Solving Influence Diagrams Using Heuristic Search

Changhe Yuan & Xiaojian Wu
Department of Computer Science and Engineering
Mississippi State University
Mississippi State, MS 39762
cyuan@cse.msstate.edu & xw83@msstate.edu

Abstract

Existing methods for solving influence diagrams are mostly based on the bottom-up dynamic programming technique. These methods may waste computation in solving decision scenarios that have zero probabilities or are unreachable from any initial state by following an optimal decision policy. Heuristic search was applied in (Qi & Poole 1995) to address these limitations, but their algorithm uses a trivial *infinity* upper bound and fails to fully utilize the potential of heuristic search. This paper develops an improved heuristic search algorithm for solving influence diagrams based on a more informative upper bound computed by relaxing the models. The algorithm is shown to be able to significantly improve the efficiency and scalability of existing methods for solving influence diagrams.

1 Introduction

An influence diagram (Howard & Matheson 1981) provides a compact and intuitive representation of the relations between random variables, decisions, and preferences in a domain and offers a popular framework for decision making under uncertainty. Numerous algorithms have been developed to solve influence diagrams (Bhattacharjya & Shachter 2007; Jensen, Jensen, & Dittmer 1994; Shenoy 1992; Zhang 1998; Qi & Poole 1995; Cooper 1988; Shachter 1986; Olmsted 1983). Most of these algorithms, whether they build a secondary structure or not, are based on the bottom-up dynamic programming approach. They start by solving small low-level decision problems and gradually build on the results to solve larger problems until the solution to the global-level decision problem is found. The drawback of these methods is that they may waste computation in solving decision scenarios that have zero probabilities or are unreachable from any initial state by following an optimal decision policy. Qi and Poole propose to address this problem using heuristic search (Qi & Poole 1995), but their method uses the trivial *infinity* upper bound to guide the search. A more informative upper bound needs to be developed to fully utilize the potential of heuristic search.

This paper develops an improved heuristic search algorithm for solving influence diagrams. We first note that, for an influence diagram of a partially observable domain, we can build an upper-bound influence diagram that not only is simpler but also provides an upper bound for the maximum expected utility of the original influence diagram (Nilsson & Hohle 2001). We then use a strong jointree of the upper-bound influence diagram to compute the probabilities and upper-bound utilities needed in an AND/OR graph search algorithm to find an optimal policy graph for the original influence diagram. Our empirical results show that the new algorithm can significantly improve the efficiency of the state-of-the-art solution methods for influence diagrams on large decision problems.

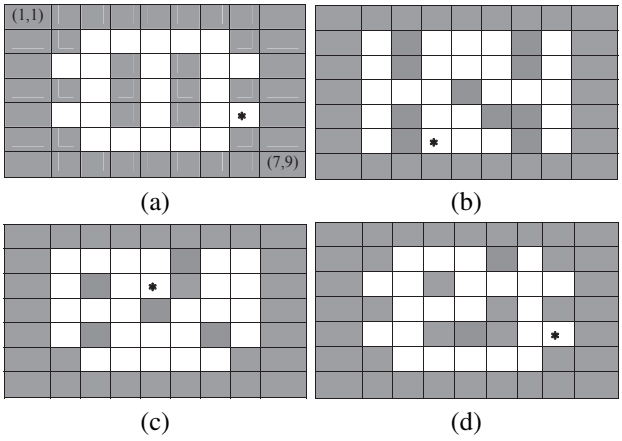

Figure 1: Four maze domains.

2 Maze Exploration Problem

Figure 1 shows four instances of the maze exploration problem (Horsch & Poole 1998; Littman, Cassandra, & Kaelbling 1995). The shaded tiles stand for walls, and the white tiles walkable space. The tiles with a star are the goal states. An agent is randomly placed at a non-goal state and tries to reach the goal. The agent has four sensors, one in each direction, which can sense whether there is wall in the neighboring tile of that direction. However, the sensors are noisy. It can detect wall correctly with probability 0.9, and it may give false alarm with probability 0.05. Based on the results of the sensors, the agent decides whether to move to a neighboring tile. The action is also noisy. The agent may fail to

move with probability 0.089, move to the next tile successfully with probability 0.89, move sideways with probability 0.02 (0.01 each side way), and move backward with probability 0.001. But all directions with walls will be assigned 0 probabilities. Remaining probabilities are normalized.

Since the actions are noisy, we may never reach the goal. The number of actions, also called the number of horizons, cannot be determined in advance. A two-horizon decision problem of the maze problem can be modeled using an influence diagram as shown in Figure 2(a). We define influence diagram in the next section.

