function to contexts involving multiplication by a probability potential.

For convenience, we often abuse notation by not specifying the variables in a potential's domain.[2] For example, we often write $\phi(\mathbf{X})$ as simply ϕ. Note that we let the operator *dom* return the variables in the domain of a potential, so that $dom(\phi(\mathbf{X})) = \mathbf{X}$ and $dom(\phi) = \mathbf{X}$.

3.1 Algorithm

As shown by several authors (e.g., Jensen & Nielsen, 2007, pp. 350-2), a variable elimination algorithm solves, or "evaluates," an influence diagram by solving the following equation for the maximum expected utility (MEU) of a strategy:

$$MEU = \sum_{\mathbf{y}_1 \in sp(\mathbf{Y}_1)} \max_{d_1 \in sp(D_1)} \cdots \sum_{\mathbf{y}_n \in sp(\mathbf{Y}_n)} \max_{d_n \in sp(D_n)} \sum_{\mathbf{x} \in sp(\mathbf{X})} \left[\prod_{\phi \in \Phi} \phi \sum_{\psi \in \Psi} \psi \right]. \tag{6}$$

In this equation, both the product and the sum inside the square brackets are over the potentials in a set of potentials, where $\Phi = \{P(C|pa(C))|C \in \mathbf{C}\}$ denotes the set of probability potentials of the influence diagram and $\Psi = \{R(pa(R)|R \in \mathbf{R}\}$ denotes the set of utility potentials. The outer sums and maximizations are over the states of the variables of the influence diagram, and the order of outer sums and maximizations reflects the order of information precedence.

2. We follow Jensen and Nielsen (2007) in referring to the set of variables $\mathbf{X}$ associated with a potential $\psi(\mathbf{X})$ as the domain of the potential. It is also common in the literature to refer to this set of variables as the *scope* of the potential (e.g., Koller & Friedman, 2009; Dechter, 2019), and to refer to the set of possible instantiations of $\mathbf{X}$ as the domain of the potential.

3.1.1 Operations on Potentials

The operations performed by variable elimination in solving Equation (6) are of two types: combination and marginalization.

Combination operations. Two potentials are *combined* by the operations of addition, multiplication, or division, and the domain of the resulting potential is the union of the domains of the combined potentials. Thus when the potentials $\psi(\mathbf{X})$ and $\psi'(\mathbf{Y})$ are combined, the domain of the resulting potential $\psi''(\mathbf{Z})$ is $\mathbf{Z} = \mathbf{X} \cup \mathbf{Y}$. Combination operations are performed pointwise, as follows.

- The *sum* of two utility potentials, $\psi(\mathbf{X})$ and $\psi'(\mathbf{Y})$, is a utility potential, $\psi''(\mathbf{Z}) = \psi(\mathbf{X}) + \psi'(\mathbf{Y})$, defined so that $\psi''(\mathbf{z}) = \psi(\mathbf{z}^{\downarrow \mathbf{X}}) + \psi'(\mathbf{z}^{\downarrow \mathbf{Y}})$, for each $\mathbf{z} \in sp(\mathbf{Z}) = sp(\mathbf{X} \cup \mathbf{Y})$. (Recall that $\downarrow$ denotes the projection operation, defined above.)

- The *product* of two probability potentials, $\phi(\mathbf{X})$ and $\phi'(\mathbf{Y})$, is a probability potential, $\phi''(\mathbf{Z}) = \phi(\mathbf{X}) \cdot \phi'(\mathbf{Y})$, defined so that $\phi''(\mathbf{z}) = \phi(\mathbf{z}^{\downarrow \mathbf{X}}) \cdot \phi'(\mathbf{z}^{\downarrow \mathbf{Y}})$, for each $\mathbf{z} \in sp(\mathbf{Z}) = sp(\mathbf{X} \cup \mathbf{Y})$. The product of a utility potential and a probability potential is defined similarly, except the result is a utility potential.

- The *division* operation is similarly performed pointwise. Note that it is only used to normalize probabilities.

Marginalization operations. Marginalization operations eliminate one or more variables from the domain of a potential. In the following definitions, we let φ denote a potential that could be either a probability or a utility potential.

- The elimination by *max-marginalization* of a decision variable D from potential φ creates a new potential φ' over the variables $dom(\varphi) \backslash \{D\}$, defined as $\varphi' = \max_D \varphi$. That is, given a potential $\varphi(D, \mathbf{X})$, elimination of the variable D by max-marginalization creates a potential $\varphi'(\mathbf{X})$, which is defined so that for each $\mathbf{x} \in sp(\mathbf{X})$:

$$\varphi'(\mathbf{x}) = \max_{d \in sp(D)} \varphi(d, \mathbf{x}). \tag{7}$$

- The elimination by *sum-marginalization* of a chance variable C from a potential φ creates a new potential φ' over the variables $dom(\varphi) \backslash \{C\}$, defined as $\varphi' = \sum_C \varphi$. For example, given a potential $\varphi(C, \mathbf{X})$, elimination of C by sum-marginalization creates a new potential $\varphi'(\mathbf{X})$ defined so that for each $\mathbf{x} \in sp(\mathbf{X})$:

$$\varphi'(\mathbf{x}) = \sum_{c \in sp(C)} \varphi(c, \mathbf{x}). \tag{8}$$

- The *restriction* operation instantiates one or more of the variables in a potential. For example, the restriction $\varphi^{R(Y=y)}(\mathbf{X})$ of a potential $\varphi(Y, \mathbf{X})$ is a potential over $\mathbf{X}$ such that for each $\mathbf{x} \in sp(\mathbf{X})$:

$$\varphi^{R(Y=y)}(\mathbf{x}) = \varphi(y, \mathbf{x}). \tag{9}$$

Algorithm 1: Variable elimination algorithm.

Input: Variables $\mathbf{V} = \mathbf{C} \cup \mathbf{D}$, probability potentials $\Phi = \{P(C|pa(C))|C \in \mathbf{C}\}$ and utility potentials $\Psi = \{R(pa(R)|R \in \mathbf{R}\}$

Output: An optimal strategy, Δ, and its expected utility (MEU)

$\Delta \leftarrow \emptyset$ // strategy is initialized
for $i \leftarrow 1$ **to** $|\mathbf{V}|$ **do** // i is index of elimination step
  Select next variable V to eliminate
  // Process probability potentials
  $\Phi_V \leftarrow \{\phi \in \Phi | V \in dom(\phi)\}$ // get set of relevant probability potentials
  $\phi_V \leftarrow \prod_{\phi \in \Phi_V} \phi$ // compute product of relevant probability potentials
  if *V is a chance variable* **then**
    $\phi_i \leftarrow \sum_V \phi_V$ // eliminate V by sum-marginalization
  else if *V is a decision variable* **then**
    $\phi_i \leftarrow \max_V \phi_V$ // eliminate V by max-marginalization
  $\Phi \leftarrow (\Phi \backslash \Phi_V) \cup \{\phi_i\}$ // update set of probability potentials
  // Process utility potentials
  $\Psi_V \leftarrow \{\psi \in \Psi | V \in dom(\psi)\}$ // get set of relevant utility potentials
  $\psi_V \leftarrow \sum_{\psi \in \Psi_V} \psi$ // compute sum of relevant utility potentials
  if *V is a chance variable* **then**
    $\phi_{cond} \leftarrow \phi_V / \phi_i$ // normalize conditional probability of V
    $\psi_i \leftarrow \sum_V \phi_{cond} \cdot \psi_V$ // sum-marginalize V to compute expected value
  else if *V is a decision variable* **then**
    $\psi_i \leftarrow \max_V \psi_V$ // max-marginalize V to compute maximum value
    $\delta_V \leftarrow \arg\max_V \psi_V$ // compute optimal policy for decision variable
    $\Delta \leftarrow \Delta \cup \{\delta_V\}$ // add policy to strategy
  $\Psi \leftarrow (\Psi \backslash \Psi_V) \cup \{\psi_i\}$ // update set of utility potentials
end
$MEU \leftarrow \sum_{\psi \in \Psi} \psi$ // after all variables are eliminated, utility potentials are scalars
return (Δ, MEU) // Δ is optimal strategy

3.1.2 Pseudocode

Algorithm 1 gives pseudocode for the algorithm. It eliminates variables in the reverse of the partial temporal order given by Equation (1). When more than one elimination order is possible, a heuristic can be used to select an order that is likely to lead to good performance. For example, it is common to use a heuristic that greedily tries to minimize the size of generated potentials (Cabañas, Cano, Gómez-Olmedo, & Madsen, 2013).

Identify relevant potentials. Once a variable V is selected for elimination, the next step is to identify the potentials that include V in their domain, and thus need to be replaced by equivalent potentials that do not. We call these potentials the *relevant potentials*. We call the variables in the union of the domains of the relevant potentials, not including the variable V itself, the *relevant variables*. The relevant variables are the variables in the domain of the new utility potential ψ_i that is generated at the end of an elimination step.

Process probability potentials. Once all relevant probability potentials are identified, they are multiplied to create the probability potential ϕ_V. Eliminating the variable V from ϕ_V creates the probability potential ϕ_i. If V is a chance variable, it is eliminated by sum-marginalization. If it is a decision variable, it can be eliminated by max-marginalization, as shown in Line 10 of Algorithm 1, which follows the description of the algorithm given by Jensen and Nielsen (2007, pp. 353–5). However, when a decision variable is eliminated, it is *d-separated* from its predecessors and any successors have already been eliminated. Thus it cannot have an effect on the value of any probability potential, even if it is in its domain, and the decision variable can be more simply eliminated by just projecting the potential onto the remaining variables (e.g., Luque et al., 2017).

Process utility potentials. Once all relevant utility potentials are identified, their sum ψ_V is computed. If V is a chance variable, ψ_V is multiplied by the normalized probability potential $\phi_{cond} = \phi_V/\phi_i$, and then V is eliminated by sum-marginalization. If V is a decision variable, it is simply eliminated from ψ_V by max-marginalization. In either case, ψ_i denotes the resulting utility potential. When a decision variable is eliminated, a policy δ_V is also computed that records the maximizing action for each instantiation $\mathbf{h}$ of the variables $\mathbf{H}$ in the domain of ψ_i, so that $\delta_V(\mathbf{h}) = \arg\max_{v \in sp(V)} \psi_V(v, \mathbf{h})$.

Return solution. Once all variables are eliminated, the algorithm returns the MEU value of Equation (6), as well as an optimal strategy Δ represented by a sequence of policies.

3.2 Strategy Graph Construction

We next describe an alternative approach to strategy representation and construction for this algorithm. Except for superficial differences of terminology and notation, the approach we describe is the same as a recent proposal of Luque et al. (2017). The only significant difference is that they describe how to modify the algorithm so that it represents a strategy as a tree, called a *strategy tree*, whereas we consider how to construct a strategy graph.[3]

In constructing a strategy graph, our approach associates each utility potential $\psi(\mathbf{X})$ with a companion function, $s_\psi : sp(\mathbf{X}) \to \mathcal{N} \cup \{nil\}$, that we call a *strategy node function*, where $\mathcal{N}$ denotes the set of nodes of the strategy graph. The strategy node function $s_\psi(\mathbf{X})$ maps each instantiation of the variables in the domain of the utility potential ψ to a node of the strategy graph, or to *nil*, if there is no corresponding node.

3.2.1 Initialization and Elimination of Unobserved Chance Variables

For a utility potential associated with a reward node of an influence diagram, the associated strategy node function maps each instantiation of the variables in its domain to *nil*.

When an unobserved chance variable is eliminated, the strategy node function generated also maps every instantiation of the variables in its domain to *nil*, since strategy graph construction does not begin until a decision variable is eliminated, and all unobserved chance variables must be eliminated before any decision variable is eliminated.

3. Although influenced by many ideas in the excellent paper of Luque et al. (2017), we introduced the concept of a strategy graph in a paper published a year before (Hansen, Shi, & Khaled, 2016), inspired by the concept of a policy graph for a finite-horizon POMDP (Kaelbling, Littman, & Cassandra, 1998).

3.2.2 Elimination of Decision Variable by Max-Marginalization

Elimination of a decision variable from a utility potential by max-marginalization results in the creation of decision nodes that are added to the strategy graph. For the first decision variable eliminated, the decision nodes created are the terminal nodes of the strategy graph.

Let $\psi_D(D, \mathbf{H})$ denote a utility potential from which the variable D is eliminated. Because the classic variable elimination algorithm eliminates all unobserved chance variables first, all variables in the domain of this potential must be observed, and so they are denoted $\mathbf{H}$ because they represent the relevant history of the process. Eliminating D generates a utility potential $\psi_i(\mathbf{H})$ and strategy node function $s_{\psi_i}(\mathbf{H})$, where for each instantiation $\mathbf{h}$ of $\mathbf{H}$, the maximizing decision is $d^* = \arg\max_{d \in sp(D)} \psi_D(d, \mathbf{h})$, the maximum utility is $\psi_i(\mathbf{h}) = \psi_D(d^*, \mathbf{h})$, and $s_{\psi_i}(\mathbf{h})$ is set equal to a newly-created decision node., which has a single outgoing arc labeled by the maximizing decision d^*, with successor node $s_{\psi_D}(d^*, \mathbf{h})$.

3.2.3 Elimination of Observed Chance Variable by Sum-Marginalization

Elimination of an observed chance variable from a utility potential by sum-marginalization results in creation of observation nodes that are added to the strategy graph.

Let $\psi_C(C, \mathbf{H})$ denote the utility potential from which the observed chance variable C is eliminated. Its elimination creates a utility potential $\psi_i(\mathbf{H})$ and strategy node function $s_{\psi_i}(\mathbf{H})$, where for each instantiation $\mathbf{h}$ of $\mathbf{H}$, we have $\psi_i(\mathbf{h}) = \sum_{c \in sp(C)} \psi_C(c, \mathbf{h})$, and $s_{\psi_i}(\mathbf{h})$ is set equal to a newly-created observation node that has an outgoing arc for each observation $c \in sp(C)$, with successor node $s_{\psi_C}(c, \mathbf{h})$.

3.2.4 Graph Simplification When Observed Chance Variable is Eliminated

Luque et al. (2017) propose some rules for simplifying a strategy tree generated by this procedure that we adopt for simplifying a strategy graph. The rules apply to observation nodes that are added to a strategy graph when an observed chance variable C is eliminated.

Removal of zero-probability observation branches. If $P(c|\mathbf{h}) = 0$ for observation $c \in sp(C)$ and history $\mathbf{h} \in sp(\mathbf{H})$, which means the conditional probability of observing c after history $\mathbf{h}$ is zero, the observation node does not include an outgoing arc for observation c. In short, zero-probability branches are not included in the strategy graph.

Removal of redundant observation nodes. If an observation node has the same successor node for every observation $c \in sp(C)$ for which $P(c|\mathbf{h}) > 0$, it is redundant, and it does not need to be included in the strategy graph. To remove it, $s_\psi(\mathbf{h})$ is set equal to the successor node of the redundant node, which is $s_\psi(\mathbf{h}, c)$, for some c where $P(c|\mathbf{h}) > 0$.

Merging of observation branches that reflect irrelevant distinctions. If two or more outgoing arcs from an observation node lead to the same successor node, the arcs can be merged into a single arc, which is labeled by the set of observations.

Removal of unreachable decision and observation nodes. Because a strategy graph is created bottom-up from its sink to its source nodes, it may turn out that some, or many, of its nodes are only reachable from a source node via a suboptimal or non-selected action, or via a zero-probability observation. These unreachable nodes can be removed from a strategy graph after it is created.

