Strategy Representation and Compression for Influence Diagrams

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

Abstract

Influence diagrams are graphical models used to represent and solve decision-making problems under uncertainty. The solution of an influence diagram, a *strategy*, is traditionally represented by tables that map histories to actions; it can also be represented by an equivalent *strategy tree*. We show how to compress a strategy tree into an equivalent and more compact *strategy graph*, making strategies easier to interpret and understand. We also show how to compress a strategy graph further in exchange for bounded-error approximation.

Introduction

An influence diagram (Howard and Matheson 1981; Jensen and Nielsen 2011) is a compact graphical representation of a decision problem under uncertainty that shows the dependencies among problem variables more clearly than an equivalent decision tree. While the size of a decision tree grows exponentially in the number of variables of the problem, the size of an influence diagram increases no more than quadratically in the number of variables.

Although an influence diagram (ID) provides a compact representation of a decision problem, it does not provide a similarly compact representation of a solution, called a *strategy*. The traditional representation of a strategy is a mapping from histories to decisions, where a history is an instantiation of decision and observation variables; the mapping is stored in a table, with one table for each decision variable. A drawback is that the size of these tables grows exponentially in the number of decision and observation variables, which reflects the no-forgetting property of IDs.

It is also possible to represent a strategy as a tree, called a *strategy tree*. This representation has a long history that dates back to methods for solving an ID by unfolding it into an equivalent decision tree: a strategy tree is a subtree of the decision tree that only includes the best action for each decision node. Compared to representing a strategy by tables, a strategy tree can make it easier to understand and explain a strategy, which is especially important in domains such as medical decision making (Segal and Shahar 2009; Luque, Dez, and Disdier 2016). It is also possible to compress a strategy tree by only including branches that are reachable with positive probability, and by eliminating nodes that are not relevant given the values of ancestors of the node in the strategy tree (Luque, Arias, and Diez 2017).

In this paper, we consider a complementary approach to strategy compression that leverages the fact that a strategy tree that contains repeated subtrees can be represented more compactly – typically, *much* more compactly – by an equivalent graph, which we call a *strategy graph*. Representing a strategy as a graph has the advantage that it makes the strategy much easier to understand and analyze, which can help make IDs more accessible and useful in practice.

We introduced the concept of a strategy graph in a recent paper that proposed using techniques for solving partially observable Markov decision processes (POMDPs) to improve the scalability of algorithms for solving IDs (Hansen, Shi, and Khaled 2016). The concept of a strategy graph was inspired by the concept of a *policy graph* as a representation of a policy for a POMDP (Kaelbling, Littman, and Cassandra 1998). Our earlier paper focused on POMDP techniques for solving IDs, however, and did not describe how to modify a traditional algorithm for solving IDs so that it constructs a strategy graph. In this paper, we generalize the concept of a strategy graph in a way that better fits IDs, and describe in detail how a traditional algorithm for solving IDs – we consider variable elimination – can construct a solution that takes the more compact form of a strategy graph. We also propose an approach to approximation that allows further compression of a strategy graph in exchange for bounded-error approximation. Although adoption of this representation of a strategy does not speed up algorithms for solving IDs, it can make the solution constructed by these algorithms easier to interpret and understand.

Background

In describing IDs, we adopt the following notation. Variables are denoted by capital letters, sometimes followed by a subscript, e.g., X_i. Values taken by variables are represented by a lower-case letter, e.g., x_i is a value of X_i. Sets of variables $\{X_1, \ldots, X_n\}$ are represented by a bold capital letter, e.g., $\mathbf{X}$. Configurations of $\mathbf{X}$, where each variable X_i takes on a value x_i, are represented by a lower-case bold letter, such as $\mathbf{x}$. We assume that all variables have a finite state space, where $sp(X)$ denotes the set of possible values of a variable X. By extension, for a set of variables $\mathbf{X}$, we have $sp(\mathbf{X}) = \prod_{x \in \mathbf{X}} sp(X)$, which is the Cartesian product of the individual state spaces.

Influence diagrams

An ID is defined on a directed acyclic graph with three kinds of nodes, as illustrated by the example in Figure 4. *Chance nodes*, drawn as circles, represent random (or chance) variables, $\mathbf{C} = \{C_1, \ldots, C_m\}$, as in a Bayesian network. *Decision nodes*, drawn as rectangles, represent decision variables, $\mathbf{D} = \{D_1, \ldots, D_n\}$. *Value nodes*, $\mathbf{V} = \{V_1, \ldots, V_q\}$, drawn as diamonds, represent the preferences of the decision maker, and have no children.

Each chance variable $C_i \in \mathbf{C}$ is associated with a conditional probability distribution, $P(C_i|pa(C_i))$, where $pa(C_i)$ denotes the set of parent variables of C_i in the graph. Each value node $V_i \in \mathbf{V}$ is associated with a utility function $U_i(pa(V_i))$ that assigns a scalar value to each instantiation of the parent variables $pa(V_i)$. Given multiple value nodes, we assume that the total utility is their sum.