3 Influence Diagram

An influence diagram is a directed acyclic graph G containing variables $\mathbf{V}$ of a decision domain. The variables in the diagram can be classified into three groups, $\mathbf{V} = \mathbf{X} \cup \mathbf{D} \cup \mathbf{U}$, where $\mathbf{X}$ are the oval-shaped *chance* variables specifying the uncertain decision environment, $\mathbf{D}$ are the square-shaped *decision* variables specifying the possible decisions to be made in the domain, and $\mathbf{U}$ are the diamond-shaped *utility* variables representing a decision maker's preferences. Similar to Bayesian networks, each variable $X_i \in \mathbf{X}$ is associated with a conditional probability distribution $P(X_i|Pa(X_i))$, where $Pa(X_i)$ is the set of parents of X_i in G. Each decision variable $D_j \in \mathbf{D}$ has multiple states, each state corresponding to a possible action. Incoming arcs into a decision variable are called *information arcs*, which require the chance variables originating these arcs to be observed before the decision is made. These variables are called the *information variables* of the decision. *No-forgetting* is typically assumed for an influence diagram, i.e., the information variables of earlier decisions are also information variables of later decisions, even though no explicit information arcs exist between them. In this paper we work with *limited-information influence diagrams* (LIMIDs) (Lauritzen & Nilsson 2001) with *explicit* information arcs. Finally, each utility node $U_i \in \mathbf{U}$ represents a function that maps each configuration of its parents to a utility value standing for the preference of the decision maker. Utility variables typically do not have other variables as children except multi-attribute utility/super-value variables.

In the influence diagram in Figure 2(a), x_i and y_i stand for the location coordinates of the agent at time i. $\{ns_i, es_i, ss_i, ws_i\}$ are the sensor readings in four directions for the same time point. d_i is the decision to be made by the agent as to which neighboring tile to move to based on all the previous sensor readings and actions. The utility variable u assigns a reward depending on whether or not the agent reaches the goal state after the last action is taken.

The decision variables in an influence diagram are typically assumed to be temporally ordered, i.e., the decisions have to be made in a particular order. Suppose there are n decision variables $D_1, D_2, ..., D_n$ in an influence diagram. The decision variables partition the variables in $\mathbf{X}$ into a collection of disjoint sets $\mathbf{I_0}, \mathbf{I_1}, ..., \mathbf{I_n}$. For each $k (0 < k < n)$, $\mathbf{I_k}$ is the set of chance variables that must be observed between D_k and D_{k+1}. $\mathbf{I_0}$ is the set of initial evidence variables that must be observed before D_1. $\mathbf{I_n}$ is the set of variables left unobserved when decision D_n is made. Therefore,

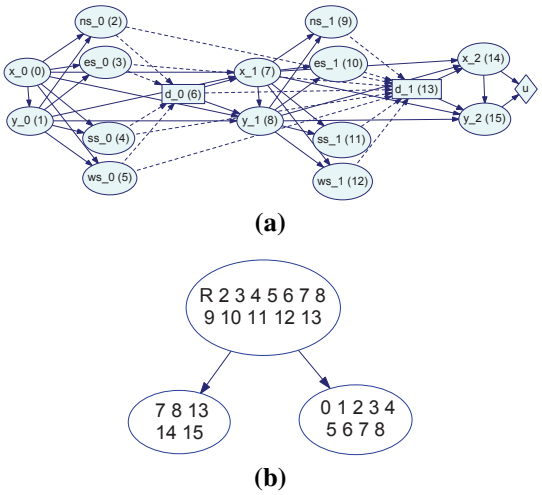

Figure 2: (a) An example influence diagram and (b) its strong jointree. The numbers in both figures stand for the indices of the variables. The node with "R" is the strong root.

a partial order $\prec$ is defined on the influence diagram over $\mathbf{X} \cup \mathbf{D}$ as follows:

$$\mathbf{I_0} \prec \mathbf{D_1} \prec \mathbf{I_1} \prec ... \prec \mathbf{D_n} \prec \mathbf{I_n}. \quad (1)$$

A solution to the decision problem defined by an influence diagram is a series of decision rules for the decision variables. A *decision rule* for D_k is a mapping from each configuration of its parents to one of the actions defined by the decision variable. A *decision policy* is a series of decision rules with one decision rule for each decision variable. The goal of solving an influence diagram is to find an *optimal* decision policy that maximizes the expected utility. The maximum expected utility is equal to

$$\sum_{\mathbf{I_0}} \max_{D_1} ... \sum_{\mathbf{I_{n-1}}} \max_{D_n} \sum_{\mathbf{I_n}} \prod_i P(X_i|Pa(X_i)) \sum_j U_j(Pa(U_j)).$$

In general, the summations and maximizations are not commutable. The methods presented in Section 4 differ in the various techniques they use to carry out the summations and maximizations in the order.

One note here is that recent research has begun to relax the assumption of ordered decisions. In particular, Jensen proposes a framework called unconstrained influence diagram which allows a partial ordering among the decisions (Jensen 2002). This framework is out of the scope of our current research.