3.2.5 Addition of Utility Potentials

The sum of utility potentials $\psi(\mathbf{X})$ and $\psi'(\mathbf{Y})$ is a new utility potential $\psi''(\mathbf{Z})$, with domain $\mathbf{Z} = \mathbf{X} \cup \mathbf{Y}$. If the strategy node functions $s_\psi(\mathbf{X})$ and $s_{\psi'}(\mathbf{Y})$ are both mappings to *nil*, the new strategy node function $s_{\psi''}(\mathbf{Z})$ also maps every instantiation of the variables in its domain to *nil*. If one strategy node function, $s_\psi(\mathbf{X})$, maps every instantiation of the variables in its domain to *nil*, while the other, $s_{\psi'}(\mathbf{Y})$, is a mapping to strategy nodes, the new strategy node function $s_{\psi''}(\mathbf{Z})$ is defined so that for each instantiation $\mathbf{z}$ of $\mathbf{Z}$, we have $s_{\psi''}(\mathbf{z}) = s_{\psi'}(\mathbf{z}^{\downarrow\mathbf{Y}})$, which preserves the strategy associated with the utility potential $\psi'(\mathbf{Y})$.

If both strategy node functions, $s_\psi(\mathbf{X})$ and $s_{\psi'}(\mathbf{Y})$, are mappings to strategy nodes, the new strategy node function, $s_{\psi''}(\mathbf{Z})$, is created by an operation we call *concatenation*, which is defined for each instantiation $\mathbf{z}$ of $\mathbf{Z}$ as follows: for every *terminal* decision node of the strategy graph rooted at $s_\psi(\mathbf{z}^{\downarrow\mathbf{X}})$, its successor node is set to $s_{\psi'}(\mathbf{z}^{\downarrow\mathbf{Y}})$, which is the root node of the other strategy graph. Essentially, concatenating two strategy graphs means combining them so that one is executed after the other. Therefore, the order in which strategy graphs are concatenated matters, and it must be the reverse of the order in which their corresponding utility potentials are generated in order to ensure that the ordering of actions along each branch of the strategy graph is consistent with the total ordering of decision nodes in the influence diagram. As a simple example, Figure 3e shows the result of concatenating the strategy graph in Figure 3d with the strategy graph in Figure 3b.

3.2.6 Multiplication of a Probability Potential by a Utility Potential

Multiplication of a probability potential $\phi(\mathbf{X})$ by a utility potential $\psi'(\mathbf{Y})$ generates a utility potential $\psi''(\mathbf{Z})$ with domain $\mathbf{Z} = \mathbf{X} \cup \mathbf{Y}$. For a given instantiation $\mathbf{z}$ of $\mathbf{Z}$, the strategy node function associated with the new utility potential is defined as follows: $s_{\psi''}(\mathbf{z}) = s_{\psi'}(\mathbf{z}^{\downarrow\mathbf{Y}})$. That is, multiplication of a probability potential by a utility potential does not change the strategy associated with the utility potential.

3.2.7 Example

To illustrate this procedure for strategy graph construction, we consider the simple influence diagram shown in Figure 3a, which is used by Luque et al. (2017) to illustrate how they modify the variable elimination algorithm to construct a strategy tree. For this example, all variables are Boolean, and the only possible elimination order is: D_2, D_1, and then Y_1.

Elimination of the decision variable D_2 generates a utility potential $\psi_1()$ and strategy node function $s_{\psi_1}()$ with an empty domain, where $s_{\psi_1}()$ maps the special state λ to the decision node shown in Figure 3b. The outgoing arc from the decision node is labeled by the action $+d = \arg\max_{d_2 \in D_2} R_2(d_2)$, where we adopt the notation $+d$ from Luque et al.

Elimination of the decision variable D_1 generates a utility potential $\psi_2(Y_1)$ and strategy node function $s_{\psi_2}(Y_1)$. For each of the two possible values y_1 of the variable Y_1, a decision node is created with an outgoing arc labeled by the optimizing action, which is $\arg\max_{d_1 \in D_1} R_1(d_1, y_1)$. The two new decision nodes are shown in Figure 3c.

Elimination of the observed chance variable Y_1 generates a utility potential $\psi_3()$ and corresponding strategy node function $s_{\psi_3}()$, which maps the special state λ to the observation node at the root of the strategy graph shown in Figure 3d. The observation node has one outgoing arc for each of the two possible values of the observed chance variable Y_1.

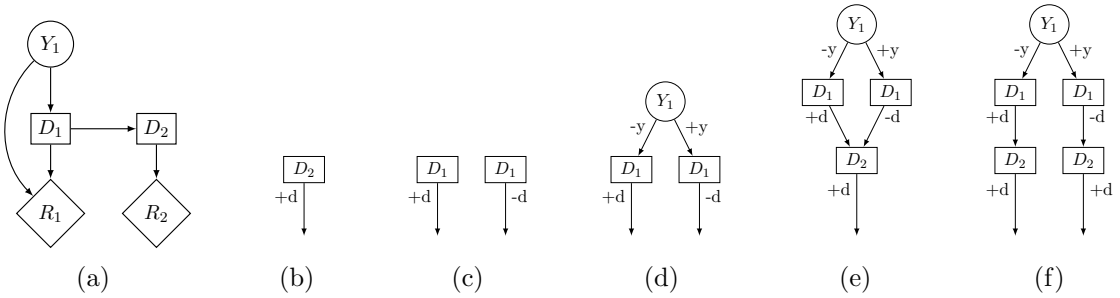


Figure 3: (a) Influence diagram from Luque et al. (2017); (b) strategy graph for $s_{\psi_1}()$, created when D_2 is eliminated; (c) strategy graph for $s_{\psi_2}(Y_1)$, created when D_1 is eliminated; (d) strategy graph for $s_{\psi_3}()$, created when Y_1 is eliminated; (e) final strategy graph created by concatenating the strategy graphs for $s_{\psi_3}()$ and $s_{\psi_1}()$; and (f) equivalent strategy tree.

After all three variables are eliminated, two scalar-valued utility potentials, ψ_1 and ψ_3, remain in the set Ψ, together with their corresponding strategy graphs. Adding the two utility potentials results in concatenation of their strategy graphs. Thus each outgoing arc from the strategy graph shown in Figure 3d leads to the root node of the strategy graph shown in Figure 3b, with the concatenated strategy graph shown in Figure 3e.

This last step of our strategy graph construction procedure is the only step that differs from the procedure described by Luque et al. (2017). For this example, they show that their procedure constructs the strategy tree shown in Figure 3f, with two identical leaf nodes, instead of the equivalent strategy graph shown in Figure 3e. If not for their claim that their procedure *always* constructs a strategy tree, and their use of this example to illustrate it, we would consider our procedure for strategy graph construction to be the same as theirs.

3.3 Coalescence

The procedure for strategy graph construction described above generates a strategy graph that is *not* a tree in some cases, such as the simple example described in Section 3.2.7. Much more often, however, it generates a strategy tree, and the strategy tree contains many duplicate subtrees that could be merged, but are not.

In the literature on influence diagrams, the term *coalescence* refers to the simplification of a strategy tree (or graph) by merging duplicate subtrees (or, more generally, duplicate subgraphs). From now on, it will be helpful to distinguish between two forms of coalescence. In the form of coalescence illustrated by the strategy graph shown in Figure 3e, duplicate subtrees are never actually created, and so they are only implicitly merged. Thus we call this form of coalescence *implicit coalescence*. But there are cases where the procedure for strategy graph construction described above generates a strategy tree, and yet an additional step can be added to the procedure that explicitly converts the strategy tree to a smaller, equivalent strategy graph. We call this second form of coalescence *explicit coalescence*. Below, we consider each form of coalescence in turn and illustrate it by example.

3.3.1 Implicit Coalescence

The following example more clearly illustrates what we mean by implicit coalescence.

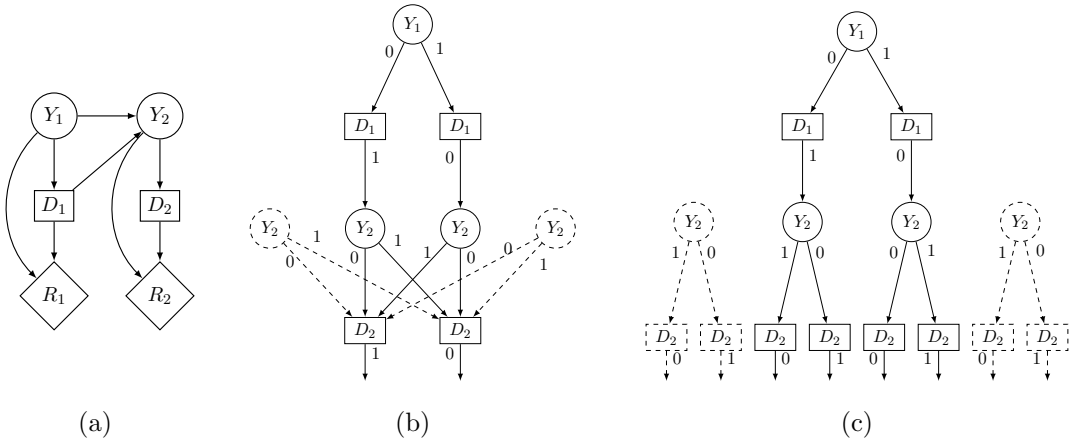


Figure 4: (a) Influence diagram for a two-stage MDP, (b) strategy graph, and (c) equivalent strategy tree. Dashed lines indicate unreachable nodes and arcs, which can be removed.

Example: Completely observable Markov decision process. Figure 4a shows a simple influence diagram for a two-stage completely observable Markov decision process (MDP). For convenience, we assume that all variables are Boolean. Figure 4b shows the strategy graph constructed by the procedure described in Section 3.2. It has the same structure regardless of the strategy, and so the labeling of arcs by 0 or 1 is arbitrary.

The only order in which the variables can be eliminated is: D_2, Y_2, D_1, then Y_1. For each eliminated variable, a node is added to the strategy graph for each instantiation of the variables in the domain of the utility potential created by eliminating the variable. For the first eliminated variable, D_2, the utility potential created is $\psi_1(Y_2)$. Since Y_2 has just two possible instantiations, the strategy graph in Figure 4b has two terminal decision nodes.

For the second eliminated variable, Y_2, the utility potential created is $\psi_2(Y_1, D_1)$. There are four possible instantiations of these two variables, and so four observation nodes are generated. But only two are reachable by following an optimal strategy. Unreachable nodes, which are shown by dashed circles and arcs, can be removed from the final strategy graph.

When the variable D_1 is eliminated, the utility potential $\psi_3(Y_1)$ is generated, and two decision nodes are added to the strategy graph. Finally, when Y_1 is eliminated, the generated utility potential $\psi()$ has an empty domain, and a single source node is added to the strategy graph. Note that the strategy graph constructed by this procedure and shown in Figure 4b is *not* a tree. For comparison, the equivalent strategy tree is shown in Figure 4c.

The construction by variable elimination of a smaller, equivalent strategy graph in which duplicate subtrees are *implicitly* merged not only makes a strategy easier to interpret, it reflects the improved efficiency of the algorithm. Like the value iteration algorithm for completely observable MDPs, variable elimination solves a completely observable MDP in low-order polynomial time in the number of stages because it never generates a utility potential with more than two variables in its domain, no matter how many stages of the problem it solves. If a strategy is represented by a strategy tree, however, the size of the tree grows exponentially in the number of stages. It follows that representing a strategy by a tree instead of a more compact graph can sometimes increase the time and space complexity of the variable elimination algorithm by an exponential factor!

Dynamic programming and implicit coalescence. To better understand implicit coalescence, we must consider its relationship to dynamic programming. The classic variable elimination algorithm solves a dynamic programming recurrence with one subproblem for each pair of eliminated variable and instantiation of variables in the domain of the resulting utility potential. For each subproblem it solves when eliminating a decision or observed chance variable, it adds a node to the strategy graph.

Variable elimination *implicitly* constructs a strategy graph, and not a strategy tree, when not all un-eliminated variables are relevant when a decision or observed chance variable is eliminated, that is, when part of the history is irrelevant. The fewer variables are relevant, and thus in the domain of the generated utility potential, the fewer subproblems are in the dynamic programming recurrence, and so the fewer nodes are added to the strategy graph.

In an insightful discussion of this form of coalescence, Tatman (1986, chp. 4) relates it to Bellman's principle of optimality – the underlying principle of dynamic programming. He points out that coalescence occurs "under certain conditions involving conditional independence among the variables in the model, the separability of the value function or both." That is, it occurs when not all un-eliminated variables are relevant in an elimination step, either because they have no effect on the state (due to conditional independence) or because they have no effect on utility (due to separability of the utility function).

For example, consider the two-stage completely observable MDP represented by the influence diagram in Figure 4a. When the decision variable D_2 is eliminated, the values of the variables Y_1 and D_1 are not relevant because (i) the decision variable D_2 is conditionally independent of the variables Y_1 and D_1 given observation of the state of Y_2, and (ii) Y_1 and D_1 are not in the domain of a reward function influenced by D_2. Note that the irrelevance of the variables Y_1 and D_1 for the decision variable D_2 given observation of the state of the variable Y_2 corresponds to the irrelevance of previous history when the current state of the Markovian process is observed. That is, in solving this influence diagram, the variable elimination algorithm leverages the same Markov property that is leveraged by the value iteration algorithm for solving the corresponding completely observable MDP.

(For the influence diagram in Figure 3b, the variables Y_1 and D_1 are similarly irrelevant when D_2 is eliminated, and so a strategy graph is constructed instead of a tree.)

3.3.2 Explicit Coalescence

The classic variable elimination algorithm is effective in leveraging variable independence for implicit coalescence (and speedup) when it solves influence diagrams that represent completely observed problems, as these two simple examples illustrate. Unfortunately, it is *not* effective in leveraging variable independence when solving influence diagrams that have unobserved chance variables, and therefore represent *partially observed* problems. For such problems, it typically constructs a strategy tree, with no implicit coalescence at all.

Example: Mildew treatment problem. Recall the mildew problem described at the end of Section 2.1 and represented by the influence diagram in Figure 1. The classic variable elimination algorithm must first eliminate all four unobserved chance variables: H, M^*, Q, and M. Doing so creates the utility potential $\psi_4(A, OQ, OM) = \sum_{H,M^*,Q,M} P(H|M^*, Q)$ $P(M^*|A, M)P(Q)P(M)U(H)$, as well as a strategy node function $s_{\psi_4}(A, OQ, OM)$ that maps each instantiation of the variables in its domain to *nil*.

Elimination of the decision variable A then generates the utility potential $\psi_5(OQ, OM) = \max_A(C(A)+\psi_4(A, OQ, OM))$, and a strategy node function $s_{\psi_5}(OQ, OM)$ that maps each instantiation of the variables in its domain to a decision node of the strategy graph. Sixteen decision nodes are created, one for each joint instantiation of the observed variables OQ and OM, or, equivalently, one for each possible history. The decision nodes are the terminal nodes of the strategy tree shown in Figure 2a, with the outgoing arc from each decision node labeled by the maximizing action for the corresponding entry in the utility potential.

Because the utility potential $\psi_5(OQ, OM)$ generated when A is eliminated includes all of the informational predecessors of the decision variable A in its domain, the decision depends on the full history of the process, and there is no implicit coalescence. Without stepping through the rest of the algorithm, it is already easy to see that it generates the strategy tree shown in Figure 2a, instead of the simpler strategy graph shown in Figure 2b.

Algorithm for explicit coalescence. For such problems, we next describe a step that can be added to the variable elimination algorithm to explicitly check for and merge duplicate subgraphs that are not merged implicitly. We call this extra step *explicit coalescence*.

Explicit coalescence is performed whenever a decision or observed chance variable is eliminated, and there are one or more un-eliminated decision and observed chance variables, denoted $\mathbf{H}$, that are relevant in this elimination step, and thus are in the domain of the generated utility potential $\psi(\mathbf{H})$. Duplicate subgraphs are possible in this case because a node added to the strategy graph for instantiation $\mathbf{h}$ of the observed variables $\mathbf{H}$ can be a duplicate of a node already added for another instantiation $\mathbf{h}'$ of $\mathbf{H}$, where *one node is a duplicate of another if they are the root nodes of identical subgraphs*.