Each decision variable $D_k \in \mathbf{D}$ has a parent set $pa(D_k)$, denoting the variables whose values are observed before the decision is made. We assume that IDs are regular and no-forgetting, which means there is a temporal ordering of the decision variables, denoted $D_1, D_2, \ldots, D_n$, and a decision node and its parents are parents of all subsequent decision nodes. We let $\mathbf{I}_0 \subseteq \mathbf{C}$ denote the set of chance variables observed before the first decision D_1. Similarly, $\mathbf{I}_k \subseteq \mathbf{C}$ denotes the set of chance variables observed between decisions D_k and D_{k+1}, and $\mathbf{I}_n \subseteq \mathbf{C}$ denotes the set of chance variables that are never observed. Thus there is a partial temporal ordering: $\mathbf{I}_0 \prec D_1 \prec \mathbf{I}_1 \prec \ldots \prec D_n \prec \mathbf{I}_n$.

An ID is solved by finding a strategy that has maximum expected utility (MEU), which is equal to

$$\sum_{\mathbf{I}_0} \max_{D_1} \ldots \max_{D_n} \sum_{\mathbf{I}_n} \left(\prod_{i=1}^{m} P(C_i|pa(C_i)) \sum_{j=1}^{q} U_j(pa(V_j)) \right). \quad (1)$$

The traditional representation of a strategy for an ID is a list of decision rules $\Delta = (\delta_1, \ldots, \delta_n)$, one for each decision variable $D_i \in \mathbf{D}$, where a decision rule is a mapping, $\delta_i : sp(pa(D_i)) \rightarrow sp(D_i)$, that prescribes an action for each instantiation of the parent variables. By definition, any chance variable that is a parent of a decision variable is *observable*, which means its value is known at the time the decision is made; from now on, we call observable chance variables *observation variables*.

Variable elimination

The approach to strategy representation we propose can be used by any algorithm for solving IDs. We illustrate its use in a variable elimination (VE) algorithm. VE evaluates Equation (1) by progressively eliminating variables and replacing probability and utility functions that mention these variables with equivalent functions that do not (Jensen and Nielsen 2007; Koller and Friedman 2009; Dechter 2000).

In order to simplify the operations needed to evaluate Equation (1), VE algorithms reformulate Equation (1) using so-called "potentials." A probability potential is denoted $\phi : sp(dom(\phi)) \rightarrow [0, 1]$, where $dom(\phi)$ denotes the set of all variables involved in the potential ϕ. Similarly, a utility potential denoted by ψ is a mapping $\psi : sp(dom(\psi)) \rightarrow \Re$. The probability and utility functions given in the initial specification of an ID can themselves be viewed as potentials. For each chance variable $X_i \in \mathbf{C}$, the conditional probability distribution $P(X_i|pa(X_i))$ is equivalent to a probability potential $\phi : sp(\{X_i\} \cup pa(X_i)) \rightarrow [0, 1]$. For each value node V_j, the corresponding utility function U_j is equivalent to a utility potential that is denoted $\psi : sp(pa(V_j)) \rightarrow \Re$.

When the original conditional probability functions and utility functions are viewed as potentials, Equation (1) can be reformulated as follows,

$$MEU = \sum_{\mathbf{I}_0} \max_{D_1} \ldots \max_{D_n} \sum_{\mathbf{I}_n} \Big(\prod \Phi \sum \Psi \Big), \quad (2)$$

where Φ denotes the set of probability potentials, Ψ denotes the set of utility potentials, and the expression $\prod \Phi(\sum \Psi)$ is the product of all probability potentials multiplied by the sum of all utility potentials. Each time the VE algorithm eliminates a variable from this expression, the sets of probability and utility functions are replaced by equivalent sets of potentials that do not depend on the eliminated variable.

The pseudocode for VE is given by Algorithm 1. It progressively eliminates variables from Equation (2) using two operators: sum-marginalization, which eliminates chance variables, and max-marginalization, which eliminates decision variables. For convenience, we let $\{X_1, \ldots, X_p\} = \mathbf{C} \cup \mathbf{D}$ denote the set of all variables. The variables must be eliminated in reverse order of the partial ordering imposed by the information constraints, which is called a *strong elimination order*. That is, VE first sum-marginalizes $\mathbf{I}_n$, then max-marginalizes D_n, then sum-marginalizes $\mathbf{I}_{n-1}$, etc.

After a variable X_i is selected for elimination, the relevant potentials are identified and combined, where a potential is relevant if the selected variable X_i is in its domain. Probability potentials are combined by element-wise multiplication; utility potentials are combined by element-wise addition, normalized by the probability component. The selected variable is eliminated by summation for a chance variable, and by maximization for a decision variable. Elimination of a variable creates one new probability potential and one new utility potential. The notation $X_i = x_i$ in lines 31 and 40 means the maximizing decision $x_i \in X_i$ in line 30 is used to compute the new potentials. When the last variable is eliminated, the VE algorithm returns a potential with no arguments (i.e., a constant) that is the value of Equation (2).

The pseudocode of Algorithm 1 differs from the standard VE algorithm only in the way it represents and constructs a strategy. We explain this difference in the rest of the paper.

Strategy representation and compression

Recall that the traditional representation of a strategy is a list of decision rules, $\Delta = (\delta_1, \ldots, \delta_n)$, one for each decision variable $D_i \in \mathbf{D}$, where each rule is a mapping, $\delta_i : sp(pa(D_i)) \rightarrow sp(D_i)$, that assigns a decision to each instantiation of the parent variables. Typically, each decision rule is represented by a table, where the dimensionality of the table is equal to the number of parent variables of the corresponding decision node. A table that represents a decision rule is created each time VE eliminates a decision variable.