4 Existing Methods for Solving Influence Diagrams

Many approaches have been developed for solving influence diagrams. The simplest approach is to unfold an influence diagram into a decision tree and solve it using dynamic programming (Howard & Matheson 1981). Another approach

called arc reversal solves an influence diagram directly using techniques such as arc-reversal and node-removal (Olmsted 1983; Shachter 1986). Once a decision variable is removed, we obtain the optimal decision rule for the decision. Several methods reduce influence diagrams into Bayesian networks by converting decision nodes into random variables such that the solutions of certain inference problems in the Bayesian networks correspond to the optimal decision policy of the influence diagrams (Cooper 1988; Zhang 1998). Another method (Shenoy 1992) transforms an influence diagram into a valuation network and applies variable elimination to solve the valuation network. A recent work compiles influence diagrams into decision circuits and use the decision circuits to compute the optimal policies (Bhattacharjya & Shachter 2007). Their method can take advantage of local structures present in an influence diagram, such as deterministic relations.

Two other algorithms are more closely related to this research and are discussed in the following.

4.1 The jointree algorithm

Another method solves an influence diagram using a strong jointree (Jensen, Jensen, & Dittmer 1994). A jointree is *strong* if it has at least one clique R, called *strong root*, such that for any pair of adjacent cliques C_1 and C_2 with C_1 closer to R than C_2, the variables in separator $S = C_1 \cap C_2$ must appear earlier in the partial order defined in Eqn 1 than $C_2 \setminus C_1$. A strong jointree for the influence diagram in Figure 2(a) is shown in Figure 2(b).

An influence diagram can be solved exactly by message passing on the strong jointree. Each clique C on the jointree contains two potentials, a probability potential ϕ_C and a utility potential ψ_C. For clique C_2 to send a message to clique C_1, ϕ_{C_1} and ψ_{C_1} should be updated as follows (Jensen, Jensen, & Dittmer 1994):

$$\phi'_{C_1} = \phi_{C_1} \times \psi_S; \quad \psi'_{C_1} = \psi_{C_1} + \frac{\psi_S}{\phi_S};$$

where

$$\phi_S = \sum_{C_2 \setminus S} \phi_{C_2}; \quad \psi_S = \max_{C_2 \setminus S} \phi_{C_2} \times \psi_{C_2}.$$

Contrary to the jointree algorithm for Bayesian networks (Lauritzen & Spiegelhalter 1988), only the collection phase of the strong jointree is needed to solve an influence diagram. After the collection phase, we can obtain the maximum expected utility by carrying out the remaining summations and maximizations in the root. We can further extract the optimal decision rules for the decision variables from some of the cliques that contain these variables.

Building a jointree for a Bayesian network may fail if the jointree is too large to fit in memory. This is also true for influence diagrams. The memory requirement of a strong jointree for an influence diagram is even higher because of the constrained order in Eqn 1. Consequently, the jointree algorithm is typically infeasible for solving large influence diagrams.

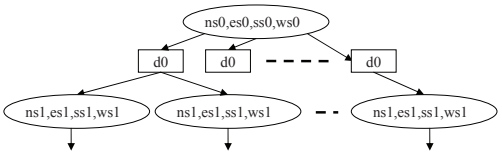

Figure 3: (a) An AND/OR graph.

4.2 AND/OR Search Graph

If a decision maker strictly follows an optimal decision policy, s/he only needs to consider some of the decision scenarios modeled in an influence diagram. The other decision scenarios will never occur because they either have zero probabilities or are unreachable. A dynamic programming-based method ignores this fact and solves all decision scenarios. Much computation is wasted.

To address this limitation, Qi and Poole propose to compile an influence diagram into an equivalent AND/OR graph (or decision graph) and apply heuristic search to find an optimal decision policy (Qi & Poole 1995). An *AND/OR graph* is a directed acyclic graph whose nodes can be classified into two types: *AND* nodes and *OR* nodes. AND nodes correspond to chance variables, and OR nodes correspond to decision variables. Each AND/OR graph has exactly one root. A utility value is associated with each arc originating from an OR node. A probability is associated with each arc originating from an AND node. The probabilities of all the arcs from an AND node sum to 1.0. Leaf nodes are terminals, each with a utility value. Figure 3 shows a partial AND/OR graph for the influence diagram in Figure 2(a). Oval-shaped nodes are AND nodes, and square-shaped nodes are OR nodes.

We then define *valuation functions* for each node in an AND/OR graph as follows: (a) If a terminal node, the value of its valuation function is equal to its utility value; (b) if an AND node, the value of its valuation function is the sum of the values of its children nodes weighted by the probabilities of the outgoing arcs; (c) if an OR node, the value of its valuation function is the maximum of the summed utility values of each child node and corresponding arc.