We detect and merge duplicate nodes as follows. Before a node is added to the strategy graph for instantiation $\mathbf{h}$ of $\mathbf{H}$, we compare it to every node added to the strategy graph for any instantiation $\mathbf{h}'$ of $\mathbf{H}$ already considered in this elimination step. If it is a duplicate of an existing node, the entry $s_\psi(\mathbf{h})$ in the strategy node function is set equal to the node already in the strategy graph, instead of creating a new node. Two decision nodes are duplicates if their single outgoing arc is labeled by the same action and leads to the same successor node (as identified by the unique index of the node). Two observation nodes are duplicates if every outgoing arc that is labeled by the same observation has the same successor node.

Duplicate subgraphs that are present in a strategy graph represent solutions of distinct subproblems that have different *relevant* histories, corresponding to different instantiations of $\mathbf{H}$. That means they do not represent shared subproblems of the dynamic programming recurrence solved by variable elimination, and explicitly merging these duplicate subgraphs in the way we have described does not result in any speedup. Indeed, it incurs overhead. However, the per-node overhead is bounded by the number of *distinct* nodes of the strategy graph that must be checked for a duplicate, and the number of distinct nodes generated is often *much* less than the number of instantiations of the variables in the domain of the utility potential. For example, consider the last decision variable D_n of an influence diagram, which is the first decision variable eliminated. No matter how many decision and observed chance variables are relevant when D_n is eliminated, and thus are in the domain of the generated utility potential, the number of *distinct* terminal decision nodes of the strategy graph cannot be greater than $|sp(D_n)|$, which is the number of actions for D_n.

There are a couple useful optimizations of this procedure for merging duplicate nodes.

Figure 5: Example of wildcard matching for merging observation nodes. (a) On the left, the two observation nodes have zero-probability branches that are not shown. (b) In the equivalent strategy graph on the right, the two observation nodes are merged.

Tie breaking for decision nodes. When selecting the best action for a decision variable given instantiation **h** of **H**, it is customary to break ties arbitrarily. However, we adopt a different tie-breaking rule to encourage further coalescence. If there is more than one optimal action, and one gives rise to a duplicate node that can be merged with an existing node of the strategy graph, and another does not, we select the optimal action that gives rise to a duplicate node in order to merge these nodes and simplify the strategy graph.

Wildcard matching for observation nodes. When comparing two observation nodes to determine whether they are duplicates, there is an additional opportunity to compress the strategy graph. When one or the other node lacks outgoing arcs corresponding to zero-probability observations, we treat a missing arc as a *wildcard* that can match an outgoing arc for the same observation for the other node, regardless of its successor node. For example, consider the two observation nodes in the strategy graph shown in Figure 5a. Although superficially different, the two observation nodes can be merged to create the equivalent and smaller strategy graph shown in Figure 5b.

Explicit coalescence for the mildew problem. When variable elimination solves the mildew problem *without* performing explicit coalescence, it constructs the strategy tree in Figure 2a. When it performs explicit coalescence, it constructs the smaller and equivalent strategy graph shown in Figure 2b. Table 1 shows how much this strategy graph is simplified by the various techniques we have described. It also shows the same information for the more complex strategy graph constructed for the *Arthronet* problem that we consider next.

	Mildew		Arthronet	
Degree of simplification	Nodes	Arcs	Nodes	Arcs
Generated strategy graph with implicit coalescence only	21	36	149,695	232,638
With suboptimal (un-selected) branches removed	21	36	10,344	15,526
With zero-probability branches removed	21	36	1,141	1,408
With redundant nodes removed	21	36	892	1,159
With duplicate nodes merged (explicit coalescence)	6	10	60	105

Table 1: For the *Mildew* and *Arthronet* influence diagrams, the size of the strategy graphs constructed by variable elimination based on how much simplification is performed.

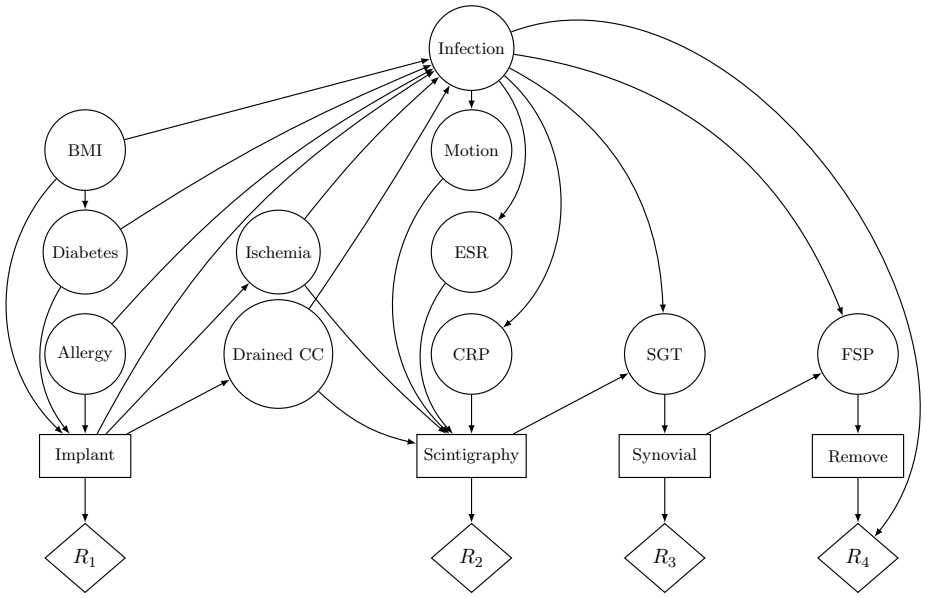


Figure 6: *Arthronet* influence diagram for knee arthroplasty problem.

Example: Arthronet influence diagram. Figure 6 shows an influence diagram created by researchers in the Department of Artificial Intelligence at UNED, in Madrid, Spain, to model a real-world medical decision problem for total knee arthroplasty. It has four decision variables, ten observed chance variables, and just one unobserved chance variable. However, the unobserved variable *Infection* makes the problem partially observable, and most of the conditional independence relations in the influence diagram involve this variable.

Of the ten observed chance variables in the influence diagram, three represent risk factors for the surgery (Allergy, Diabetes, BMI), three represent postoperative observations by the surgeon (Ischemia, Drained CC, Knee motion), and four represent the results of laboratory tests (C-reactive protein, ESR, Sequential Ga67 Tc99, Frozen sections PMN). All variables have two possible values.[4] If the surgeon implants a prosthesis, the next two decisions are whether to perform a scintigraphy or synovial biopsy, which are medical procedures that test for infection. Depending on the test outcomes, the final decision is whether to remove the prosthesis. The optimization problem requires weighing the the effectiveness of medical treatment in QALYs ("quality-adjusted life years") against the treatment cost, where one QALY is valued at 30,000 Euros. The influence diagram is described in detail by León (2011). It is also available in the open-source influence diagram solver *OpenMarkov*.[5]

4. Some chance variables have a dimension of three instead of two, but only because they have a "dummy state" in case the other states of the variable are impossible, as is common in modeling asymmetric problems (Bielza & Shenoy, 1999). A dummy state is added to the chance variables *Ischemia* and *Drained CC* in case the prosthesis is not implanted. It is added to the chance variable *SGT* in case a scintigraphy is not done, and to the chance variable *FSP* in case a synovial biopsy is not done.
5. The influence diagram we solve is the same one described by León (2011) and available in *OpenMarkov* at the URL: http://www.probmodelxml.org/networks/. However, Figure 6 shows this influence diagram with an equivalent but simpler reward structure, with just one reward node for each decision variable.

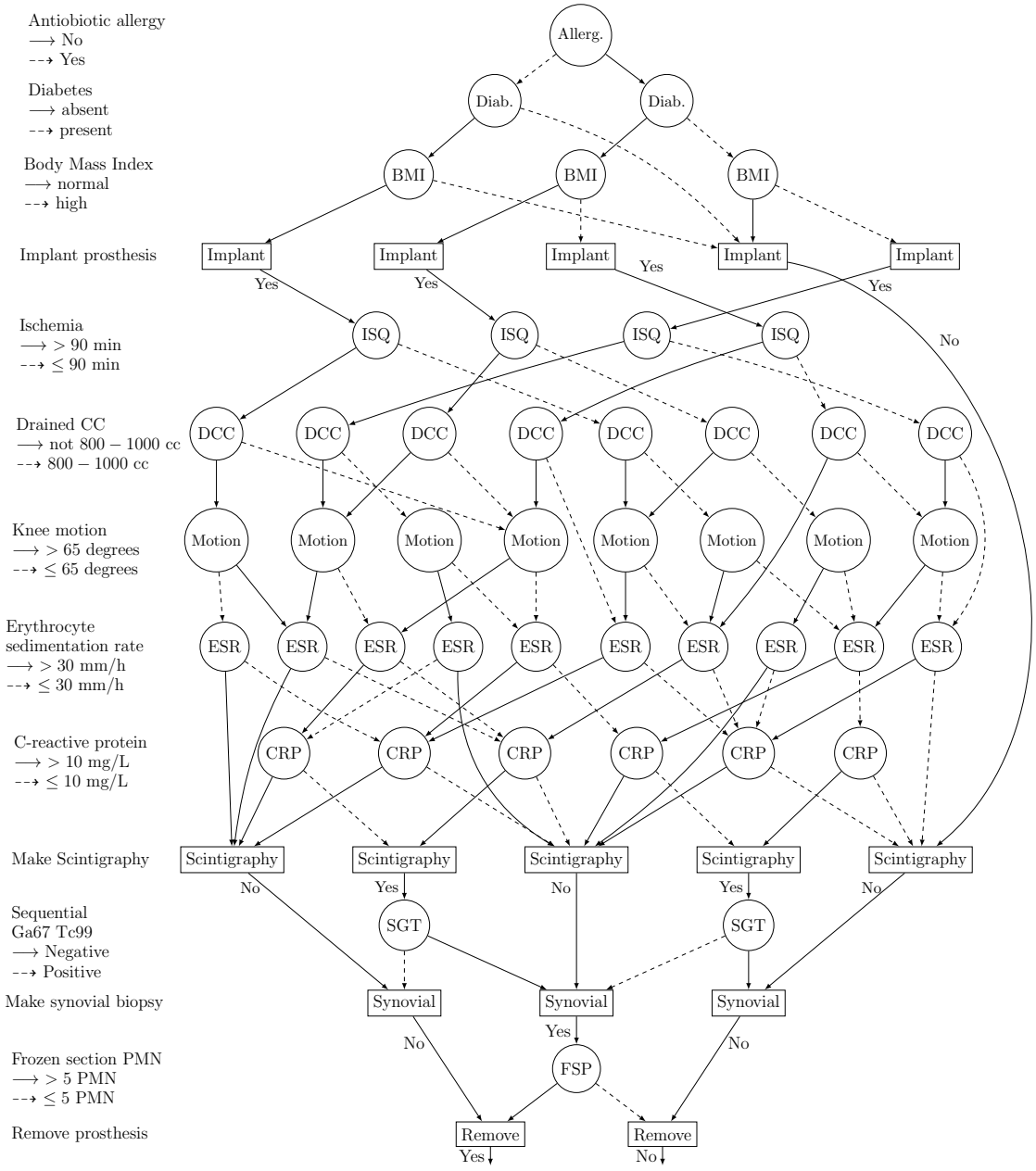


Figure 7: Optimal strategy graph for the Arthronet problem.

Figure 7 shows an optimal strategy graph for this problem after simplification in all the ways we have described. Table 1 shows the effectiveness of the simplification techniques. Without them, the strategy graph construction procedure described above generates a tree with 149,695 nodes and 232,638 arcs! Simplifying the strategy tree by removing unreachable and redundant nodes, as described in Section 3.2.4, reduces its size by three orders of magnitude. In this form, it is the same strategy tree created by the procedure of Luque et al. (2017). Without explicit coalescence, however, it still has 892 nodes and 1159 arcs.

Coalescence of duplicate subgraphs reduces the size of the strategy graph so that it has only 60 nodes and 105 arcs – an order of magnitude further reduction. This further reduction is due entirely to explicit coalescence. For this problem, the variable elimination algorithm itself does not perform any implicit coalescence. That is because it must eliminate the unobserved chance variable *Infection* first, before it eliminates any other variable. Although there are many conditional independence relations in the influence diagram, most involve the unobserved chance variable *Infection*. Once it is eliminated, *all* remaining variables become relevant in every subsequent elimination step and *all* are in the domain of every utility potential that is generated when a variable is eliminated. Because the optimal action for every decision variable is conditioned on *all* variables that are informational predecessors of the decision variable, that is, the entire prior history, a strategy tree is generated that has a branch for every possible history. As a result, explicit coalescence must be performed to convert this strategy tree to the equivalent strategy graph shown in Figure 7.

3.3.3 Elimination Order Within Groups of Observed Chance Variables

Before concluding our discussion of the traditional algorithm, we mention one more way in which it is possible to compress a strategy graph into a smaller equivalent graph.

In our discussion of elimination-ordering constraints in Sections 2.1, we noted that the variables in a set $\mathcal{Y}_{i+1}$ of chance variables that are observed after decision D_i, and before the next decision D_{i+1}, can be eliminated in any order relative to each other. For the *Arthronet* influence diagram, for example, the three observed chance variables before the *Implant* decision variable can be eliminated in any order. Similarly, the five observed chance variables between the *Implant* and *Scintigraphy* decision variables can be eliminated in any order relative to each other. Both the structure and the size of a strategy graph can be affected by the order in which these subsets of observed chance variables are eliminated.

By solving the *Arthronet* problem for all valid elimination orders, we found that the strategy graph in Figure 7 is the smallest optimal strategy graph for *any* elimination order, with just 60 nodes. But the size of the smallest strategy graph for a given elimination order, when averaged over all valid elimination orders, is a little more than 80 nodes, which shows that elimination order can affect the size of a strategy graph.

Recall that a node of a strategy graph can be generated for each instantiation of variables in the domain of a utility potential, and so there is a correlation between the size of a strategy graph and the size of utility potentials. It suggests that an elimination-ordering heuristic that minimizes the size of utility potentials will also tend to minimize strategy graph size.

4. Generalized Variable Elimination: Influence Diagrams and POMDPs

Recently, we proposed an improved variable elimination algorithm for influence diagrams, called *generalized variable elimination* (Hansen, 2021). The algorithm behaves exactly like the classic variable elimination algorithm reviewed in Section 3 when it eliminates variables in an order that is allowed by the classic algorithm. But it can also use techniques adapted from the value iteration algorithm for POMDPs to relax traditional constraints on elimination order, which leads to improved performance in many cases. In particular, the generalized algorithm is *not* constrained to eliminate all unobserved chance variables before it eliminates any other variable.

In this section, we review the generalized variable elimination algorithm and show how to modify it so that it represents a strategy as a graph. Then we show that it allows a more efficient approach to coalescence than is possible for the classic algorithm.

Notation. Recall that $sp(\mathbf{X})$ denotes the set of states of a joint variable $\mathbf{X}$. We let $bsp(\mathbf{X})$ denote the set of all *belief states* over the possible states of $\mathbf{X}$, where the term belief state (adopted from the POMDP literature) refers to a probability distribution over the possible states of a hidden variable. We let $b(\mathbf{X}) \in \mathbf{bsp}(\mathbf{X})$ denote a particular belief state, and $b(\mathbf{x})$ the probability (or belief) that the unobserved variable $\mathbf{X}$ is in state $\mathbf{x} \in sp(\mathbf{X})$.