Algorithm 1: Variable elimination with strategy graph construction

Input: An influence diagram and a strong elimination order of its variables: $\mathbf{X} = \{X_1, X_2, \ldots, X_p\}$
Output: An optimal strategy graph and MEU

// Initialize sets of probability and utility potentials
$\Phi \leftarrow \{P(C_i|pa(C_i))|C_i \in \mathbf{C}\}$
$\Psi \leftarrow \{U(pa(V_j)|V_j \in \mathbf{V}\}$
DecisionVariableEliminated $\leftarrow$ false
for $i \leftarrow p$ **downto** 1 **do** // eliminate variable X_i
  // Get potentials that depend on X_i
  $\Phi_{X_i} \leftarrow \{\phi \in \Phi | X_i \in dom(\phi)\}$
  $\Psi_{X_i} \leftarrow \{\psi \in \Psi | X_i \in dom(\psi)\}$
  // combine potentials
  $\phi_{X_i} \leftarrow \prod \Phi_{X_i}$ // joint probability
  $\psi_{X_i} \leftarrow \prod \Phi_{X_i} (\sum \Psi_{X_i})$ // expected utility
  if X_i *is a chance variable* **then**
    // eliminate X_i by sum marginalization
    $\mathbf{Y} \leftarrow \text{dom}(\psi_{X_i}) \backslash X_i$ // input variables for ψ_{X_i}
    foreach $\mathbf{y}$ *of* $\mathbf{Y}$ **do**
      $\psi'_{X_i}(\mathbf{y}) \leftarrow \sum_{X_i} \psi_{X_i}(X_i, \mathbf{y})$
      if *DecisionVariableEliminated* **then**
        // save strategy graph
        $\delta_{X_i}(\mathbf{y}) = \emptyset$
        foreach x_i *of* X_i **do**
          $\delta_{X_i}(\mathbf{y}) \leftarrow \delta_{X_i}(\mathbf{y}) \cup \{(x_i, \delta_{X_{i+1}}(x_i, \mathbf{y})\}$
        end
        // $\mathcal{N}$ is nodes of current strategy graph
        $\delta_{X_i}(\mathbf{y}) \leftarrow \text{Compress}(\delta_{X_i}(\mathbf{y}), \mathcal{N})$
    end
    $\phi'_{X_i} \leftarrow \sum_{X_i} \phi_{X_i}$
  else if X_i *is a decision variable* **then**
    // eliminate X_i by max marginalization
    $\mathbf{Y} \leftarrow \text{dom}(\psi_{X_i}) \backslash X_i$ // input variables for ψ_{X_i}
    foreach $\mathbf{y}$ *of* $\mathbf{Y}$ **do**
      $x_i \leftarrow \arg\max_{X_i} \psi_{X_i}(X_i, \mathbf{y})$
      $\psi'_{X_i}(\mathbf{y}) \leftarrow \psi_{X_i = x_i}(X_i, \mathbf{y})$
      // save strategy graph
      if *DecisionVariableEliminated* **then**
        $\delta_{X_i}(\mathbf{y}) \leftarrow \{(x_i, \delta_{X_{i+1}}(x_i, \mathbf{y}))\}$
      else
        $\delta_{X_i}(\mathbf{y}) \leftarrow \{(x_i, nil)\}$
        DecisionVariableEliminated $\leftarrow$ true
      $\delta_{X_i}(\mathbf{y}) \leftarrow \text{Compress}(\delta_{X_i}(\mathbf{y}), \mathcal{N})$
    end
    $\phi'_{X_i} \leftarrow \phi_{X_i = x_i}$
  // update sets of potentials
  $\Phi \leftarrow (\Phi \backslash \Phi_{X_i}) \cup \{\phi'_{X_i}\}$
  $\Psi \leftarrow (\Psi \backslash \Psi_{X_i}) \cup \{\frac{\psi'_{X_i}}{\phi'_{X_i}}\}$
end
// return strategy graph and MEU
return $\{ \delta_{X_1}, \psi'_{X_1} \}$

Strategy graph representation

We propose to represent a strategy as a graph, called a *strategy graph*. A strategy tree is a special case of a strategy graph, of course, and so the following definition applies to both strategy trees and strategy graphs.

Definition 1. *A* strategy graph *represents a strategy for an influence diagram in the form of a rooted directed acyclic graph with two kinds of nodes:*

1. *A* decision node *corresponds to a decision variable of the ID, and has a single outgoing arc labeled by the choice of an action. The outgoing arc leads to a successor node, or to* nil *if it does not have a successor.*
2. *An* observation node *corresponds to an observation variable of the ID, and each of its outgoing arcs is labeled by a non-empty subset of the states of the variable (i.e., the observations), where each state labels at most one arc.*

A strategy graph specifies a strategy, as follows. Beginning from the root of the graph, a path from the root to a leaf is followed based on the observed state of each observation node along the path, and the sequence of actions taken is determined by the labels on the outgoing arcs from the decision nodes on the path.

Examples of strategy graphs are shown in Figures 5 and 6. Note that the ordering of variables is the same on every path from the root to a leaf; it is the reverse of the order in which the variables are eliminated in solving the ID. However, it is not necessary for every path from the root to a leaf to include a node for every decision and observation variable of the ID. Note also that every path from the root to a leaf ends at an artificial *nil* node, which is the only type of node that does not have a successor. The *nil* node has no function but to serve as a placeholder so that the outgoing edge from the last decision node can have a successor node. It could easily be left out of a display of the strategy graph to improve clarity.