A *policy graph* of an AND/OR graph is a subgraph such that: (a) it consists of the root of the AND/OR graph; (b) if a non-terminal AND node is in the policy graph, all its children is in the policy graph; and (c) if a non-terminal OR node is in the policy graph, exactly one of its children is in the policy graph. The solution to an AND/OR graph is the policy graph with the maximum expected utility at the root. Heuristic search can be applied to search for an optimal policy graph of the AND/OR graph. In order for heuristic search to be effective, it is necessary to have informative upper bounds that measure how good a partial policy graph is and choose to expand the most promising search branches first. However, Qi and Poole's method (Qi & Poole 1995) uses the trivial *infinity* upper bound during the search and effectively reduces to an uninformed search algorithm.

5 An Improved Heuristic Search for Solving Influence Diagrams

In this section we present an improved heuristic search algorithm that relies on an informative upper bound in solving an influence diagram. We first discuss how to construct a relaxed upper-bound model for an influence diagram of partially observable domains. We then describe how to use a strong jointree of the upper-bound model to efficiently compute the upper bounds and probabilities needed by a heuristic search algorithm for solving the influence diagram.

5.1 Upper-bound influence diagrams

Intuitively, the more information, the better decisions can be made. This is the basic idea behind our proposed method. We let the decision variables of an influence diagram obtain extra information. Since any decision policy of the original influence diagram is part of a policy of the augmented influence diagram, we can easily prove that the resulting model has a higher or equal maximum expected utility.

Adding extra information variables only makes sense if the augmented influence diagram can be further made simpler than the original model. The key observation is that certain variables make some or all of the existing information variables non-requisite. An information variable I_i is said to be *non-requisite* (Nielsen & Jensen 1999; Lauritzen & Nilsson 2001) for a decision node D if

$$I_i \perp (\mathbf{U} \cap de(D))|D \cup (Pa(D) \setminus \{I_i\}). \quad (2)$$

where $de(D)$ are the descendants of D. A *reduction* of an influence diagram is obtained by deleting all the non-requisite information arcs (Nilsson & Hohle 2001). We have the following theorem.

Theorem 1 *A reduced influence diagram has the same maximum expected utility as the original influence diagram.*

Proof: We can prove the theorem by contradiction. If the reduced influence diagram has a decision policy with higher maximum expected utility, we can augment the decision policy by prescribing the same optimal decision for any states of the non-requisite variables, because these variables are conditionally independent from the utility variables given the decision and other information variables. Consequently the augmented decision policy has the same maximum expected utility, leading to contradiction. Similarly, if the original influence diagram has a higher maximum expected utility, we can project its optimal decision policy to the reduced influence diagram by ignoring the non-requisite variables, which also results in contradiction. □

Although reduction does not affect the expected utility, it does affect the constrained order in Equation 1, because the information variables are not restricted to appear before their decision anymore, which may result in an order with smaller constrained treewidth (Dechter & Rish 2003).

To select the extra information variables with such property, we apply a method proposed in (Nilsson & Hohle 2001). Let $nd(X)$ be the non-descendant variables of variable X, $fa(X)$ be $Pa(X) \cup \{X\}$, $fa(\mathbf{X})$ be $\cup_{X_i \in \mathbf{X}} fa(X_i)$, and $\triangle_j$ be $\{D_1, ...D_j\}$. We have the following theorem (Nilsson & Hohle 2001).

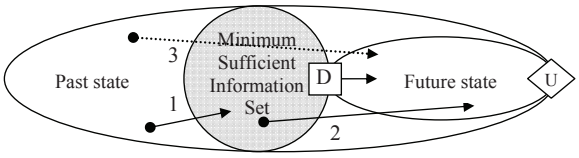

Figure 4: Relations between past and future information states and the minimum sufficient information set.

Theorem 2 *For an influence diagram with the constrained order in Equation 1, if we add to each decision variable D_j the following additional information variables in the order of $j = n, ..., 1$,*

$$N_j = \arg\min_{B \subseteq (B_j \cap nd(D_j))} \{|B||fa(\triangle_j) \perp (\mathbf{U} \cap de(D_j))|(B \cup \{D_j\})\}, \quad (3)$$

where

$$B_j = \begin{cases} \mathbf{U} \cup \mathbf{D} & j=n, \\ \cap_{i=j+1}^{k} \{n \in V | n \perp (\mathbf{U} \cap de(d_j))|fa(d_i)\} & j<n, \end{cases}$$

the following holds for any D_j in the resulting influence diagram:

$$(de(D_j) \cap \mathbf{U}) \perp fa(\triangle_{i-1})|fa(D_j). \quad (4)$$

Readers are referred to (Nilsson & Hohle 2001) for the proof of the theorem. We name N_j as the *minimum sufficient information set* (MSIS) of D_j. The intuition behind the method can be illustrated graphically using Figure 4. The shaded oval shows the MSIS set N_D of decision D. The past state affects the variables in $N_D \cup \{D\}$, illustrated by the arc labeled 1, and $N_D \cup \{D\}$ affects the future state, as illustrated by arc 2. The future state further determines the values of the utility variables. $N_D \cup \{D\}$ d-separates the past and future states and prevents the direct influence shown by arc 3. Essentially, decision D tries to obtain the most informative information about the current state in order to choose a best action to affect the future state so as to maximize the expected values of its descendant utility variables. The minimum sufficient information set is related to *extremality* defined in (Zhang 1998) and *blocking* defined in (Nielsen & Jensen 1999).