Recall also that $\psi(\mathbf{X})$ denotes a utility potential defined as $\psi : sp(\mathbf{X}) \rightarrow \Re$. To model decision making under imperfect information, we let utility potentials have belief states over the possible states of unobserved variables as input. Thus we adopt the convention that $\psi(\mathbf{H}, B(\mathbf{U}))$ denotes a utility potential that is defined as $\psi : sp(\mathbf{H}) \times bsp(\mathbf{U}) \rightarrow \Re$, where the variables in the domain of the potential are partitioned into two subsets: a subset $\mathbf{H}$ (for "history") of decision and observed chance variables, and a subset $\mathbf{U}$ (for "unobserved") of unobserved chance variables. In this context, $B(\mathbf{U}))$ denotes a *belief variable* for the unobserved joint variable $\mathbf{U}$, where an instantiation of a belief variable $B(\mathbf{U})$ is a belief state $b(\mathbf{U})$, just as an instantiation of the observed variable $\mathbf{H}$ is a state $\mathbf{h}$.

4.1 Generalized Algorithm

Generalized variable elimination has relaxed constraints on elimination order that follow from more general definitions of utility potentials and operations on utility potentials.

4.1.1 Piecewise-linear and convex utility potentials

The key concept of the generalized algorithm is the concept of a *piecewise-linear and convex* (PWLC) utility potential. It is modeled on the concept of a PWLC value function for a POMDP, although it is a more general concept that includes a POMDP value function as a special case. A PWLC utility potential $\psi(\mathbf{H}, B(\mathbf{U}))$ is a utility potential that is represented by an indexed family of sets of ordinary potentials $\{\Gamma_{\mathbf{h}}\}_{\mathbf{h} \in sp(\mathbf{H})}$, where the ordinary potentials in each set $\Gamma_{\mathbf{h}}$ have domain $\mathbf{U}$. For a given history $\mathbf{h} \in sp(\mathbf{H})$ and belief state $b(\mathbf{U}) \in bsp(\mathbf{U})$, the value of the utility potential is defined as:

$$\psi(\mathbf{h}, b(\mathbf{U})) = \max_{\gamma \in \Gamma_{\mathbf{h}}} \sum_{\mathbf{u} \in sp(\mathbf{U})} b(\mathbf{u})\gamma(\mathbf{u}). \tag{10}$$

Interestingly, a similar generalization of the concept of a utility potential is used by some algorithms for solving limited-memory and multi-objective influence diagrams (Mauá, de Campos, & Zaffalon, 2012; Mauá & Cozman, 2016; Marinescu, Razak, & Wilson, 2012, 2017). But it is not used in the same way that we use it to relax constraints on elimination order in solving an influence diagram, and our development of this concept is more closely related to, and was inspired by, the dynamic programming approach to solving POMDPs.

Although the potentials in each set $\Gamma_{\mathbf{h}}$ that represents a PWLC utility potential are ordinary potentials, and the operations performed on them are the same operations for ordinary potentials, we refer to these potentials from now on as *linear potentials*. We do so both to indicate their role in representing a PWLC utility potential, and to distinguish them from ordinary (probability and utility) potentials used in the rest of the algorithm.

Pruning sets of linear potentials. The complexity of operations on PWLC utility potentials increases with the size of the sets of linear potentials that represent them. It follows that it is useful to remove redundant linear potentials from these sets in order to keep them as small as possible.

Given a set $\Gamma_{\mathbf{h}}$ of linear potentials over the unobserved variables $\mathbf{U}$ that represents a PWLC utility potential $\psi^{R(\mathbf{H}=\mathbf{h})}(B(\mathbf{U}))$, a linear potential $\gamma' \in \Gamma_{\mathbf{h}}$ is said to be *dominated* by the other linear potentials in the set, $\Gamma_{\mathbf{h}} \backslash \{\gamma'\}$, if for all belief states $b \in bsp(\mathbf{U})$:

$$\sum_{\mathbf{u} \in sp(\mathbf{U})} b(\mathbf{u}) \gamma'(\mathbf{u}) \leq \max_{\gamma \in \Gamma_{\mathbf{h}} \backslash \{\gamma'\}} \sum_{\mathbf{u} \in sp(\mathbf{U})} b(\mathbf{u}) \gamma(\mathbf{u}). \tag{11}$$

That is, a linear potential $\gamma' \in \Gamma_{\mathbf{h}}$ is dominated if there is no belief state $b \in bsp(\mathbf{U})$ for which it provides a better value than any other linear potential in $\Gamma_{\mathbf{h}} \backslash \{\gamma'\}$. In this case, the linear potential γ' is redundant and can be removed from the set $\Gamma_{\mathbf{h}}$ without affecting the value of the PWLC utility potential it represents. This condition can be tested by solving the following linear program, where the objective is to maximize the value of the variable ϵ.

$$\begin{aligned} \text{Variables: } & \epsilon,\ \{b(\mathbf{u})\}_{\mathbf{u} \in sp(\mathbf{U})} \\ \text{Constraints: } & \sum_{\mathbf{u} \in sp(\mathbf{U})} \left[b(\mathbf{u}) \cdot \left(\gamma'(\mathbf{u}) - \gamma(\mathbf{u}) \right) \right] \geq \epsilon,\ \forall \gamma \in \Gamma_{\mathbf{h}} \backslash \{\gamma'\} \\ & \sum_{\mathbf{u} \in sp(\mathbf{U})} b(\mathbf{u}) = 1 \text{ and } b(\mathbf{u}) \geq 0, \forall \mathbf{u} \in sp(\mathbf{U}) \end{aligned} \tag{12}$$

If the scalar value ϵ maximized by this linear program is non-positive, then γ' is dominated, and it can be safely removed from the set $\Gamma_{\mathbf{h}}$, that is, it can be "pruned."

From now on, we let $Prune(\Gamma)$ denote an operator that takes a set Γ of linear potentials with domain $\mathbf{U}$ and prunes its dominated linear potentials. A naive way to implement this operator is to test each linear potential by solving the above linear program. More efficient algorithms are described in the POMDP literature, to which we refer for details (Cassandra, Littman, & Zhang, 1997; Walraven & Spaan, 2017; Hansen & Bowman, 2020).

4.1.2 Operations on Piecewise-Linear and Convex Utility Potentials

We next describe how to generalize the operations on utility potentials performed by the variable elimination algorithm so that they apply to PWLC utility potentials.

Combination operations. We first consider the combination operations of addition and multiplication. (The division operation is not performed on utility potentials.) Recall that when two potentials are combined, the domain of the new potential is the union of the domains of the combined potentials. For PWLC utility potentials, we must also distinguish between observed and unobserved variables in the domain of a potential. Thus when an ordinary potential $\psi(\mathbf{H}, \mathbf{U})$ and a PWLC utility potential $\psi'(\mathbf{H}', B(\mathbf{U}'))$ are combined, the resulting PWLC utility potential is $\psi''(\mathbf{H}'', B(\mathbf{U}''))$, where $\mathbf{H}'' = \mathbf{H} \cup \mathbf{H}'$ is the set of observed variables and $\mathbf{U}'' = \mathbf{U} \cup \mathbf{U}'$ is the set of unobserved variables.

In the following definitions, we adopt the notational convention that a PWLC utility potential $\psi'(\mathbf{H}', B(\mathbf{U}'))$ is represented by an indexed family of sets of linear potentials over $\mathbf{U}'$, which is denoted $\{\Gamma'_{\mathbf{h}'}\}_{\mathbf{h}' \in sp(\mathbf{H}')}$.

- The *sum* of an ordinary utility potential $\psi(\mathbf{H}, \mathbf{U})$ and a PWLC utility potential $\psi'(\mathbf{H}', B(\mathbf{U}'))$ is a PWLC utility potential $\psi''(\mathbf{H}'', B(\mathbf{U}''))$, represented by an indexed family of sets of linear potentials, $\{\Gamma''_{\mathbf{h}''}\}_{\mathbf{h}'' \in sp(\mathbf{H}'')}$, where the linear potentials have domain $\mathbf{U}''$, and for each instantiation $\mathbf{h}''$ of $\mathbf{H}''$:

$$\Gamma''_{\mathbf{h}''} = Prune\left(\left\{\psi^{R\left(\mathbf{H}=\mathbf{h}''^{\downarrow\mathbf{H}}\right)}(\mathbf{U}) + \gamma'(\mathbf{U}') | \gamma' \in \Gamma'_{\mathbf{h}''^{\downarrow\mathbf{H}'}}\right\}\right). \tag{13}$$

  Note that $\psi^{R\left(\mathbf{H}=\mathbf{h}''^{\downarrow\mathbf{H}}\right)}(\mathbf{U}) = \psi(\mathbf{h}''^{\downarrow\mathbf{H}}, \mathbf{U})$ based on the definition of the restriction operator given by Equation (9). The sum of two PWLC utility potentials is considered in Appendix A.

- The *product* of a probability potential $\phi(\mathbf{H}, \mathbf{U})$ and PWLC utility potential $\psi'(\mathbf{H}', B(\mathbf{U}'))$ is defined similarly, except addition in Equation (13) is replaced by multiplication.

Marginalization operations. We next consider generalizations of the max and sum-marginalization operations. (The restriction operation only applies to ordinary potentials.)

- The elimination by *max-marginalization* of a decision variable D from a PWLC utility potential $\psi(\mathbf{H}, D, B(\mathbf{U}))$ creates a PWLC potential $\psi'(\mathbf{H}, B(\mathbf{U}))$, where for each instantiation $\mathbf{h}$ of $\mathbf{H}$:

$$\Gamma'_{\mathbf{h}} = Prune\left(\cup_{d \in sp(D)} \Gamma_{(\mathbf{h},d)}\right). \tag{14}$$

  That is, $\Gamma'_{\mathbf{h}}$ is the union of the sets $\{\Gamma_{(\mathbf{h},d)}\}_{d \in sp(D)}$, with subsequent pruning.

- Let $sp(C) = \{c_1, c_2, \ldots, c_{|sp(C)|}\}$ denote the set of states (or "observations") of an observed chance variable C. The elimination by *sum-marginalization* of an observed chance variable C from a PWLC utility potential $\psi(\mathbf{H}, C, B(\mathbf{U}))$ creates a PWLC utility potential $\psi'(\mathbf{H}, B(\mathbf{U}))$, where for each instantiation $\mathbf{h}$ of $\mathbf{H}$:

$$\Gamma'_{\mathbf{h}} = Prune\left(\left(\left(\Gamma_{(\mathbf{h},c_1)} \oplus \Gamma_{(\mathbf{h},c_2)}\right) \oplus \Gamma_{(\mathbf{h},c_3)}\right) \ldots \oplus \Gamma_{(\mathbf{h},c_{|sp(C)|})}\right). \tag{15}$$

  In this equation, the symbol $\oplus$ denotes the *cross sum* operator, defined as $A \oplus B = \{a + b | a \in A, b \in B\}$, where A and B are sets of linear potentials. The efficiency of this computation is dramatically improved by interleaving the cross-sum and pruning operations, as in the incremental pruning algorithm (Cassandra et al., 1997), so that:

$$\Gamma'_{\mathbf{h}} = Prune\left(Prune\left(Prune\left(\Gamma_{(\mathbf{h},c_1)} \oplus \Gamma_{(\mathbf{h},c_2)}\right) \oplus \Gamma_{(\mathbf{h},c_3)}\right) \ldots \oplus \Gamma_{(\mathbf{h},c_{|sp(C)|})}\right). \tag{16}$$

- The elimination by *sum-marginalization* of an unobserved chance variable C from a PWLC utility potential $\psi(\mathbf{H}, B(C, \mathbf{U}))$ creates a PWLC utility potential $\psi'(\mathbf{H}, B(\mathbf{U}))$, where for each instantiation $\mathbf{h}$ of $\mathbf{H}$:

$$\Gamma'_{\mathbf{h}} = Prune\left(\left\{\sum_C \gamma(C, \mathbf{U}) | \gamma \in \Gamma_{\mathbf{h}}\right\}\right). \tag{17}$$

  That is, the unobserved chance variable C in the domain of each linear potential $\gamma(C, \mathbf{U})$ in a set $\Gamma_{\mathbf{h}}$ is eliminated by simple sum-marginalization.

Conversion between representations of utility potentials. The generalized variable elimination algorithm represents utility potentials as ordinary potentials whenever possible, and as PWLC potentials in order to relax traditional constraints on elimination order.

At the start of the algorithm, all utility potentials are ordinary potentials. But to allow elimination of a decision variable by max-marginalization from an ordinary utility potential that includes unobserved chance variables in its domain, the ordinary utility potential must first be converted to a PWLC utility potential.

This conversion is easily done. For any ordinary utility potential $\psi(\mathbf{H}, \mathbf{U})$, with observed variables $\mathbf{H}$ and unobserved variables $\mathbf{U}$ in its domain, we can construct an equivalent PWLC potential $\psi'(\mathbf{H}, B(\mathbf{U}))$, which is represented by an indexed family of sets of linear potentials $\{\Gamma'_{\mathbf{h}}\}_{\mathbf{h} \in sp(\mathbf{H})}$, where each set $\Gamma'_{\mathbf{h}}$ is a singleton set defined as follows:

$$\Gamma'_{\mathbf{h}} = \{\psi^{R(\mathbf{H}=\mathbf{h})}(\mathbf{U})\}. \tag{18}$$

For any history $\mathbf{h}$ and belief state $b(\mathbf{U})$, we have $\sum_{\mathbf{u} \in sp(\mathbf{U})} b(\mathbf{u})\psi(\mathbf{h}, \mathbf{u}) = \psi'(\mathbf{h}, b(\mathbf{U}))$, and thus the two representations are equivalent. Simply stated, any ordinary (that is, linear) potential can also be represented by a PWLC potential.

When a PWLC utility potential $\psi(\mathbf{H})$ does not have any unobserved chance variables in its domain, it is also easily converted to an equivalent ordinary utility potential $\psi'(\mathbf{H})$, as follows: $\psi'(\mathbf{h}) = \max_{\gamma \in \Gamma_{\mathbf{h}}} \gamma$. In this case, each set $\Gamma_{\mathbf{h}}$ in the representation of $\psi(\mathbf{H})$ contains a single linear potential with an empty domain, and thus a scalar value, which is assigned to $\psi'(\mathbf{h})$. Thus any PWLC potential that does not have any unobserved variables in its domain can be represented by an equivalent ordinary (linear) potential.

4.1.3 Generalization of Pseudocode

The generalized variable elimination algorithm follows the pseudocode given by Algorithm 1, with just a couple of differences that reflect its use of PWLC utility potentials: (i) it has more relaxed constraints on elimination order, (ii) the condition for identifying relevant PWLC potentials is slightly more complex than the condition for identifying relevant ordinary potentials, and (iii) the representation of a policy is generalized so that it can be a function of belief states as well as history. We consider each of these generalizations in turn.

Relaxed constraints on elimination order. Recall from the discussion in Section 2.1 that the classic variable elimination algorithm eliminates variables in reverse order of the partial order of information precedence given by Equation (1). Because it assumes that the set $\mathbf{X}$ of unobserved chance variables comes last in the partial order of information precedence, it requires all unobserved chance variables to be eliminated first.

The generalized algorithm eliminates variables in an order that is also constrained by the partial order of information precedence, but with a key difference. It does not need to eliminate all unobserved variables before it eliminates any other variable. That is, the new algorithm eliminates variables in an order that respects the following revised partial order of information precedence,

$$\mathbf{Y}_1 \prec D_1 \prec \mathbf{Y}_2 \prec \ldots \prec D_{n-1} \prec \mathbf{Y}_n \prec D_n, \tag{19}$$

where this partial order no longer includes the set $\mathbf{X}$ of unobserved variables. Because the variables in $\mathbf{X}$ are not observed, they provide no information. It follows that they are not

actually subject to constraints based on information precedence. By representing utility potentials as PWLC functions, we are able to drop the constraint that all unobserved variables must be eliminated before any other variable is eliminated. Relaxing this constraint on elimination order is the central idea of the generalized approach to variable elimination.