Algorithm 1 shows the pseudocode for a variable elimination algorithm that has been modified to construct a strategy graph, with the part of the pseudocode related to strategy graph construction highlighted. The strategy graph is constructed recursively from its leaves to its root, as the problem is solved. Note that the algorithm does not begin to construct a strategy graph until a decision variable has been eliminated, since the set $\mathbf{I}_n$ of chance variables eliminated before the first decision variable is eliminated are unobservable, and a strategy is not conditioned on them.

In the traditional VE algorithm, a strategy is represented as a mapping from histories to actions. That is, for each decision variable X_i, there is a mapping, $\delta_{X_i} : sp(\mathbf{Y}) \rightarrow sp(X_i)$, that prescribes an action $x_i \in X_i$ for each instantiation $\mathbf{y}$ of the parent variables $\mathbf{Y}$ of X_i in the ID. By contrast, for each decision *and observation variable* X_i of an ID, we let $\delta_{X_i} : sp(\mathbf{Y}) \rightarrow \mathcal{N}$ represent a mapping from each instantiation $\mathbf{y}$ of $\mathbf{Y}$ to a node of a strategy graph, where $\mathcal{N}$ denotes the set of nodes of the strategy graph. Note that different instantiations can map to the same node of a strategy graph; in that way, there can be compression. Note also that each node of a strategy graph is the root of a subgraph that itself can be viewed as a strategy graph, since the definition of a strategy graph is recursive.

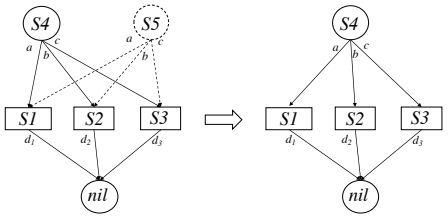

Figure 1: On the left is a strategy graph just after node $S5$ has been added; on the right is the compressed strategy graph.

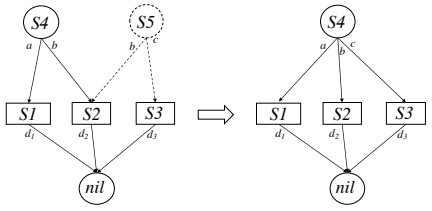

Figure 2: On the left, nodes $s4$ and $s5$ have zero-probability branches that are not shown, allowing them to be merged.

In the pseudocode of Algorithm 1, especially lines 19 - 22, 34, and 36, we represent a node of the strategy graph by a set of ordered pairs, with one pair for each state of the corresponding variable. The first element of a pair represents a state of the variable (and thus an outgoing edge from the node), and the second represents the successor node for this state/edge, itself represented by a set of ordered pairs. (For a leaf node of the strategy graph, the successor could be *nil*.)

Initially, a new node of the strategy graph is created for each instantiation $\mathbf{y}$ of the input variables $\mathbf{Y}$ of the utility potential ψ_{X_i} for a variable X_i. As the result of compression, however, $\delta_{X_i}(\mathbf{y})$ can point to the same node of the strategy graph for different instantiations $\mathbf{y}$ of $\mathbf{Y}$, as we next explain.

Strategy compression

The VE algorithm begins to construct the strategy graph after the unobservable chance variables $\mathbf{I}_n$ have been eliminated. From that point, a new node is created each time the utility of an instantiation $\mathbf{y}$ of the variables $\mathbf{Y}$ is computed for the variable X_i currently being eliminated.

Each newly-created node of the strategy graph has outgoing edges that lead to previously-created nodes of the strategy graph, or else to *nil*. After a node is added to the graph, a procedure is invoked, called *Compress* in the pseudocode, that attempts to compress the strategy graph. The details of the procedure are shown in Algorithm 2.

Algorithm 2 uses two kinds of rules to compress a strategy graph. The first kind, consisting of rules 1(a) through 1(c), plus rule 2(a), considers only the newly-added node, its outgoing edges, and their successor nodes in the strategy graph. Note that rules 1(a) through 1(c) are also used by Luque, Arias, and Diez (2017) to compress a strategy tree. These rules leverage reachability by removing zero-probability branches; they also leverage *context-specific independence* (Boutilier et al. 1996) to remove nodes that are conditionally irrelevant on one path, but not others. These rules can transform a strategy tree into a compressed strategy tree, but they cannot transform a tree into a graph.

The second kind of rule, consisting of rules 1(d) and 2(b), transforms a strategy tree into a strategy graph; this form of compression is the primary contribution of our paper. These rules consider not only the newly-added node, its outgoing edges, and their successor nodes in the strategy graph; they also consider all of the other nodes of the strategy graph. A newly-created node is merged with an existing node of the strategy graph if they have the same outgoing edges, and the same successor nodes for each edge. Essentially, a new node

---

Algorithm 2: Strategy graph compression.

Input: For variable X_i and instantiation $\mathbf{y}$ of ancestor variables $\mathbf{Y}$, the input is (a) a newly-created node n of a strategy graph, and (b) all the other nodes, $\mathcal{N}$, of the strategy graph.

Output: Compressed representation of the strategy graph rooted at node n.