To construct an upper-bound influence diagram, we find the MSIS set for each decision in the order of $D_n, ..., D_1$ and add them as information variables to the corresponding decisions. We then delete the non-requisite information arcs. The resulting influence diagram guarantees to provide an upper bound on the maximum expected utility of the original influence diagram.

Note that the information set N_j may contain the existing information variables $Pa(D_j)$. In that case D_j simply obtain more information. More often than not, however, the information set makes some or all variables in $Pa(D_j)$ non-requisite. This is especially true for *partially observable* domains. In such a decision problem, we cannot directly observe the state of a system. We can only rely on indirect information, such as sensor readings, to infer the state.

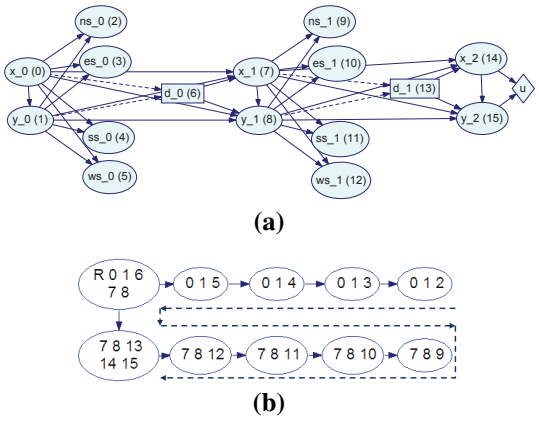

Figure 5: (a) the upper-bound influence diagram for the diagram in Figure 2(a), and (b) its strong jointree.

For such domains, the MSIS set typically contains the state variables, which means direct knowledge about the system state. The direct knowledge makes indirect sensor readings non-requisite and typically results in a simpler model. This is an especially effective way to deal with the complexity brought by the no-forgetting assumption.

We use the influence diagram in Figure 2(a) as an example. The information set for d_1 is originally $\{ns_0, es_0, ss_0, ws_0, d_0, ns_1, es_1, ss_1, ws_1\}$. We find that its minimum sufficient information (MSIS) set is $\{x_1, y_1\}$. We also find that the MSIS set for d_0 is $\{x_0, y_0\}$. By adding $\{x_1, y_1\}$ and $\{x_0, y_0\}$ as information variables to d_1 and d_0 respectively and reducing the influence diagram, we obtain the much simpler influence diagram in Figure 5(a). The strong jointree for the new influence diagram is shown in Figure 5(b), which is also much smaller than the strong jointree for the original model. Essentially, the upper-bound influence diagram assumes the actual location is directly observable to the agent, which renders all the previous sensor readings nonrelevant any more. This effectively transforms a partially observable decision problem into a fully observable one. The resulting influence diagram and, hence, its jointree is much easier to solve. Arguably, most of the real-world decision problems are partially observable and can be tackled using this method.

Another way to look at the strong jointree of the upper-bound influence diagram is that it allows commuting the maximizations and summations when computing the maximum expected utility. In other words, *we choose summations or maximizations for the variables according to the original influence diagram but carry them out in an order that is consistent with the upper-bound influence diagram*. Similar to MAP problems (Park & Darwiche 2003), the commutation generates an upper bound for the maximum expected utility for the original influence diagram.

It is also shown that finding the minimum sufficient information (MSIS) set for a decision variable in an influence diagram is equivalent to finding a minimum separating set in the moralized graph of the influence diagram (Acid & de Campos 1996; Nilsson & Hohle 2001). In particular, Acid and de Campos (Acid & de Campos 1996) propose an algorithm based on the *Max-flow Min-cut* algorithm (Ford & Fulkerson 1956) for finding a minimum separating set between two sets of nodes in a Bayesian network with some of the separating variables being fixed. We can apply the algorithm to find the MSIS sets. The two sets of nodes are $fa(\triangle_j)$ and $\mathbf{U} \cap de(D_j)$. The only fixed separating variable is D_j. The algorithm first introduces two dummy variables, *source* and *sink*, to the moralized graph. The source is connected to the neighboring variables of $fa(\triangle_j)$, and the sink to the neighboring variables of $\mathbf{U} \cap de(D_j)$. We then create a max-flow network out of the undirected graph by assigning each edge capacity 1.0, solving which produces a minimum separating set between the sink and source that contains D_j.

It is possible that multiple minimum separating sets exist. The algorithm in (Acid & de Campos 1996) finds the one that is closest to the second variable set. It is also a sensible choice for our problem because *more recent information is typically more informative* (Jensen 2002).