Although the generalized algorithm is not required to eliminate all unobserved chance variables first, there are still constraints on how long the elimination of unobserved chance variables can be postponed. These constraints turn on the following definition, which is analogous to the definition of the set of informational predecessors of a decision variable given by Equation (2).

Definition 2. *The set of causal successors of a decision variable D, denoted $Succ(D)$, is the set of all chance variables that are descendants of the decision variable D via some directed path of conditional arcs in the influence diagram.*

Only conditional arcs are used to define causal precedence, just as only informational arcs are used to define informational precedence. Elimination-ordering constraints based on causal precedence take this form: *any variable that is a causal successor of a decision variable must be eliminated before the decision variable itself is eliminated.* This constraint ensures that a decision variable is eliminated from all probability potentials before it is eliminated from any utility potential.

We combine the elimination-ordering constraints based on causal precedence with the elimination-ordering constraints based on information precedence by requiring that variables be eliminated in an order that is *consistent* with an influence diagram, defined as follows.

Definition 3. *An elimination order is* consistent *with an influence diagram if each decision variable is eliminated (i) after all of its causal successors are eliminated and (ii) before any of its informational predecessors are eliminated.*

Identification of relevant potentials. When a variable is eliminated, all potentials that have a value that depends on the eliminated variable must be replaced by equivalent potentials that do not. We call the potentials that need to be replaced when a variable is eliminated the *relevant potentials*. For the classic algorithm, they are the potentials that include the variable selected for elimination in their domain.

For the generalized algorithm, it is still the case that any potential that includes the eliminated variable in its domain is relevant. But if the eliminated variable is an observed chance variable, a PWLC utility potential that does *not* include the eliminated variable in its domain is nevertheless relevant if an unobserved variable in its domain is *d-connected* to the eliminated variable. In that case, the observed state of the eliminated variable can influence a *belief* about the state of the unobserved variable, and in that way it can influence the value of the PWLC potential (Hansen, 2021).

There is a simple way to avoid this extra test for relevance, however. We simply assume that if the utility potential ψ_i generated at the end of an elimination step is PWLC, it is relevant in the next elimination step. Although this assumption does not necessarily hold, it almost always holds, and when it doesn't, the algorithm is still correct. For even if it treats a PWLC potential as relevant when it is not, the algorithm is correct as long as it processes every relevant potential. We use this implementation of the algorithm for our results in this paper. When implemented in this way, there cannot be more than one PWLC utility potential at a time, and so there is no need to add two PWLC utility potentials.

Generalized representation of strategy. Finally, recall that a strategy can be represented by a sequence of policies, $\Delta = (\delta_{D_1}, \ldots, \delta_{D_n})$, with one policy for each decision variable $D_i \in \mathbf{D}$. A policy δ_{D_i} has the same domain as the utility potential created when the decision variable D_i is eliminated. Thus when variables are eliminated in a traditional order, a policy is represented in the traditional way as a mapping $\delta_{D_i} : sp(Pred(D_i)) \rightarrow sp(D_i)$, as described in Section 2.2. But when variables are eliminated in a non-traditional order, the utility potential created when a decision variable D_i is eliminated is PWLC. Therefore the corresponding policy is a mapping,

$$\delta_{D_i} : sp(\mathbf{H}_i) \times bsp(\mathbf{U}_i) \rightarrow sp(D_i), \tag{20}$$

which is defined so that

$$\delta_{D_i}(\mathbf{h}_i, b(\mathbf{U}_i)) = d\left(\arg\max_{\gamma \in \Gamma_{\mathbf{h}_i}} \sum_{\mathbf{u}_i} b(\mathbf{u}_i)\gamma(\mathbf{u}_i)\right), \tag{21}$$

where $d(\gamma) \in sp(D_i)$ denotes the action associated with linear potential γ. The action is associated with the linear potential when it is created, and it represents the action used to generate the linear potential.

As already pointed out, this representation of a strategy has two disadvantages. First, it can be difficult for a human to understand. Second, when a policy is a function of belief states, strategy execution requires updating a belief state by Bayes' rule. Both disadvantages can be overcome by representing a strategy as a graph, which we consider next.

4.2 Strategy Graph Construction

Because the generalized algorithm behaves exactly like traditional variable elimination when it eliminates variables in an order allowed by the traditional algorithm, the approach to strategy graph construction and simplification for traditional variable elimination described in Sections 3.2 and 3.3 is also used by the generalized algorithm. It remains only to show how to construct a strategy graph when variables are eliminated in an order that is not allowed by the traditional algorithm, and operations are performed on PWLC utility potentials.

Recall that a PWLC utility potential, $\psi : sp(\mathbf{H}) \times bsp(\mathbf{U}) \rightarrow \Re$, is represented by an indexed family of sets of linear potentials, $\{\Gamma_{\mathbf{h}}\}_{\mathbf{h} \in sp(\mathbf{H})}$, where the linear potentials have the domain $\mathbf{U}$. The corresponding *strategy node function* maps each instantiation of the variables in the domain of the utility potential to a node of the strategy graph, as follows,

$$s_\psi : sp(\mathbf{H}) \times bsp(\mathbf{U}) \rightarrow \mathcal{N} \cup \{nil\}, \tag{22}$$

where $\mathcal{N}$ denotes the set of nodes of the strategy graph. We can represent the strategy node function by the same indexed family of sets of linear utility potentials that represents the PWLC utility potential, so that

$$s_\psi(\mathbf{h}, b(\mathbf{U})) = n\left(\arg\max_{\gamma \in \Gamma_{\mathbf{h}}} \sum_{\mathbf{U}} b(\mathbf{u}) \cdot \gamma(\mathbf{u})\right), \tag{23}$$

where $n(\gamma)$ denotes the node of the strategy graph associated with the linear potential γ, or else *nil*, if the linear potential is not associated with any node of the strategy graph.

Below, we describe how generalized variable elimination constructs a strategy graph with a strategy node function that is represented in this way.

4.2.1 Elimination of Decision Variable by Max-Marginalization

Recall that in order to eliminate a decision variable D from an ordinary utility potential $\psi(\mathbf{H}, D, \mathbf{U})$ with unobserved variables $\mathbf{U}$ in its domain, the ordinary utility potential must first be converted to an equivalent PWLC potential, $\psi(\mathbf{H}, D, B(\mathbf{U}))$, as described at the end of Section 4.1.2. In this case, each linear potential γ in the representation of $\psi(\mathbf{H}, D, B(\mathbf{U}))$ is associated with *nil*, that is, $n(\gamma) = nil$, since strategy graph construction does not begin until the decision variable D is actually eliminated from the utility potential.

Elimination of a decision variable D from a PWLC utility potential $\psi(\mathbf{H}, D, B(\mathbf{U}))$ creates a PWLC utility potential $\psi'(\mathbf{H}, B(\mathbf{U}))$, represented by an indexed family of sets of linear potentials, $\{\Gamma'_{\mathbf{h}}\}_{\mathbf{h}\in sp(\mathbf{H})}$, where each set $\Gamma'_{\mathbf{h}}$ is defined by Equation (14). Before pruning, each set $\Gamma'_{\mathbf{h}}$ is the union of the sets $\{\Gamma_{\mathbf{h},d}\}_{d\in sp(D)}$, and so each linear potential $\gamma' \in \Gamma'_{\mathbf{h}}$ corresponds to a linear potential γ from a set $\Gamma_{\mathbf{h},d}$, for some d. Therefore, for each linear potential γ' in Γ'_h, we set $n(\gamma')$ equal to a newly-created decision node that has an outgoing arc labeled by the action d, where the outgoing arc has successor node $n(\gamma)$.

4.2.2 Elimination of Observed Chance Variable by Sum-Marginalization

Elimination of an observed chance variable C from a PWLC utility potential $\psi(\mathbf{H}, C, B(\mathbf{U}))$ creates another PWLC utility potential $\psi'(\mathbf{H}, B(\mathbf{U})$, which is represented by an indexed family of sets of linear potentials $\{\Gamma'_{\mathbf{h}}\}_{\mathbf{h}\in sp(\mathbf{H})}$, where each set $\Gamma'_{\mathbf{h}}$ is the cross sum of the sets $\{\Gamma_{\mathbf{h},c}\}_{c\in sp(C)}$, as defined by Equation (16). For each linear potential $\gamma' \in \Gamma'_{\mathbf{h}}$, we set $n(\gamma')$ equal to a newly-created observation node that has one outgoing arc for each observation $c \in sp(C)$. For the outgoing arc labeled by observation c, the successor node is $n(\gamma_c)$, where $\gamma_c \in \Gamma_{(\mathbf{h},c)}$ is the linear potential for observation c used to compute $\gamma' = \sum_{c\in sp(C)} \gamma_c$.

4.2.3 Graph Simplification When Observed Chance Variable is Eliminated

Whenever an observed chance variable C is eliminated from a PWLC utility potential $\psi(\mathbf{H}, C, B(\mathbf{U}))$, the observation nodes added to the strategy graph can be simplified in the same ways described in Section 3.2.4, with the following minor changes.

Removal of zero-probability observation branches. Recall that an observation node with an outgoing arc for each observation $c \in sp(C)$ can be simplified by removing arcs that correspond to zero-probability observations. For observation nodes created by eliminating an observed chance variable C from a PWLC utility potential $\psi(\mathbf{H}, C, B(\mathbf{U}))$, we generalize the check for zero-probability observations as follows. If $P(c|\mathbf{h}, \mathbf{u}) = 0$ for *every* instantiation $\mathbf{u}$ of $\mathbf{U}$, then the outgoing arc for observation c has probability zero and can be removed from the strategy graph. In words, an observation c is impossible after history $\mathbf{h}$ if it is impossible after history $\mathbf{h}$ no matter what the unobserved state $\mathbf{u}$.

This generalized test can remove many zero-probability observations. But it is not guaranteed to remove all, by contrast to the related test for the traditional variable elimination algorithm. However, any zero-probability observations that are missed in this step can be identified and removed in a post-processing step, as described at the end of Section 4.3.1.

Removal of redundant observation nodes. If the observation node $n(\gamma)$ associated with a linear potential $\gamma \in \Gamma'_{\mathbf{h}}$ is redundant, which means all of its outgoing arcs lead to the same successor node, it is removed and $n(\gamma)$ is set equal to the successor node.

4.2.4 Elimination of Unobserved Chance Variable by Sum-Marginalization

Recall that elimination of an unobserved chance variable C from a PWLC utility potential $\psi(\mathbf{H}, B(C, \mathbf{U}))$ creates another PWLC utility potential $\psi'(\mathbf{H}, B(\mathbf{U}))$, represented by an indexed family of sets of linear potentials $\{\Gamma'_{\mathbf{h}}\}_{\mathbf{h}\in sp(\mathbf{H})}$, where each set $\Gamma'_{\mathbf{h}}$ is defined by Equation (17). Each linear potential $\gamma' \in \Gamma'_{\mathbf{h}}$ is created from a linear potential $\gamma \in \Gamma_{\mathbf{h}}$, as follows: $\gamma'(\mathbf{U}) = \sum_C \gamma(C, \mathbf{U})$. To construct the corresponding strategy node function $s_{\psi'}(\mathbf{H}, B(\mathbf{U}))$, we simply let $n(\gamma') = n(\gamma)$. In words, the new linear potential $\gamma'(\mathbf{U})$ points to the same node of the strategy graph as the linear potential $\gamma(C, \mathbf{U})$ from which it is created by eliminating the unobserved chance variable C by sum-marginalization.

If the set $\mathbf{U}$ of unobserved variables is empty, the PWLC utility potential $\psi'(\mathbf{H}, B(\mathbf{U}))$ created by this operation can be converted to an ordinary utility potential, as described at the end of Section 4.1.2. In that case, of course, it is straightforward to convert the PWLC strategy node function to an equivalent ordinary strategy node function.

4.2.5 Addition and Multiplication

When an ordinary potential $\psi(\mathbf{H}, \mathbf{U})$ is added to, or multiplied by, a PWLC potential $\psi'(\mathbf{H}', B(\mathbf{U}'))$, represented by an indexed family of sets of linear potentials over $\mathbf{U}'$, denoted $\{\Gamma'_{\mathbf{h}}\}_{\mathbf{h}'\in\mathbf{H}'}$, the result is a PWLC potential $\psi''(\mathbf{H}'', B(\mathbf{U}''))$, represented by a new indexed family of sets of linear potentials over $\mathbf{U}''$, denoted $\{\Gamma''_{\mathbf{h}''}\}_{\mathbf{h}''\in\mathbf{H}''}$, as defined by Equation (13). Since each linear potential $\gamma'' \in \Gamma''_{\mathbf{h}''}$ is derived from a corresponding linear potential $\gamma' \in \Gamma'_{\mathbf{h}''\downarrow\mathbf{H}'}$, the strategy node for γ'' is inherited from γ', that is, $n(\gamma'') = n(\gamma')$. Thus no new strategy nodes are created by this operation, although strategy nodes may be pruned if their corresponding linear potentials are pruned.

4.2.6 Example: Mildew Treatment

To illustrate this procedure for strategy graph construction, we consider how a strategy graph is constructed when the generalized algorithm solves the mildew problem represented by the influence diagram in Figure 1. When solving this problem, it can eliminate variables in an order that is not allowed by the traditional variable elimination algorithm, and so we show how a non-traditional elimination order affects strategy graph construction.

Eliminate unobserved chance variables H and M^*. Because the unobserved chance variables H and M^* are causal successors of the decision variable A, they must be eliminated first, generating the utility potential $\psi_2(A, Q, M) = \sum_{H,M^*} P(H|M^*, Q)P(M^*|A, M)U(H)$.

Eliminate decision variable A. Unlike traditional variable elimination, the generalized algorithm can eliminate the decision variable A before the unobserved chance variables Q and M are eliminated. Doing so generates a PWLC utility potential $\psi_3(B(\mathrm{Q}, \mathrm{M})) = \max_A(C(A) + \psi_2(A, Q, M))$, represented by the following set of four linear potentials:

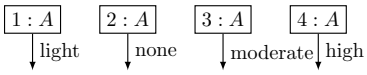
$$\begin{aligned}\Gamma_3=\{&\langle(7.75, 4.85, 1.45, -0.8, 9.9, 9.9, 4.85, 1.45, 11.75, 9.85, 7.75, 4.85, 12.5, 11.6, 9.85, 7.75), 2\rangle,\\ &\langle(4.75, 4.75, 4.17, 1.17, 6.9, 6.9, 6.9, 5.89, 8.75, 8.75, 8.37, 6.43, 9.5, 9.5, 9.32, 8.25), 3\rangle,\\ &\langle(5.75, 5.17, 2.17, -1.0, 7.9, 7.9, 6.89, 2.17, 9.75, 9.37, 7.43, 5.17, 10.5, 10.32, 9.25, 7.43), 1\rangle,\\ &\langle(3.75, 3.75, 3.75, 3.17, 5.9, 5.9, 5.9, 5.9, 7.75, 7.75, 7.75, 7.37, 8.5, 8.5, 8.5, 8.32), 4\rangle\}.\end{aligned}$$

Traditional VE			
	Eliminated variable	Relevant vars. Obs.	Unobs.
1	H		Q, M*
2	Q	OQ	M*
3	M*	OQ, A	M
4	M	OQ, A, OM	
5	A	OQ, OM	
6	OQ	OM	
7	OM		

(a)

Generalized VE			
	Eliminated variable	Relevant vars. Obs.	Unobs.
1	H		Q, M*
2	M*	A	Q, M
3	A		Q, M
4	OQ		Q, M
5	Q		M
6	OM		M
7	M		

(b)



(c)

Figure 8: The two tables show the order in which *Mildew* variables are eliminated by (a) traditional and (b) generalized variable elimination, with the relevant observed and unobserved variables for each elimination step. On the right, (c) shows the terminal nodes of the strategy graph after the generalized algorithm eliminates the decision variable A.