1. If X_i is an observation variable:
   (a) (Remove zero-probability branches): For the newly-created node n, remove an observation x_i and corresponding branch if it has zero probability given instantiation $\mathbf{y}$ of $\mathbf{Y}$.
   (b) (Merge branches with the same predecessor and successor nodes): If two outgoing edges from the newly-created node n have the same successor node, they can be replaced by a single outgoing edge that is labeled by the labels of both original edges, with the interpretation that this edge is followed if either of the conditions corresponding to the labels are true.
   (c) (Remove irrelevant observation nodes): If the newly-created node n has only one outgoing edge, then the corresponding observation is irrelevant in this context, and the node can be removed and replaced by a pointer to its successor node.
   (d) (Merge isomorphic subgraphs): If the newly-created node n corresponding to observation variable X_i is identical to a node n' already in the strategy graph, in the sense that for every outgoing edge labeled by the same state x_i of X_i for both n and n', the successor node is the same, then nodes n and n' can be merged. The merged node has every outgoing edge and successor node that is part of either n or n'.
2. Else if X_i is a decision variable:
   (a) (Remove no-op actions): If the newly-created decision node n is a no-op, as could be the case if the utility of every action is worse than the utility of doing nothing, then it can be removed and replaced by a pointer to the successor node of its outgoing edge.
   (b) (Merge isomorphic subgraphs): If the newly-created decision node n has the same action for its outgoing edge, and the same successor node, as another node n' in the strategy graph, the two nodes can be merged.

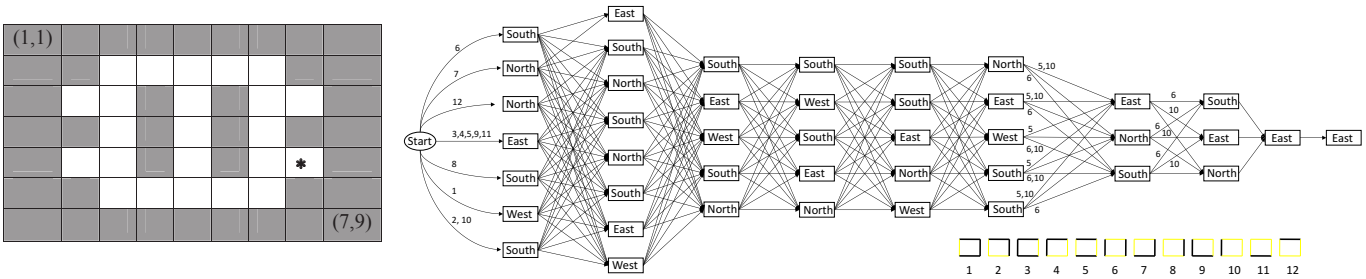

Figure 3: (a) Maze for ten-stage maze navigation problem and (b) optimal strategy graph.

is merged into an existing node of the strategy graph when the two nodes represent identical subgraphs of the strategy graph. But it is important to note that entire subgraphs do not need to be compared; the subgraphs are identical if the two nodes have the same outgoing edges and successor nodes.

Figure 1 illustrates the application of rule 2(d); it shows a newly-added node that is merged with an equivalent node already in the strategy graph. Figure 2 illustrates the interaction of this rule with rule 1(a), which prunes zero-probability branches. Because utility is not affected by the successor node of a zero-probability branch, a zero-probability branch can play the role of a "wildcard" that can be matched to anything, allowing additional compression.

It is obvious that the compression rules of Algorithm 2 preserve the equivalence of a strategy. They can also lead to dramatic compression, as the results presented later in the paper show. But the degree of compression is problem-dependent, and often depends on other factors too.

For example, variable ordering matters. If there are several observation variables between one decision variable and the next, then the order in which they are eliminated by the VE algorithm can affect the size of the strategy graph. A similar effect is well-known for ordered binary decision diagrams, where variable ordering can have a significant effect on the the size of a decision diagram, that is, its degree of compression (Bollig and Wegener 1996).

Elimination-ordering heuristics for VE algorithms typically focus on improving the efficiency with which an ID is solved. But if it is also desirable to minimize the size of the resulting strategy graph, the effect of the elimination order on the size of the strategy graph may also need to be considered. For IDs that are easily solved, it may be useful to solve the ID with several different elimination orders, in the attempt to find one that leads to the smallest strategy graph.

The degree of compression can also depend on tie-breaking issues, although these effects are more subtle. Consider the possibility of adding two different nodes to a strategy graph that have the same utility for the same instantiation $\mathbf{y}$ of $\mathbf{Y}$, and variable X_i. Further, suppose that when a choice must be made of which of these nodes to add to the strategy graph, neither one matches a node already in the strategy graph, but one turns out to match a node that is later added to the strategy graph, and one does not. In this case, the tie has to be broken without knowing which way of breaking the tie will lead to greater compression.

Bounded-error approximation

A strategy graph created by the compression procedure of Algorithm 2 may be compressed even further in exchange for bounded-error approximation. We next describe a very simple implementation of this idea.

When eliminating a decision variable X_i, consider the possibility of selecting a sub-optimal action for an instantiation $\mathbf{y}$ of $\mathbf{Y}$, where the sub-optimality is bounded by some threshold $\epsilon > 0$. If the sub-optimal action creates a new node for the strategy graph that can be merged into an existing node, and the optimal action does not, then choosing the sub-optimal action will result in additional compression of the strategy graph, in exchange for bounded-error approximation. The threshold ϵ could be used to derive a suboptimality bound. However, a *much* tighter bound can usually be found by simply comparing the expected utility computed by VE when this approach to approximation is used, and the optimal expected utility.

Unlike other approaches to bounded-error approximation, the motivation for this technique is not to speed up computation, or improve scalability, since it does neither. (Importantly, it does not incur extra overhead either.) Instead, the tradeoff this approach offers between approximation and compression of the strategy graph may be useful as a form of sensitivity analysis, or as an approach to solving IDs with imprecise parameters. For example, it can be used for sensitivity analysis by showing how a strategy can be compressed with limited loss of utility; the part of the strategy graph that is eliminated by simplification can be viewed as the less important part of the strategy graph.