5.2 Efficient AND/OR graph search for solving influence diagrams

The relaxed influence diagram of last section can be used to compute an upper bound on the maximum expected utility of the original influence diagram (Nilsson & Hohle 2001). We show in this section how to use the relaxed influence diagram to compute upper bounds for the maximum expected utilities of all the decision scenarios encountered in searching for an optimal decision policy. The main idea is to build a strong jointree for the upper-bound influence diagram and use efficient incremental jointree evaluation to compute these bounds. Similar techniques have been used in computing upper bounds for finding MAP solutions in Bayesian networks (Yuan & Hansen 2009).

Constructing AND/OR graph Since the goal is to find an optimal decision policy for the original influence diagram, we have to construct an AND/OR graph that is consistent with the original constrained order. For the influence diagram in Figure 2(a), the partial order is

$$\{ns_0, es_0, ss_0, ws_0\} \prec \{d_0\} \prec \{ns_1, es_1, ss_1, ws_1\} \prec \{d_1\} \prec \{x_0, y_0, x_1, y_1, x_2, y_2, u\}. \quad (5)$$

Its upper-bound influence diagram and jointree are shown in Figure 5. The AND/OR graph used in (Qi & Poole 1995) has alternating AND and OR layers, as shown in Figure 3. Each AND node corresponds to the information set of a decision. Each OR node corresponds to a decision. However, the joint probability distributions of all the information sets that are needed in order to construct such an AND/OR graph are often not readily available.

Instead, we construct an AND/OR graph based on the structure of the strong jointree in Figure 5(b). We first need to expand the variables in $\{ns_0, es_0, ss_0, ws_0\}$ according to the order in Equation 5. The jointree does not have a clique that contains all four variables. Actually they are all in different cliques. So we expand the variables one by one. That means the first four layers of our AND/OR graph are

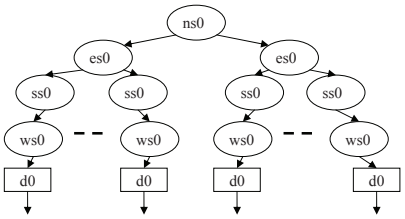

Figure 6: AND/OR graph used in our approach.

all AND layers. *An AND/OR graph does not necessarily have to alternate between AND and OR layers*. Figure 6 shows part of our AND/OR graph. Although our AND/OR graph has more layers, it can naturally utilize the probabilities and utilities computed by the jointree. If we start by expanding ns_0, we need the probabilities of $P(ns_0)$ to label the outgoing arcs. We can readily look up the probabilities from clique $(0, 1, 2)$ after an initial full jointree evaluation. Note that we use the jointree of the relaxed influence diagram to compute the probabilities. We can do that because these probabilities are the same as those computed from the original influence diagram. This is due to the fact that the same set of actions will reduce both models into the same Bayesian networks. Relaxing an influence diagram only changes the expected utilities of the decision variables.

After expanding ns_0, we expand any of $\{es_0, ss_0, ws_0\}$. Suppose the next variable is es_0, we need the conditional probabilities of es_0 given ns_0. These probabilities can be computed by setting the state of ns_0 as new evidence to the jointree and evaluate the jointree again. The same process is used to expand $\{ss_0, ws_0\}$.

Note that we do not have to expand one variable at a time. If a clique has multiple variables in the same information set, the variables can be expanded *together* because a joint probability distribution over them can be easily computed. Expanding them together also saves the need to do marginalization. For example, variables x_1, y_1, x_2, y_2 (7,8,14,15) are in the same information group and also reside in a same clique. We can expand them as a single layer in the AND/OR graph.

After $\{ns_0, es_0, ss_0, ws_0\}$ are expanded, we expand d_0 as an OR layer. This is where the upper bounds are needed. We set the states of $\{ns_0, es_0, ss_0, ws_0\}$ as evidence to the jointree and compute the expected utility values for d_0 by reevaluating the jointree. The expected utilities of d_0 are upper bounds for the same decision scenarios of the original model. We can use the upper bounds to select the most promising decision alternative to search first. The exact value will be returned once the subgraph is searched. If the value is higher than the upper bounds of the remaining decision alternatives, these alternatives are immediately pruned because they cannot be part of an optimal decision policy. We repeat the above process until a complete policy graph is found.

Incremental jointree evaluation It is clear that repeated jointree evaluation has to be performed in computing the upper bounds and conditional probabilities. A naive approach is at each time to set the states of instantiated variables as evidence and perform a *full* jointree evaluation. However, that is too costly. We can use an efficient *incremental* jointree evaluation method to compute the probabilities and upper-bound utilities.