Each of these linear potentials has dimension 16, with instantiations of the unobserved variables Q and M listed in the order: ([f, no], [f, l], [f, m], [f, s], [a, no], [a, l], [a, m], [a, s], [g, no], [g, l], [g, m], [g, s], [v, no], [v, l], [v, m], [v, s]). Note that each linear potential is associated with the index of a node of the strategy graph shown in Figure 8c, where there is one decision node for each of the four possible actions of the decision variable A.

Eliminate observed chance variable OQ and unobserved chance variable Q. The observed chance variable OQ is eliminated next, generating the PWLC utility potential $\psi_4(B(\text{Q, M})) = \sum_{OQ} P(OQ|Q)\psi_3(B(\text{Q, M}))$, which is represented by a set Γ_4 of 190 linear potentials of dimension 16. This set and corresponding strategy graph are too large to show here. When the unobserved chance variable Q is then eliminated, the PWLC utility potential $\psi_5(B(\text{M})) = \sum_{Q} P(Q)\psi_4(B(\text{Q, M}))$ is created, which is represented by the set:

$$\begin{aligned}\Gamma_5 = \{&\langle(10.285, 9.045, 5.54, 2.65), 2\rangle, \langle(6.285, 6.285, 6.285, 6.037), 4\rangle,\\ &\langle(7.525, 7.423, 7.0449, 5.2716), 11\rangle, \langle(8.025, 7.516, 6.6395, 5.1193), 15\rangle,\\ &\langle(6.705, 6.705, 6.6426, 5.8858), 16\rangle, \langle(8.675, 8.124, 6.7258, 3.7282), 7\rangle,\\ &\langle(9.355, 8.636, 6.2398, 3.5454), 10\rangle, \langle(6.975, 6.975, 6.8312, 5.6934), 18\rangle,\\ &\langle(7.285, 7.285, 7.037, 5.344), 3\rangle, \langle(8.335, 7.826, 6.8453, 4.7699), 8\rangle,\\ &\langle(7.865, 7.721, 6.9254, 4.2299), 6\rangle\langle(7.215, 7.113, 6.8391, 5.621), 12\rangle\\ &\langle(6.625, 6.625, 6.583, 5.9175), 17\rangle\langle(6.605, 6.503, 6.3525, 5.9329), 13\rangle\\ &\langle(9.265, 8.235, 6.1455, 3.8745), 9\rangle\langle(9.665, 8.8418, 5.8904, 2.714), 5\rangle\\ &\langle(6.945, 6.843, 6.6505, 5.8134), 14\rangle\}.\end{aligned}$$

All 17 of the linear potentials in Γ_5 have dimension 4, with the states of the unobserved variable M in the order: (no, l, m, s). Each linear potential is also associated with the index of a node of the strategy graph shown in Figure 9. However, only 14 of the 17 linear potentials have a corresponding observation node in the strategy graph. The other three have a corresponding decision node because three of the originally-generated observation

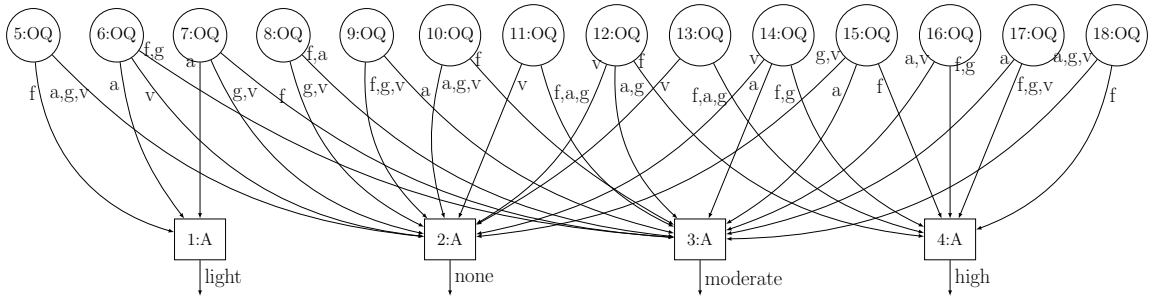


Figure 9: Strategy graph for *Mildew* problem after elimination of unobserved variable Q.

nodes are redundant, which means all of their outgoing arcs lead to the same successor node. After removal of these redundant observation nodes, the corresponding linear potentials are associated with the index of the successor decision node of the removed redundant node.

Eliminate observed chance variable OM and unobserved chance variable M. Elimination of the observed chance variable OM generates the PWLC utility potential $\psi_6(B(\mathrm{M})) = \sum_{OM} P(OM|M)\psi_5(B(\mathrm{M}))$, which is represented by a set Γ_6 of 328 linear potentials of dimension 4. Again, it is too many to show here. Elimination of the unobserved chance variable M then generates an ordinary utility potential $\psi_7(\lambda) = \sum_M P(M)\psi_6(B(\mathrm{M}))$, which has an empty domain and a scalar value of 8.50046 (rounded to five decimal places). The final strategy graph is shown in Figure 2b.

Discussion. When the traditional variable elimination algorithm solves this problem, as described in Section 3.3.2, it constructs the strategy tree shown in Figure 2a – unless it also uses *explicit* coalescence to merge duplicate subgraphs. By contrast, when the generalized algorithm solves this problem by eliminating variables in the order shown in Table 8b, it constructs the strategy graph in Figure 2b, with duplicate subgraphs merged *implicitly*.

A comparison of Tables 8a and 8b shows why the traditional algorithm needs to perform coalescence explicitly while the generalized algorithm can perform coalescence implicitly. When the traditional algorithm eliminates all unobserved chance variables first, the un-eliminated decision and observed chance variables are relevant in *all* subsequent elimination steps. By contrast, when the generalized algorithm postpones elimination of the unobserved chance variables Q and M until the decision variable and some of the observed chance variables are eliminated, the un-eliminated decision and observed chance variables are *not* relevant in subsequent elimination steps, which leads to implicit coalescence.

For this problem, traditional variable elimination finds a solution faster than generalized variable elimination (when the latter eliminates variables in the order shown in Table 8b). But that is primarily because a belief state dynamic programming recurrence provides little or no computational leverage when solving a problem with a single decision variable and such a short history.[6] For the multi-stage influence diagrams we consider next, eliminating variables in a non-traditional order leads to speedup, as well as implicit coalescence.

6. It is also because the generalized algorithm solves a more general problem. After the variable OM is eliminated, and before the variable M is eliminated, it has found an optimal solution for *all* possible belief states over the possible states of the unobserved variable M. By contrast, the traditional algorithm only finds a solution for a single initial belief state over the possible states of M.

4.3 Coalescence

Like traditional variable elimination, the generalized algorithm can perform coalescence both implicitly and explicitly. But unlike the traditional algorithm, it can perform implicit coalescence when solving influence diagrams that represent partially observable problems.

4.3.1 Implicit Coalescence

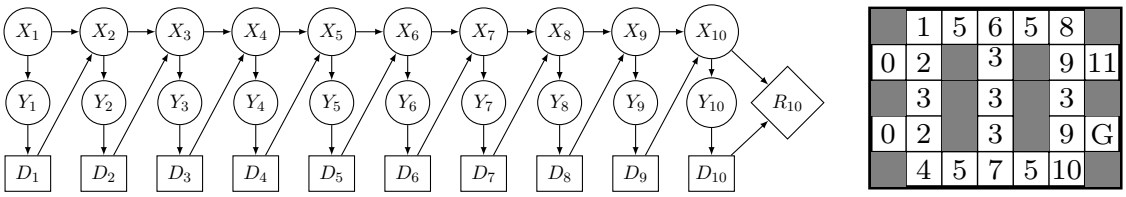
In Section 3.3.1, we related implicit coalescence to the dynamic programming recurrence solved by the traditional algorithm. A similar analysis applies to the generalized algorithm.

When variables are eliminated in an order that is *not* allowed by the traditional variable elimination algorithm, the generalized algorithm solves a dynamic programming recurrence that is defined not only for all instantiations of the observed variables in the domain of the utility potential created when the variable is eliminated, but for all belief states over the possible instantiations of the unobserved variables. In this more general dynamic programming recurrence, a subproblem is defined by a triple of the variable eliminated, an instantiation of the remaining *observed* variables in the domain of the utility potential, and a belief state over possible instantiations of the remaining *unobserved* variables. Although the space of belief states is continuous, it is partitioned into a finite number of regions because each linear potential in the representation of a PWLC utility potential gives the optimal utility for a region of belief states. It follows that when variables are eliminated in a non-traditional order, the dynamic programming recurrence solved by the generalized algorithm has one subproblem for each linear potential in the representation of a PWLC utility potential it constructs. In turn, the strategy graph has one node for each linear potential generated when a decision variable or observed chance variable is eliminated.

Because the generalized algorithm allows dynamic programming to be performed over belief states as well as observed states, it can leverage independence among variables more effectively. In particular, observed variables that represent prior history can be conditionally independent of the variable selected for elimination *given a belief state over the possible states of the relevant unobserved variables*, and thus irrelevant in an elimination step.

Consider the mildew problem. The traditional algorithm first eliminates all unobserved variables: H, Q, M^*, and M. When it eliminates decision variable A, the two observed variables OQ and OM are relevant, and a decision node is created for each instantiation of them, leading to construction of the strategy tree in Figure 2a. By contrast, the generalized algorithm can postpone elimination of the unobserved variables Q and M. In this case, when A is eliminated, the observed variables OQ and OM (representing prior history) are *not* relevant given a belief state over the possible states of the relevant unobserved variables Q and M, and so the strategy graph in Figure 2b is constructed, instead of a strategy tree.

We have the following parallel. In the case of an influence diagram for a completely observable MDP, observation of the current state makes the previous history irrelevant, as illustrated by the example considered in Section 3.3.1. In the case of the influence diagram for the mildew problem, which is a partially observed problem, a belief state over the possible states of the unobserved chance variables Q and M makes the previous history irrelevant. In both cases, the irrelevance of previous history reflects a Markovian property of the problem that is leveraged by dynamic programming in solving the problem. Leveraging Markovian independence from previous history leads to implicit coalescence, and often speedup.



(a) (b)

Figure 10: (a) Influence diagram for ten-stage (b) maze POMDP.

Example: Maze POMDP. For a vivid example of how the generalized algorithm can leverage a Markovian property over belief states to speed up problem solving, as well as construct a strategy graph with implicit coalescence, we consider a toy robot planning problem introduced in previous work on limited-memory influence diagrams (Nilsson & Hohle, 2001) and also used as an example in our earlier paper (Hansen, 2021).

Figure 10a shows an influence diagram that represents a ten-stage path-planning problem for the "maze" shown in Figure 10b. Each white cell in the maze represents a state of an unobserved chance variable X_t, for each stage $t = 1, \ldots, 10$, with an absorbing goal state marked by the letter G. In all, there are 23 possible states. Shaded cells and the outside borders of the maze represent walls through which the robot cannot pass. For each decision variable D_t, for $t = 1, \ldots, 10$, the robot can take one of four possible actions; it can move a single step in any of the four compass directions. It successfully moves in its intended direction with probability 0.89. It moves sideways with probability 0.02 (0.01 for each side), it moves backward with probability 0.001, and it fails to move with probability 0.089. If its movement would take it into a wall, or if it is in the goal state, it remains where it is. For each observed chance variable Y_t, for $t = 1, \ldots, 10$, the robot can accurately sense whether the neighboring cell in each direction of the compass is a wall. There are 13 possible observations, including perfect observation of the goal state, with the non-goal observations indexed from 0 through 11. The index of the observation received when the robot is in a given state of the maze is shown in that state in Figure 10b. Because the same observation can be received in different states, the problem is partially observable. (For example, observation 3 can be received in five different states.)

The problem begins with the robot placed randomly in a non-goal state, so that it does not know its initial location. The robot performs an action in each of a sequence of ten stages. If it reaches the absorbing 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.

Elimination order and algorithm efficiency. When the influence diagram for this problem is solved by traditional variable elimination, all unobserved chance variables must be eliminated before any other variable is eliminated. But after all ten unobserved chance variables are eliminated, the utility potential created when the decision variable D_{10} is eliminated includes in its domain all nine of the other decision variables and all ten observed chance variables. Thus it has $13^{10} \cdot 4^9$ possible instantiations, one for each possible history before the last decision! As a result, traditional variable elimination cannot solve this problem even after many hours of computation time.

By contrast, generalized variable elimination finds an optimal solution in just a couple of seconds. Unlike the traditional algorithm, it can eliminate the decision variable D_{10} before eliminating any other variable. It does so by optimizing for all belief states over the possible states of the unobserved variable X_{10}, which is the only relevant variable when D_{10} is eliminated first. After that, generalized variable elimination can eliminate variables in the same order in which variables are eliminated by the standard value iteration algorithm for POMDPs (Cassandra et al., 1997), which is D_{10}, X_{10}, Y_{10}, D_9, X_9, Y_9, D_8, etc. However, we found that the algorithm runs even faster, and constructs a smaller strategy graph, when it eliminates variables in the order: D_{10}, Y_{10}, X_{10}, D_9, Y_9, X_9, D_8, etc.

Elimination order and implicit coalescence. When this influence diagram is solved by traditional variable elimination, and explicit coalescence is not performed, a strategy tree of *immense* size is created, with $52^9 \cdot 13$ terminal decision nodes alone. (Many of these nodes can be removed from the final strategy tree because they are only reachable via suboptimal or zero-probability branches. But they are all initially generated!)

When generalized variable elimination solves the same problem by eliminating variables in the order D_{10}, Y_{10}, X_{10}, D_9, Y_9, X_9, D_8, etc., even if explicit coalescence is not performed, it generates an equivalent strategy graph with just 71 nodes and 167 arcs. The *much* smaller strategy graph constructed by generalized variable elimination compared to the strategy tree constructed by traditional variable elimination reflects *much* more effective use of dynamic programming in solving this problem. In particular, the problem is solved more efficiently by solving a dynamic programming recurrence over belief states instead of histories. Note that when variables are eliminated in this non-traditional order, each time a decision variable D_t or an observed chance variable Y_t is eliminated, the only relevant variable is the unobserved chance variable X_t. That is, *no observed variable that represents the history of the process is ever relevant in any elimination step.* By contrast, when variables are eliminated in a traditional order, *every* observed variable that represents the history of the process is relevant in every elimination step.

Additional simplification in a post-processing step. Some further simplification of the strategy graph is still possible. In Section 4.2.3, we described how generalized variable elimination can remove zero-probability observations from a strategy graph, which can lead to subsequent removal of unreachable nodes. However, it can only do so if there are relevant decision and observed chance variables that represent prior history and affect observation probabilities when an observed chance variable is eliminated. When the maze POMDP is solved by generalized variable elimination using our elimination order, however, there are no relevant *observed* variables in any elimination step! Thus for this example and elimination order, zero-probability observations cannot be removed in the same way.

Once a strategy graph has been constructed, however, it is possible to identify and remove zero-probability observations in a post-processing step. For each observation node in the final strategy graph, its ancestor nodes in the graph can be analyzed to determine whether a given observation has zero probability or not. By performing this analysis, we were able to further simplify the strategy graph constructed by generalized variable elimination, and the number of nodes in the graph was reduced from 71 to 58.