This approach could also be useful for IDs with imprecise parameters. For example, to model such problems, Cabanas et al. (2017) consider *interval-valued IDs* where the probabilities and utilities of an ID are not represented exactly; instead, they are represented by intervals that bound the uncertainty about the exact values of the parameters. In this framework, the value of an ID cannot be determined exactly; instead, it is represented by an interval. As a result, a VE algorithm for interval-valued IDs may only find a set of potentially optimal strategies, instead of a single strategy, and a secondary criterion may be needed to choose a strategy from this set. Our approach could be used to select the strategy from this set that is most compact, and easiest to understand.

Examples and analysis

We consider some examples that illustrate the benefits of compressing a strategy tree into a strategy graph.

Maze navigation Figure 3(a) shows a partially observable navigation problem introduced in previous work on limited-memory IDs (Nilsson and Hohle 2001). For a randomly placed robot, the objective is to reach a goal state marked by a star within ten steps. At each step, the robot receives one of 12 possible observations that are shown in the lower right of Figure 3(b), where each observation represents a different configuration of the surrounding walls. After receiving an observation, the robot moves to a neighboring cell in one of the 4 possible directions of the compass, which means there are 4 possible actions. It follows that there are 48^{10} different possible histories over ten stages!

This huge number of possible histories makes it impossible to solve this problem using a traditional algorithm for IDs. But the problem is easily solved by an exact POMDP solver, or by an ID solver that uses POMDP techniques (Hansen, Shi, and Khaled 2016). The POMDP approach finds an optimal strategy (policy) graph that takes the relatively simple form shown in Figure 3(b). The primary reason for showing this strategy graph in this paper is that it illustrates how dramatic the degree of compression can be; in this case, a strategy tree with 48^{10} nodes has been compressed into an equivalent graph with only 44 nodes!

In the POMDP framework, the convention is that each node of a policy graph represents both a decision (which is the label of the node) and a subsequent observation, where the possible observations are represented by the labels on the outgoing edges from the node. The policy graph shown in Figure 3(b) reflects this convention. It makes sense to follow this convention when representing a policy for a POMDP, since the POMDP model assumes that each action is immediately followed by an observation. For the IDs considered in the rest of this section, however, we create strategy graphs that have distinct decision and observation nodes. This alternative representation, which is the strategy representation explained earlier in the paper, makes it easier to allow several observation nodes between successive decision nodes.

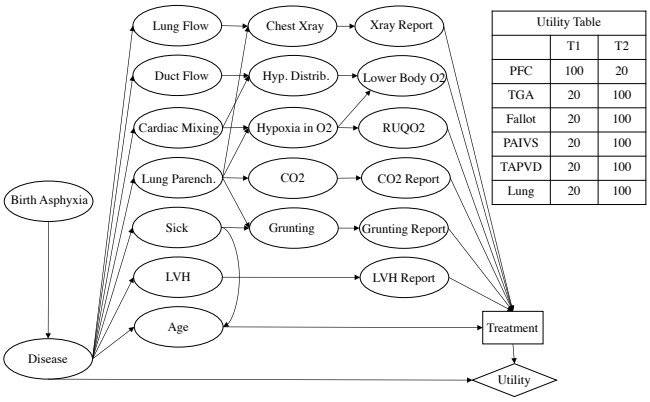

Figure 4: Influence diagram for diagnosis/treatment problem based on the *CHILD* belief network, with utility table.

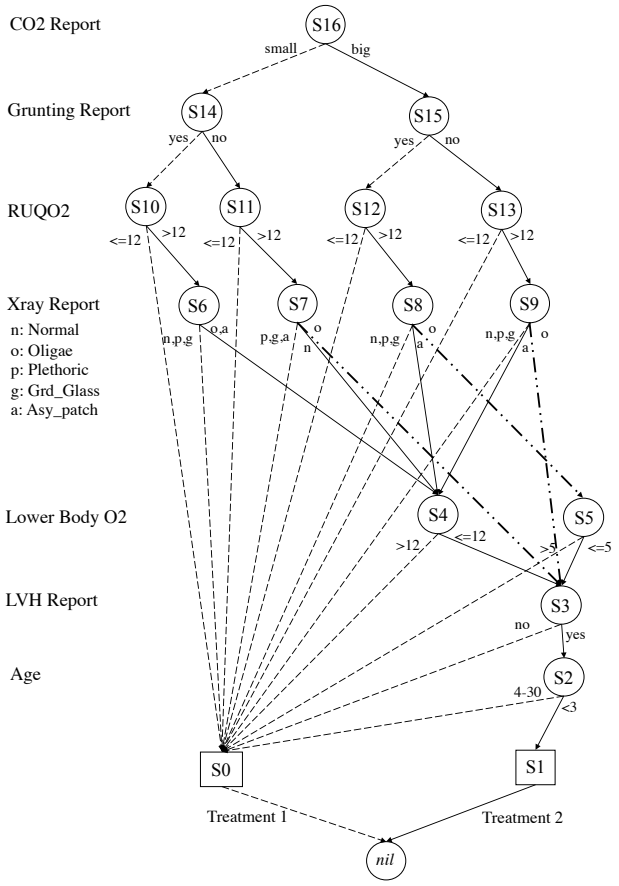

Figure 5: Optimal strategy graph for the *Child* influence diagram.