The key idea is that we can choose a particular order of the variables that satisfies the constraints of Eqn 5 such that an *incremental jointree evaluation scheme* can be used to compute these information. Given such an order, we know exactly which variables have been searched and which variable will be searched next at each search step. We only need to send messages from the parts of the jointree that contain the already searched variables to a clique with the next search variable. For example, after we search es_0, the only message needs to be sent to obtain $P(ns_0|es_0)$ is the message from clique $(0, 1, 3)$ to $(0, 1, 2)$. There is no need to evaluate the whole jointree. If we choose the following search order for the maze problem

$$ns_0, es_0, ss_0, ws_0, d_0, ns_1, es_1, ss_1, ws_1, d_1, x_0, y_0,$$
$$x_1, y_1, x_2, y_2,$$

we can use an incremental message passing in the direction of the dashed arc in Figure 5(b) to compute the probabilities and upper bounds needed in one downward pass of a depth-first search.

Both message collection and distribution are needed in this new scheme, unlike evaluating a strong jointree for the original influence diagram. The messages being propagated contain two parts: utility potentials and probability potentials. The distribution phase is typically needed to compute the conditional probabilities. For example, suppose we decide to expand es_0 before ns_0, we have to send message from clique $(0, 1, 3)$ to $(0, 1, 2)$ to obtain $P(ns_0|es_0)$. We only need to send the probability potential in message distribution. We do not need to send utility potentials because past payoffs do not count towards the expected utilities of future decisions.

Efficient backtracking in DFBnB We use *depth-first branch-and-bound* (DFBnB) to utilize the efficient incremental bound computation. Depth-first search makes sure that the search need not jump to a different search branch before backtracking is needed. In other words, the jointree only needs to work with one search history at a time.

We do need to backtrack to a previous search node once we finish a search branch or realize that a search branch is not promising and should be pruned. We need to retract all the newly set evidence since the generation of that search node and restore the jointree to a previous state. One way to achieve this is to reinitialize the jointree with correct evidence and perform a full jointree evaluation, which is too costly. We propose to cache the potentials and separators along the message propagation path before changing them by either setting evidence or overriding them with new messages. When backtracking, we simply restore the most recently cached potentials and separators in the reverse order. The jointree will be restored to the previous state with no additional computations. This backtracking method is much more efficient than reevaluating the whole jointree.

maze	#horizons	Jointree		DFBnB+policy graph		DFBnB+expected utility	
		time (ms)	memory (MB)	time (ms)	memory (MB)	time (ms)	memory (MB)
a	2	16	12.2	172	4.1	141	4.3
	3	500	318.0	5,438	16.5	5,438	6.8
	4	-	-	304,766	552.3	300,750	10.4
	5	-	-	-	-	20,736,937	15.2
b	2	16	12.2	171	4.7	156	4.9
	3	578	318.0	8,484	22.3	8,578	7.6
	4	-	-	512,188	894.6	510,125	11.5
	5	-	-	-	-	25,242,156	16.5
c	2	15	12.2	140	5.2	125	5.3
	3	500	318.0	6,375	23.5	6,422	8.5
	4	-	-	461,453	838.8	325,094	12.6
	5	-	-	-	-	15,275,281	17.9
d	2	15	12.2	141	5.7	141	6.0
	3	563	318.0	6,031	22.4	6,078	9.2
	4	-	-	423,875	894.6	412,938	13.7
	5	-	-	-	-	17,065,531	19.3

Table 2: Running time (milliseconds) and memory (MB) of three algorithms, the jointree algorithm, DFBnB+infinity bound, and DFBnB+jointree bound, in solving the maze problem with different number of horizons. "policy graph" means the search graph is stored. "expected utility" means search graph is not stored. "-" shows failure of the algorithm because memory requirement is greater than the amount of available memory (4GB).

number of horizons	size of last decision rule
2	4,096
3	262,144
4	16,777,216
5	1,073,741,824

Table 1: The size of the last decision rule for the maze domain with different numbers of horizons.

6 Empirical Evaluation

Our DFBnB algorithm for solving influence diagrams has two versions. If the goal is to find the optimal policy graph, the algorithm has to keep the search graph in the memory. Otherwise if the goal is only to compute the maximum expected utility, there is no need to store the search graph. The algorithm only needs to keep track of one search path during the search. This will significantly reduce the memory requirement of the search and improves its scalability. We denote these two search algorithms as "DFBnB+policy graph" and "DFBnB+expected utility" respectively.

We compared our algorithms against the jointree algorithm (Jensen, Jensen, & Dittmer 1994) on the four maze problems in Figure 1. The jointree algorithm is not affected by the type of tasks because a strong jointree has to be built in both cases. Experiments were performed on a 3.2 GHz processor with 4 gigabytes of RAM running a 64-bit version of Windows XP.

First, because of the no-forgetting assumption, the last decision of an influence diagram has all the previous decisions and their information variables as its own information variables and, hence, has the largest decision rule. As the number of horizons increases, the size of the last decision rule grows exponentially. Table 1 shows the sizes for the maze influence diagrams with different number of horizons. Clearly the difficulty of solving these influence diagrams increases rather quickly.