Figure 11 shows this 58-node strategy graph with one more simplification: each decision node has an incoming arc from an observation node in the previous step only if the arc

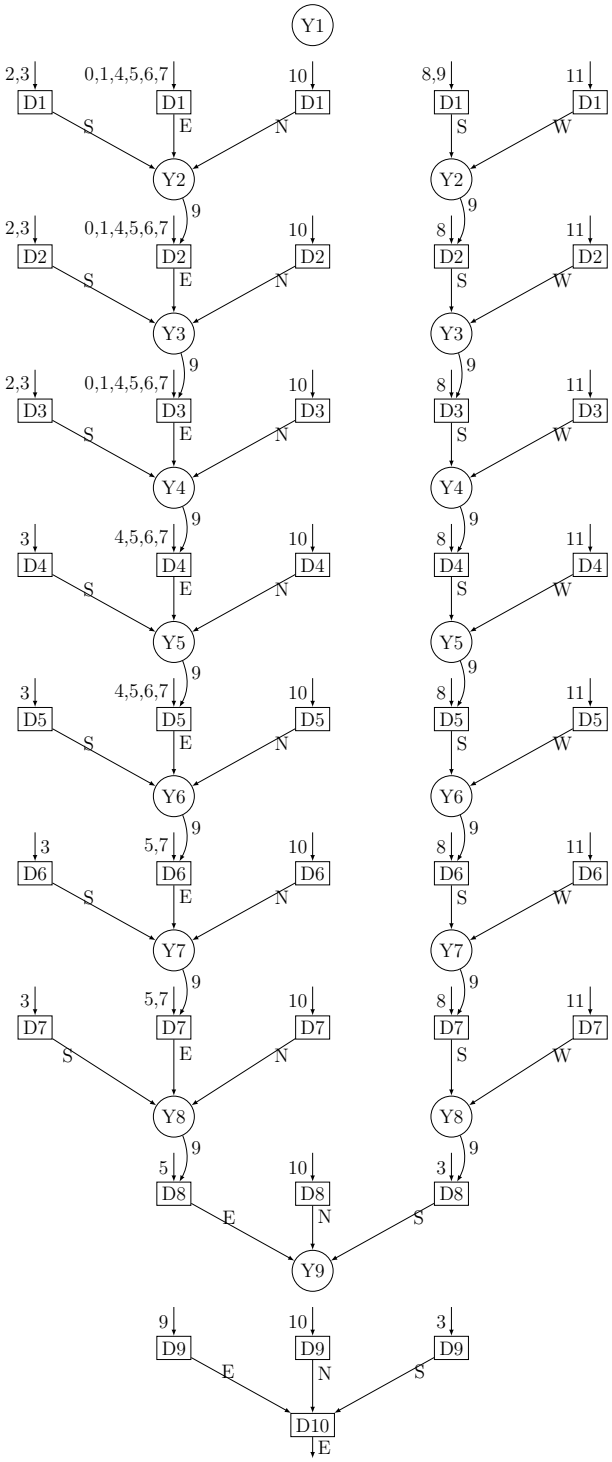


Figure 11: Strategy graph for the maze POMDP. The graph is simplified by not showing the predecessor node of an arc when the successor node depends only on the observation.

is labeled by observation 9. Otherwise, it has an incoming arc that is labeled by the observation(s) that lead to this node regardless of previous history. This simplification is possible because the most recent observation is enough to determine the next decision node of the strategy graph for this problem unless a wall is observed to the west only, which is observation 9 in Figure 10b. Observation 9 is possible in just two states of the maze: the state immediately west of the goal and the state two steps above it, which is immediately west of a dead-end state. *Surprisingly, an optimal strategy for the maze POMDP only needs memory to disambiguate the identical observation that is received in these two states.* By contrast to the traditional algorithm, generalized variable elimination is able to leverage the very simple structure of an optimal strategy for this problem to speed up problem solving.

The strategy graph in Figure 11 has been simplified in a couple additional ways. None of the arcs is labeled by observation of the goal state because once the absorbing goal state is reached, the strategy no longer matters. Similarly, none of the arcs is labeled by an observation that reveals the robot must be in a state from which the goal cannot be reached in the remaining number of steps, since again, the strategy does not matter in that case.

As this example shows, construction of a strategy graph that is simplified in all the ways we have described can help make the logic of a strategy easier to interpret and analyze.

4.3.2 Explicit Coalescence

We have shown that variable elimination implicitly merges duplicate subgraphs of a strategy graph when one or more un-eliminated decision or observed chance variables are not relevant in an elimination step, that is, when part of the prior history is not relevant. The more un-eliminated observed variables are irrelevant, the more coalescence is performed implicitly. In fact, when *all* un-eliminated observed variables are irrelevant in *every* elimination step, there are no duplicate subgraphs at all, as illustrated by the strategy graphs for the maze and mildew problems, as well as for the two-stage completely observable MDP.

For many or most influence diagrams, however, regardless of elimination order, there are elimination steps in which some un-eliminated decision or observed chance variables are relevant. When part or all of the history *is* relevant when a decision or observed chance variable is eliminated, the strategy graph constructed by variable elimination can have un-merged duplicate subgraphs, and so we must consider how to merge them explicitly.

Algorithm for explicit coalescence. Let $\psi_i(\mathbf{H}, B(\mathbf{U}))$ denote a PWLC utility potential that is created by eliminating a decision or observed chance variable, where $\mathbf{H}$ denotes the variables that represent the relevant history. For each instantiation $\mathbf{h}$ of $\mathbf{H}$, there is a set $\Gamma_{\mathbf{h}}$ of linear potentials over the unobserved variables $\mathbf{U}$, where each linear potential γ in $\Gamma_{\mathbf{h}}$ is associated with a node $n(\gamma) \in \mathcal{N}$ of the strategy graph.

We detect and merge duplicate nodes of the strategy graph as follows. After a set $\Gamma_{\mathbf{h}}$ of linear potentials is created and pruned, the node $n(\gamma)$ associated with each linear potential $\gamma \in \Gamma_{\mathbf{h}}$ is compared to the nodes in each set of linear potentials $\Gamma_{\mathbf{h}'}$ already created for other instantiations $\mathbf{h}'$ of $\mathbf{H}$. If node $n(\gamma)$ is a duplicate of an already-existing node, they are merged in the same way described above for nodes eliminated in a traditional order.

For efficiency, we perform this comparison-and-merge step *after* pruning a set $\Gamma_{\mathbf{h}}$ of linear potentials, so as not to waste effort on nodes that will be pruned. This delay requires no additional checking, since one node cannot be duplicate of another node in the same set.

Elim. Step	Eliminated Variable	Dim.	Relevant variables: Observed	Relevant variables: Unobserved	Irrelevant var. count
1	Remove prosthesis	2		Infection	12
2	Frozen sections PMN	3	Synovial	Infection	10
3	Make synovial biopsy	2		Infection	10
4	Sequential Ga67 Tc99	3	Scintigraphy	Infection	9
5	Make scintigraphy	2		Infection	9
6	C-reactive protein	2		Infection	8
7	Erythrocyte sedimentation rate	2		Infection	7
8	Knee motion	2		Infection	6
9	Knee deep infection	2	Allergy, Diabetes, BMI, Implant, Ischemia, Drained CC		0
10	Drained CC	3	Allergy, Diabetes, BMI, Implant, Ischemia		0
11	Ischemia	3	Allergy, Diabetes, BMI, Implant,		0
12	Implant prosthesis	2	Allergy, Diabetes, BMI		0
13	Body mass index	2	Allergy, Diabetes		0
14	Diabetes	2	Allergy		0
15	Antibiotic allergy	2			0

Table 2: The order in which *Arthronet* variables are eliminated by generalized variable elimination, showing the dimension of each eliminated variable, the relevant observed and unobserved variables in the elimination step, and the count of irrelevant variables. It is the many irrelevant variables before *Infection* is eliminated that leads to implicit coalescence.

To further improve the efficiency of this comparison-and-merge step, we could leverage the fact that a new node only needs to be compared to all *distinct* nodes that have been generated so far, which could be *much* less than the number of linear potentials generated.

Example: Arthronet. To illustrate the complementary effect of implicit and explicit coalescence in simplifying a strategy graph, we consider how the generalized algorithm solves the Arthronet influence diagram shown in Figure 6 and described in Section 3.3.2.

Recall that in solving this problem, the classic variable elimination algorithm must eliminate the unobserved chance variable for *Infection* first. Once this variable is eliminated, *all* of the remaining variables are relevant in *every* subsequent elimination step, and so the only form of coalescence performed by the traditional algorithm is explicit coalescence.

By contrast, generalized variable elimination can postpone elimination of the unobserved chance variable for *Infection*. As a result, when variables are eliminated in the order shown in Table 2, only the unobserved chance variable for *Infection*, and occasionally a decision variable, is relevant in the first eight elimination steps. Although the problem is small enough that it can be solved easily by both traditional and generalized variable elimination, our implementation of the generalized algorithm solves it more than an order of magnitude faster by leveraging independence relations among variables more effectively.

Generalized variable elimination not only speeds up problem solving, it constructs a strategy graph in which most duplicate subgraphs are *implicitly* merged. The first row of

Degree of simplification	Traditional VE Nodes	Traditional VE Arcs	Generalized VE Nodes	Generalized VE Arcs
Generated strategy graph with implicit coalescence only	149,695	232,638	162	391
With suboptimal (un-selected) branches removed	10,344	15,526	97	167
With zero-probability branches removed	1,141	1,408	85	131
With redundant nodes removed	892	1,159	64	110
With duplicate nodes merged by explicit coalescence	60	105	60	105

Table 3: For *Arthronet*, the table shows the size of the strategy graphs constructed by traditional and generalized variable elimination (VE) based on how much simplification is performed.

Table 3 compares the size of the strategy graphs constructed by traditional and generalized variable elimination when there is no simplification of the graph except for implicit coalescence. In this case, the traditional algorithm constructs a strategy tree. By contrast, generalized variable elimination constructs an equivalent but *much* smaller strategy graph that reflects the more efficient dynamic programming recurrence it solves.

The second and third rows of Table 3 compare the size of the strategy graphs after unreachable nodes and arcs are removed, which are nodes and arcs that are only reachable via an un-selected action or zero-probability observation. The fourth row shows the further compression achieved by removing redundant observation nodes. Finally, the last row of the table shows that when all forms of strategy graph simplification are performed, including explicit coalescence, both algorithms construct the same strategy graph shown in Figure 7, which has 60 nodes and 105 arcs.

Although both algorithms construct the same strategy graph, the generalized algorithm does so much more efficiently for two reasons. First, it solves a more efficient dynamic programming recurrence that better leverages independence relations among variables. Second, the initial strategy graph constructed by the generalized algorithm is *much* smaller, due to implicit coalescence, and so the overhead incurred for compressing it further by explicit coalescence and other simplification techniques is *much* less.

5. Coalescence in Exchange for Bounded-Error Approximation

Finally, we consider one last form of coalescence that can be used to simplify a strategy graph further in exchange for bounded-error approximation. As usual, how this form of coalescence is performed depends on elimination order.

5.1 Traditional Elimination Order

Recall that Section 3.3.2 describes a tie-breaking rule for action selection that leads to additional coalescence when variables are eliminated in a traditional order. It applies when there is more than one optimal action for a decision variable, given an instantiation $\mathbf{h}$ of the variables $\mathbf{H}$ in the domain of the utility potential generated by eliminating the decision variable. If one action gives rise to a duplicate node that can be merged with an existing node of the strategy graph, and another does not, the action that gives rise to a duplicate node is selected in order to merge the duplicate subgraphs, and simplify the strategy graph.

We can modify this tie-breaking rule to allow further coalescence in exchange for bounded-error approximation. For each instantiation $\mathbf{h}$ of the variables $\mathbf{H}$, we consider all actions that are bounded-suboptimal, which means the value of the action is within some threshold $\delta > 0$ of the optimal value for any action. If a bounded-suboptimal action gives rise to a duplicate node that can be merged with an existing node of the strategy graph, we choose the best bounded-suboptimal action that gives rise to a duplicate node, so that the duplicate nodes can be merged. If no bounded-suboptimal action gives rise to a duplicate node, we choose an optimal action, and add a new node to the strategy graph.

This approach to bounded-error approximation does not lead to speedup when potentials are represented by tables, as in the simplest implementation of variable elimination. In fact, it incurs some overhead. However, it can simplify a strategy graph.[7]

5.2 Non-Traditional Elimination Order

When variables are eliminated in a non-traditional order, which means the variable elimination algorithm generates PWLC utility potentials, we can use a related but different technique to encourage further coalescence in exchange for bounded-error approximation.

Recall the procedure described in Section 4.1.1 for pruning dominated linear potentials from an indexed family of sets of linear potentials that represents a PWLC utility potential. A linear potential is said to be *approximately dominated* if the linear program of Equation (12) for testing whether it is dominated has an objective value ϵ that is less than or equal to some small user-adjusted threshold $\delta > 0$, instead of zero. By pruning approximately dominated linear potentials, we create a bounded-suboptimal PWLC utility potential that is represented by smaller sets of linear potentials. We say that a PWLC utility potential $\psi(\mathbf{H}, B(\mathbf{U}))$ is bounded-suboptimal, with bound δ, if there is no state $\mathbf{h}$ and belief state $b(\mathbf{U})$ for which its value is suboptimal by more than δ.[8]

Because each linear potential corresponds to a node of the strategy graph, pruning approximately-dominated linear potentials leads to a smaller strategy graph. Moreover, this approach to bounded-error approximation also leads to computational speedup, since operations on PWLC utility potentials are faster when the sets of linear potentials that represent them are smaller. In fact, approximate pruning is a widely-used approach to achieving speedup in exchange for approximation when solving POMDPs, and it often provides substantial speedup in exchange for a relatively small degree of approximation.

To see how pruning approximately-dominated linear potentials has the effect of merging duplicate nodes, note that pruning an approximately-dominated linear potential means that for any belief state for which it gives an optimal value, we replace it with another linear potential that gives a value for this belief state that is sub-optimal by less than δ. Because each linear potential corresponds to a node of the strategy graph, pruning an approximately-dominated linear potential is equivalent to removing a corresponding optimal node from the

7. It may be possible to use bounded-error approximation for both speedup and coalescence if utility potentials are represented by data structures that leverage *context-specific independence* for both reduced storage and more efficient operations (Gómez & Cano, 2003; Cabañas, Gómez-Olmedo, & Cano, 2016).

8. Pruning approximately-dominated linear potentials ensures bounded suboptimality under the condition that any linear potential that is used to prune another linear potential cannot itself be pruned, since approximate dominance is not transitive. This condition is ensured by the pruning algorithms cited at the end of Section 4.1.1.

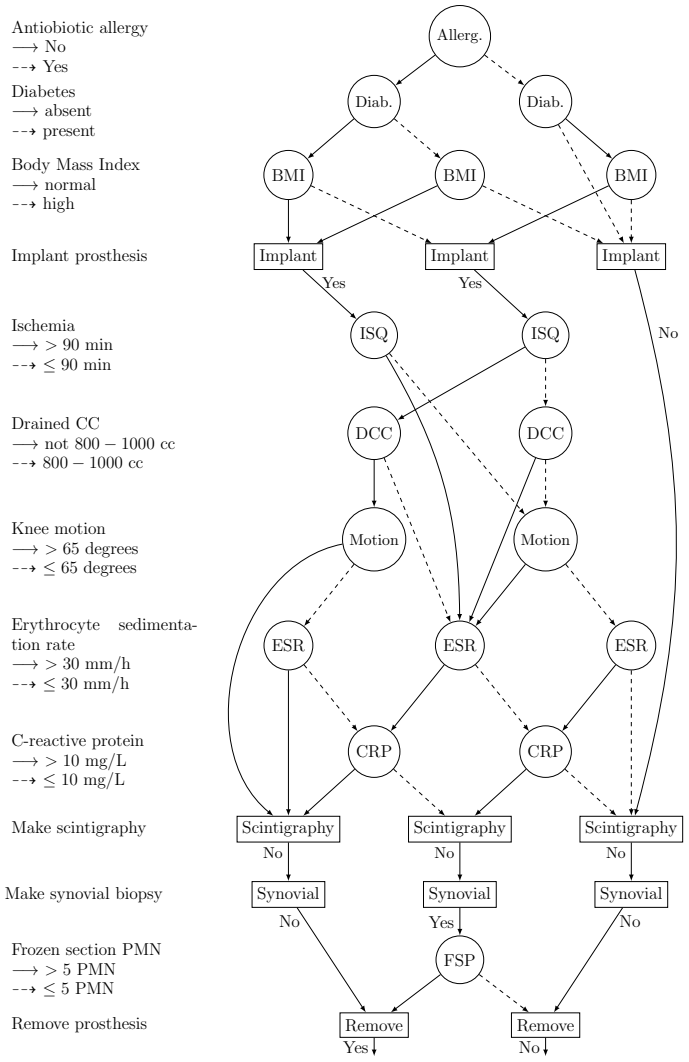


Figure 12: Smaller, bounded-suboptimal strategy graph for Arthronet problem.

strategy graph, and replacing it with a bounded sub-optimal node that is a duplicate of a node that is already in the strategy graph, allowing them to be *implicitly* merged.