	Strategy Tree			Strategy
	All	Reach.	Compr.	graph
CO2Report	2	2	2	1
GruntingReport	4	4	4	2
RUQO2	12	12	4	4
XrayReport	60	60	8	4
LowerBodyO2	180	180	17	1
LVH_Report	360	360	17	1
Age	1,080	1,080	34	1
Treatment	2,160	1,080	34	2
Total	3,858	2,778	120	17

Table 1: Comparison of the number of nodes in the strategy trees, and equivalent strategy graph, for the *CHILD* influence diagram. The column *All* shows the number of nodes in the strategy tree before any compression; the column *Reach.* shows the number of reachable nodes under an optimal strategy; the column *Compr.* shows the number of nodes in a fully-compressed strategy tree (with many irrelevant nodes removed based on context-specific independence); and the last column shows the number of nodes in the strategy graph.

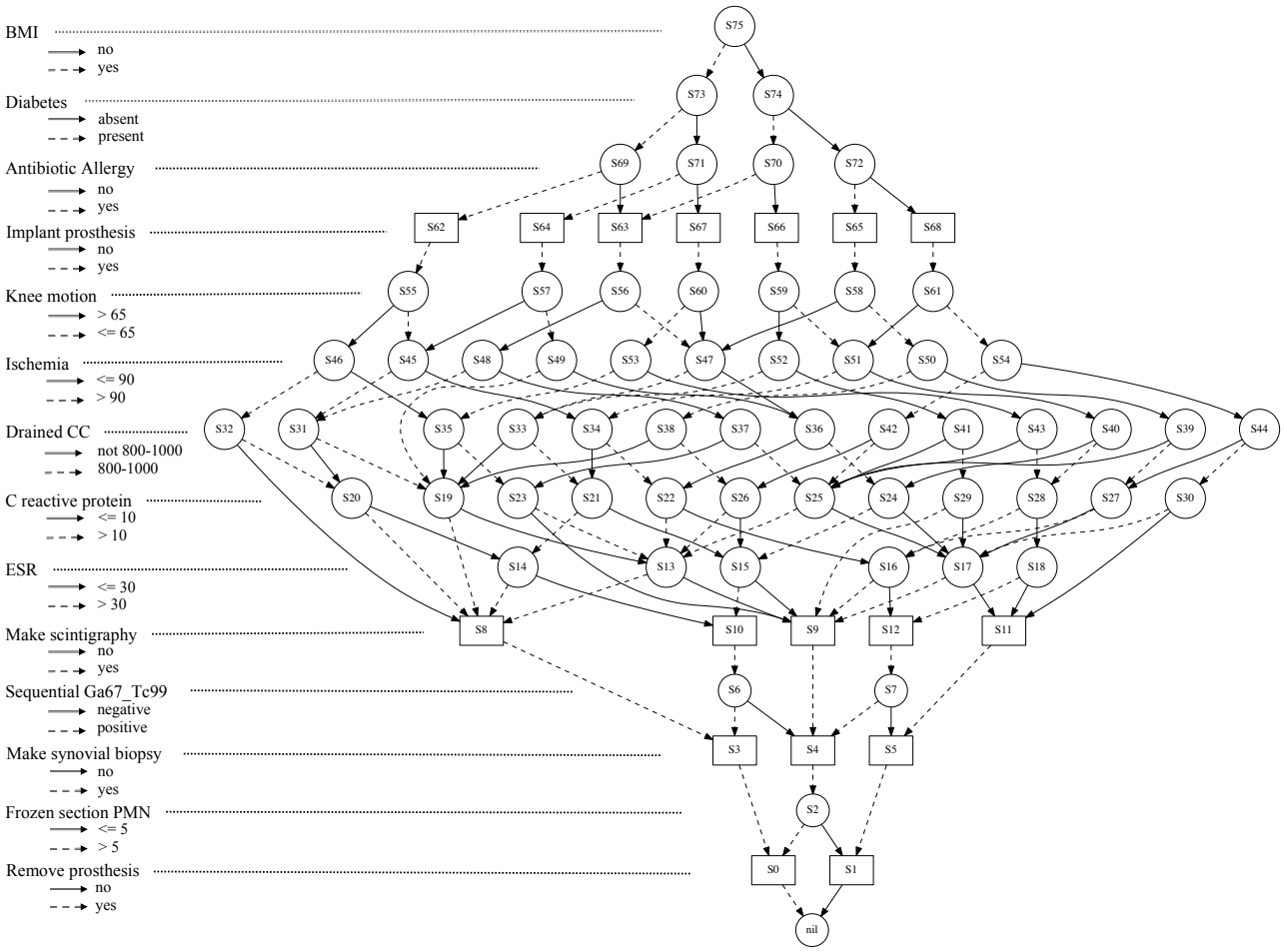

Figure 6: Bounded-suboptimal strategy graph for Arthronet influence diagram.

Child diagnosis/treatment problem We next consider an ID that is based on a Bayesian network for diagnosing congenital heart disease in a newborn baby with asphyxia (Spiegelhalter and Cowell 1992). This Bayesian network, named the CHILD network, is well-known in the graphical models community, especially as a test case for learning Bayesian networks. As in the original network, we assume the chance variable for birth asphyxia is always set to true. We transform this Bayesian network into an ID by adding a decision variable that represents a choice of two possible treatments, as well as a utility node that is a function of both the underlying disease and the treatment. The ID, with utility table, is shown in Figure 4. The decision and utility nodes we added are not intended to be medically realistic; they were added simply to create a useful test problem.