This approach to bounded-error approximation leads to speedup as well as implicit coalescence, which reflects the fact that it is a form of approximate dynamic programming.

Example: Arthronet. Figure 12 shows a bounded-suboptimal strategy graph for the Arthronet problem that is constructed by generalized variable elimination when it prunes approximately-dominated linear potentials by checking whether the objective value ϵ of the linear program of Equation (12) is less than or equal to the threshold $\delta = 0.1$. The expected utility of this smaller strategy graph is 0.468 (when rounded to three decimal places) compared to the expected utility of the optimal strategy graph shown in Figure 7, which is 0.496. It is a relatively small decrease in expected utility in exchange for reducing the size of the strategy graph by half.

A conservative bound on the sub-optimality of a strategy constructed by this approach is easily obtained by multiplying the approximation parameter δ by the number of times an indexed family of sets of linear potentials that represents a PWLC utility potential is pruned using this technique. Obviously, δ can be adjusted to tighten or loosen this bound.

Model approximation and coalescence. We mention one more way in which it may be useful to perform additional coalescence in exchange for bounded-error approximation. For influence diagrams that model real-world problems, probabilities and utilities are often estimated and inexact. Given a model with inexact parameters, it may make sense to ignore differences of expected utility that are less than a threshold $\delta > 0$ that reflects the degree of model approximation. In that case, the threshold δ can be used to compress the strategy graph constructed by variable elimination in the way described above, that is, by ignoring differences of expected utility that are less than δ when doing so allows further coalescence.

Intuitively, a smaller strategy graph constructed by ignoring small differences in utility is more likely to include only those steps of a strategy that are justified by an inexact model.

6. Conclusion

We have introduced an approach to strategy representation for influence diagrams that represents a strategy as a graph, and simplifies the graph as much as possible so that it is easier to interpret and analyze. We have also shown how to modify a variable elimination algorithm for influence diagrams so that it constructs a strategy graph.

Among the techniques we have described for simplifying a strategy graph, we have given special attention to coalescence, that is, merging duplicate subgraphs. We have refined the traditional concept of coalescence by distinguishing among three forms it can take. Implicit coalescence is performed automatically by the variable elimination algorithm itself when it solves a dynamic programming recurrence that leverages independence relations among variables, especially Markovian independence from variables that represent previous history. Explicit coalescence merges duplicate subgraphs that are not merged implicitly. Because it is performed by extra steps that are added to the variable elimination algorithm, it incurs some overhead. Finally, coalescence can also be performed in exchange for bounded-error approximation. In this case, a bounded-suboptimal node is added to a strategy graph instead of an optimal node when the bounded-suboptimal node is a duplicate of a node already in the strategy graph. As a result, the duplicate nodes can be merged in order to simplify the strategy graph further, and more than would be otherwise possible.

Both traditional and generalized variable elimination construct the same strategy graph when they perform both implicit and explicit coalescence, in addition to other simplification techniques. But when solving influence diagrams with unobserved chance variables, which represent partially observed problems, the traditional algorithm must rely almost entirely on explicit coalescence. By contrast, the generalized algorithm can rely *much* more, and often almost entirely, on implicit coalescence. It has this advantage when solving partially observed problems because it can solve dynamic programming recurrences over belief states, and thus it can eliminate variables in an order that better leverages independence relations among variables, including variables that represent prior history. In many cases, eliminating variables in an order that better leverages variable independence also leads to computational speedup. As a result, implicit coalescence is often associated with faster problem solving.

Some of the techniques we use to simplify strategy graphs are similar to techniques used to simplify or "reduce" *decision diagrams* (Bryant, 1986; Bahar, Frohm, Gaona, Hachtel, Macii, Pardo, & Somenzi, 1993). In particular, two of the rules we use to simplify strategy graphs – removing redundant nodes and merging duplicate nodes – are essentially the same reduction rules used to simplify decision diagrams. Furthermore, the well-known effect of variable ordering on the size of a decision diagram is related to the effect that the ordering of observed chance variables can have on the size of a strategy graph, as discussed in Section 3.3.3. However, and by contrast to decision diagrams, the information precedence constraints of an influence diagram severely limit the possible re-orderings of decision and observed chance variables in a strategy graph.

The concept of a strategy graph generalizes the concept of a strategy tree for an influence diagram, and the concept of coalescence, as we have discussed. In addition, it is inspired by, and generalizes, the concept of a *policy graph* for a finite-horizon POMDP (Kaelbling et al., 1998). That is in keeping with the fact that the generalized algorithm not only generalizes the traditional variable elimination algorithm for influence diagrams, it also generalizes the value iteration algorithm for finite-horizon POMDPs. Both a strategy graph and a policy graph allow a strategy to be executed without updating a belief state, and both can be easier for a human user to understand. However, a policy graph has just one kind of node instead of two, since a node of a policy graph corresponds to an action and an arc corresponds to an observation. Because a strategy graph has distinct decision and observation nodes, it can more clearly represent the observational structure of a strategy when observations are factored into several variables, as is common for both influence diagrams and factored POMDPs (Boutilier & Poole, 1996). This advantage is illustrated by the strategy graphs for the Arthronet problem shown in Figures 7 and 12.

Influence diagrams have traditionally been used to solve relatively small problems, and, for small problems, strategy graphs can be easy to interpret and understand. But for larger and more complex problems, additional techniques may be needed to help users interpret more complex strategies. A promising direction for future research is to consider more sophisticated techniques for visualization and analysis of strategies that are represented by very large and complex strategy graphs.

Acknowledgments This research was supported by the National Science Foundation under Award IIS:RI #1718384. We thank the reviewers for comments and suggestions that have helped to improve this paper.

Appendix A. Addition of Piecewise-Linear and Convex Utility Potentials

In this paper, especially in Section 4.1.3, we describe an implementation of generalized variable elimination that assumes that if the utility potential generated in an elimination step is PWLC, it is relevant in the next elimination step. Under this assumption, there can never be more than one PWLC utility potential at a time, and so we never need to add two PWLC utility potentials. However, our earlier paper describes a version of generalized variable elimination that does not make this simplifying assumption (Hansen, 2021). Since it may need to add two PWLC utility potentials, we review this additional operation and explain how it affects strategy graph construction.

When a PWLC utility potential $\psi(\mathbf{H}, B(\mathbf{U}))$, represented by an indexed family of sets of linear utility potentials over $\mathbf{U}$, $\{\Gamma_{\mathbf{h}}\}_{\mathbf{h}\in sp(\mathbf{H})}$, is added to another PWLC utility potential $\psi'(\mathbf{H}', B(\mathbf{U}'))$, which is represented by an indexed family of sets of linear utility potentials over $\mathbf{U}'$, $\{\Gamma'_{\mathbf{h}'}\}_{\mathbf{h}'\in sp(\mathbf{H}')}$, the new PWLC utility potential $\psi''(\mathbf{H}'', B(\mathbf{U}''))$ is represented by an indexed family of sets of linear utility potentials over $\mathbf{U}''$, $\{\Gamma''_{\mathbf{h}''}\}_{\mathbf{h}''\in sp(\mathbf{H}'')}$, where each set is created as follows:

$$\Gamma''_{\mathbf{h}''} = Prune\left(\left\{\gamma+\gamma' | \gamma\in\Gamma_{\mathbf{h}''\downarrow\mathbf{H}}, \gamma'\in\Gamma'_{\mathbf{h}''\downarrow\mathbf{H}'}\right\}\right). \tag{24}$$

The corresponding strategy node function, $s_{\psi''}(\mathbf{H}'', B(\mathbf{U}''))$, is constructed as follows. For each linear potential γ'' in $\Gamma''_{\mathbf{h}''}$, where $\gamma'' = \gamma + \gamma'$ as specified by Equation (24), the associated strategy node, denoted $n(\gamma'')$, is the root of a strategy graph that is constructed by *concatenating* the strategy graph rooted at the node $n(\gamma)$ with the strategy graph rooted at the node $n(\gamma')$, where concatenation of strategy graphs is the same operation described in Section 3.2.5 for traditional variable elimination. Recall that the order in which strategy node functions are concatenated must be the reverse of the order in which their corresponding utility potentials were generated, in order to ensure that the resulting strategy graph respects the total ordering of decisions.

For example, if we assume that ψ comes before ψ' in temporal order, the strategy graphs rooted at $n(\gamma)$ and $n(\gamma')$ are concatenated by adding an arc from every terminal node of the strategy graph rooted at $n(\gamma)$ to $n(\gamma')$, which is the root node of the other strategy graph, and then setting $n(\gamma'')$ to the root node of the concatenated strategy graphs.

References

Bahar, R. I., Frohm, E. A., Gaona, C. M., Hachtel, G. D., Macii, E., Pardo, A., & Somenzi, F. (1993). Algebraic decision diagrams and their applications. In *Proc. of IEEE/ACM Int. Conf. on Computer-aided Design*, pp. 188–191. IEEE Computer Society Press.

Bielza, C., & Shenoy, P. P. (1999). A comparison of graphical techniques for asymmetric decision problems. *Management Science*, *45*(11), 1552–1569.

Boutilier, C., & Poole, D. (1996). Computing optimal policies for partially observable Markov decision processes using compact representations. In *Proc. of 13th National Conf. on Artificial Intelligence*, pp. 1168–1175.

Bryant (1986). Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, *C-35*(8), 677–691.

Cabañas, R., Gómez-Olmedo, M., & Cano, A. (2016). Using binary trees for the evaluation of influence diagrams. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, *24*(01), 59–89.

Cabañas, R., Cano, A., Gómez-Olmedo, M., & Madsen, A. (2013). Heuristics for determining the elimination ordering in the influence diagram evaluation with binary trees. In *12th Scandinavian Conf. on Artificial Intelligence: SCAI 2013, volume 257 of Frontiers in Artificial Intelligence and Applications*, pp. 65–74. IOS Press.

Cassandra, A., Littman, M., & Zhang, N. (1997). Incremental pruning: A simple, fast, exact method for partially observable Markov decision processes. In *Proc. of 13th Conf. on Uncertainty in Artificial Intelligence*, pp. 54–61.

Dechter, R. (2000). A new perspective on algorithms for optimizing policies under uncertainty. In *Proc. 5th Int. Conf. on Artificial Intelligence Planning Systems*, pp. 72–81.

Dechter, R. (2019). *Reasoning with Probabilistic and Deterministic Graphical Models: Exact Algorithms*. Morgan & Claypool.

Gómez, M., & Cano, A. (2003). Applying numerical trees to evaluate asymmetric decision problems. In Nielsen, T. D., & Zhang, N. L. (Eds.), *Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, pp. 196–207, Berlin, Heidelberg. Springer.

Hansen, E. (2021). An integrated approach to solving influence diagrams and finite-horizon partially observable decision processes. *Artificial Intelligence*, *294*, 1–48.

Hansen, E., & Bowman, T. (2020). Improved vector pruning in exact algorithms for solving POMDPs. In *Proc. of 36th Conference on Uncertainty in Artificial Intelligence (UAI)*.

Hansen, E., Shi, J., & Khaled, A. (2016). A POMDP approach to influence diagram evaluation. In *Proc. of 25th International Joint Conf. on Artificial Intelligence (IJCAI-16)*, pp. 3124–3132. AAAI Press.

Howard, R., & Matheson, J. (1981). Influence diagrams. In Howard, R., & Matheson., J. (Eds.), *The Principles and Applications of Decision Analysis*, pp. 719–762, Menlo Park, CA.

Jensen, F., Jensen, F., & Dittmer, S. (1994). From influence diagrams to junction trees. In *Proc. of 10th Conf. on Uncertainty in Artificial Intelligence*, pp. 367–373.

Jensen, F., & Nielsen, T. (2007). *Bayesian Networks and Decision Graphs* (2nd edition). Springer, New York.

Jensen, F., & Nielsen, T. (2011). Probabilistic decision graphs for optimization under uncertainty. *4OR - A Quarterly Journal of Operations Research*, *9*(1), 1–28.

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

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

León, D. (2011). A probabilistic graphical model for total knee arthroplasty. Master's thesis, Dept. Artificial Intelligence, UNED, Madrid, Spain.

Luque, M., Arias, M., & Díez, F. (2017). Synthesis of strategies in influence diagrams. In *Proc. of 33rd Conf. on Uncertainty in Artificial Intelligence (UAI-17)*.

Luque, M., & Díez, F. (2010). Variable elimination for influence diagrams with super-value nodes. *International Journal of Approximate Reasoning*, *51*(6), 615 – 631.

Luque, M., Díez, F., & Disdier, C. (2016). Optimal sequence of tests for the mediastinal staging of non-small cell lung cancer. *BMC Medical Informatics and Decision Making*, *16*(9), 1 – 14.

Marinescu, R., Razak, A., & Wilson, N. (2012). Multi-objective influence diagrams. In *Proc. of 28th Conf. on Uncertainty in Artificial Intelligence*, pp. 574–583.

Marinescu, R., Razak, A., & Wilson, N. (2017). Multi-objective influence diagrams with possibly optimal policies. In *Proc. of 31th AAAI Conf. on Artificial Intelligence*, pp. 3383–3389.

Mauá, D., & Cozman, F. G. (2016). Fast local search methods for solving limited memory influence diagrams. *International Journal of Approximate Reasoning*, *68*, 230–245.

Mauá, D., de Campos, C., & Zaffalon, M. (2012). Solving limited memory influence diagrams. *Journal of Artificial Intelligence Research*, *44*, 97–140.

Ndilikilikesha, P. (1994). Potential influence diagrams. *International Journal of Approximate Reasoning*, *10*(3), 251–285.

Nilsson, D., & Hohle, M. (2001). Computing bounds on expected utilities for optimal policies based on limited information. Tech. rep. 94, Danish Informatics Network in the Agricultural Sciences.

Olmsted, S. (1983). *On representing and solving decision problems.* Ph.D. thesis, Stanford University.

Segal, I., & Shahar, Y. (2009). A distributed system for support and explanation of shared decision-making in the prenatal testing domain. *Journal of Biomedical Informatics*, *42*(2), 272–286.

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

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

Smith, J., Holtzman, S., & Matheson, J. (1993). Structuring conditional relationships in influence diagrams. *Operations Research*, *41*, 280–297.

Tatman, J. (1986). *Decision Processes in Influence Diagrams: Formulation and Analysis.* Ph.D. thesis, Stanford University.

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

Walraven, E., & Spaan, M. (2017). Accelerated vector pruning for optimal POMDP solvers. In *Proc. of 31st AAAI Conf. on Artificial Intelligence*, pp. 3672–3678.