We solved this ID using the variable elimination algorithm, modified to construct a strategy graph. Table 1 compares the sizes of three different strategy trees, and the equivalent strategy graph. First shown is the original strategy tree before any compression. The "reachable" strategy tree includes only nodes and edges that are reachable by following optimal actions and positive-probability observations (edges). (For this ID, there are no zero-probability observations.) The fully-compressed strategy tree is the result of merging outgoing edges from a node that have the same successor node (rule 1b), and removing observations that are conditionally irrelevant (rule 1c). This compressed strategy tree is the same strategy tree that would be found by the algorithm of Luque, Arias, and Diez (2017). For this ID, compression based on conditional irrelevance dramatically reduces the size of the strategy tree.

The last column of Table 1 shows the size of the strategy graph after isomorphic subgraphs are merged. The results show how more compression is achieved by transforming a strategy tree into an equivalent strategy graph. The optimal strategy graph is shown in Figure 5, and has 17 nodes and 27 edges. Note that Table 1 shows the number of nodes in each level of the tree/graph, as well as the total number of nodes.

Arthronet problem The last ID we consider represents a realistic model of medical decision making for total knee arthroplasty (León 2011). Named Arthronet, it has eleven chance variables (ten of them are observable), four decision nodes, and four utility nodes. The ID and its parameters are available in the software package OpenMarkov.[1]

[1] www.probmodelxml.org/networks/

Variable	Strategy tree			Strat. graph
	All	Reach.	Comp.	
BMI	2	2	2	1
Diabetes	4	4	4	2
Antibiotic allergy	8	8	8	4
Implant prosthesis	16	8	8	8
Knee motion	48	16	16	8
Ischemia	144	32	32	16
Drained CC	288	64	64	32
C reactive protein	576	128	115	37
ESR	1,152	256	146	19
Make scintigraphy	2,304	256	146	8
Seq. Ga67 Tc99	6,912	430	201	2
Synovlal blopsy	13,824	430	201	3
Froz. section PMN	41,472	514	402	1
Remove prosthesis	82,944	514	402	2
Total nodes	149,694	2,662	1,747	143

Table 2: Comparison of number of nodes in the strategy trees, and equivalent strategy graph, for the Arthronet influence diagram. The column *All* shows the number of nodes in the strategy tree before any compression; the column *Reach.* shows the number of reachable nodes under an optimal strategy, with zero-probability branches also removed; the column *Comp.* shows the number of nodes in a fully-compressed strategy tree; and the last column shows the number of nodes in the strategy graph.

One reason for using this ID as a test problem is that Luque, Arias, and Diez (2017) report that the strategy tree for this ID found by their algorithm has too many nodes to be easily understood by a user; our results show that even a compressed strategy tree has 1,747 nodes. By contrast, our algorithm finds a strategy graph that is more than an order of magnitude smaller, with only 143 nodes. These compression results are shown in Table 2.

In fact, the strategy graph can be compressed even further using bounded-error approximation. Figure 6 shows the strategy graph that results from using a suboptimality bound of 0.2 when choosing an action for a decision node; in this approach, the bounded-suboptimal action is chosen that leads to the most compression. The resulting strategy graph has only 76 nodes, and yet its utility is 2.05655, which is very close to the optimal utility of 2.05668.

Conclusion

We have introduced an approach to strategy representation for influence diagrams that compresses a strategy tree into a simpler and easier-to-understand strategy graph. The motivation for this approach is in keeping with the original motivation for influence diagrams, which is to facilitate understanding and communication with users. We have also shown that a strategy graph can be further compressed in exchange for bounded-error approximation, making it possible to perform a sensitivity analysis that tests which parts of a strategy can be omitted without significantly affecting performance, as well as simplifying the strategy further.

References

Bollig, B., and Wegener, I. 1996. Improving the variable ordering of OBDDs is NP-complete. *IEEE Transactions on Computers* 45(9):993–1002.

Boutilier, C.; Friedman, N.; Goldszmidt, M.; and Koller, D. 1996. Context-specific independence in Bayesian networks. In *Proc. of the 12th Conference on Uncertainty in Artificial Intelligence*, 115–123.

Cabanas, R.; Antonucci, A.; Cano, A.; and Gomez-Olmedo, M. 2017. Evaluating interval-valued influence diagrams. *International Journal of Approximate Reasoning* 80:393 – 411.

Dechter, R. 2000. A new perspective on algorithms for optimizing policies under uncertainty. In *Proc. of the 5th International Conf. on AI Planning Systems (AIPS-2000)*, 72–81.

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

Howard, R., and Matheson, J. 1981. Influence diagrams. In Howard, R., and Matheson., J., eds., *The Principles and Applications of Decision Analysis*, 719–762.

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

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

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

Koller, D., and 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.; and Diez, F. 2017. Synthesis of strategies in influence diagrams. In *Proc. of the 33rd Conference on Uncertainty in Artificial Intelligence (UAI-17)*.

Luque, M.; Dez, F.; and 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).

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

Segal, I., and 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.

Spiegelhalter, D. J., and Cowell, R. G. 1992. Learning in probabilistic expert systems. In Bernardo, J.; Berger, J.; Dawid, A.; and Smith, A., eds., *Bayesian Statistics 4*. Oxford: Clarendon Press. 447–466.