We tested the algorithms on the maze influence diagrams with up to 5 horizons. Table 2 lists the running time and the amount of memory used by the algorithms. For small-horizon influence diagrams, the jointree algorithm is most efficient. This is because the strong jointrees for these influence diagrams are rather small and can be built successfully. Once the jointrees are built, only one collection phase is necessary in solving the influence diagram. For the DFBnB algorithms, although their jointrees are even smaller, they have to perform repeated message propagation to compute upper bounds and probabilities during the search.

However, the jointree algorithm failed in solving the maze models with more than 3 horizons, because the strong jointrees are too large to fit in memory. The "DFBnB+policy graph" algorithm could solve the maze models with up to 4 horizons. But it failed on the 5-horizon models because the policy graphs became too large. The "DFBnB+expected utility" algorithm could solve the 5-horizon models. In fact, the amount of memory needed by this algorithm is proportional to the depth of the AND/OR search graph and only grows linearly. Therefore, this algorithm can theoretically solve the maze models with a large number of horizons without running out of memory. However, the amount of running time needed may quickly become too prohibitive.

We note that the statistics of the jointree algorithm across the different maze problems are rather similar. This is because the influence diagrams for the maze problems share the same structures and differ only in the parameters. The jointree algorithm is not affected much by the parameters.

However, the parameters determine the asymmetry of a search problem, which in turn affects the search efficiency. The results show that the DFBnB algorithms have rather different performance on the different maze problems. Another observation is that although whether to store the search graph or not affects the scalability of the DFBnB algorithms, it does not affect the efficiency of the search by that much.

7 Concluding Remarks

This paper develops an improved heuristic search algorithm for solving influence diagrams. We made two major contributions in the algorithm. First, we propose an efficient approach for using the jointree algorithm to compute upper bounds for AND/OR graph search. The main idea is to first generate an upper-bound influence diagram by allowing each decision variable to obtain extra information. A jointree of the upper-bound influence diagram is then used to compute upper bounds for a heuristic search. Second, we propose a new approach for constructing an AND/OR graph based on the structure of the strong jointree of the upper-bound influence diagram. Our results show that the new algorithms significantly improves the state-of-the-art jointree algorithm. The maze domain used in our experiments is a classical problem used in the research community of partially observable Markov decision processes (POMDPs). However, our proposed methods are applicable to more general influence diagrams.

Acknowledgements

This research was supported by the National Science Foundation grants IIS-0842480 and EPS-0903787. All experimental data have been obtained using SMILE, a Bayesian inference engine developed at the Decision Systems Laboratory at University of Pittsburgh and available at `http://genie.sis.pitt.edu`.

References

Acid, S., and de Campos, L. 1996. An algorithm for finding minimum d-separating sets in belief networks. In *Proceedings of the 12th Annual Conference on Uncertainty in Artificial Intelligence (UAI-96)*, 3–10. San Francisco, CA: Morgan Kaufmann.

Bhattacharjya, D., and Shachter, R. 2007. Evaluating influence diagrams with decision circuits. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*, 9–16. AUAI Press.

Cooper, G. 1988. A method for using belief networks as influence diagrams. In *Proceedings of the Fourth Conference on Uncertainty in Artificial Intelligence*, 55–63. Minneapolis, MN, USA: Elsevier Science, New York, NY.

Dechter, R., and Rish, I. 2003. Mini-buckets: A general scheme for approximating inference. *Journal of ACM* 50(2):1–61.

Ford, L. R., and Fulkerson, D. R. 1956. Maximal flow through a network. *Canadian Journal of Mathematics* 8:399–404.

Horsch, M. C., and Poole, D. 1998. An anytime algorithm for decision making under uncertainty. In *Proceedings of the 14th Annual Conference on Uncertainty in Artificial Intelligence (UAI-98)*.

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

Jensen, F.; Jensen, F. V.; and Dittmer, S. L. 1994. From influence diagrams to junction trees. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, 367–373. Morgan Kaufmann.

Jensen, F. V. 2002. Unconstrained influence diagrams. In *In Proc. of the 18th International Conference on Uncertainty in Artificial Intelligence (UAI-02)*, 234–241. Morgan Kaufmann Publishers.

Lauritzen, S. L., and Nilsson, D. 2001. Representing and solving decision problems with limited information. *Management Science* 47:1235–1251.

Lauritzen, S. L., and Spiegelhalter, D. J. 1988. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society, Series B (Methodological)* 50(2):157–224.

Littman, M. L.; Cassandra, A. R.; and Kaelbling, L. P. 1995. Learning policies for partially observable environments: Scaling up. In *Proceedings of the Twelfth International Conference on Machine Learning*, 362–370. Morgan Kaufmann.

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

Nilsson, D., and Hohle, M. 2001. Computing bounds on expected utilities for optimal policies based on limited information. Technical Report 94, Dina Research.

Olmsted, S. M. 1983. *On representing and solving decision problems*. Ph.D. Dissertation, Stanford University, Engineering-Economic Systems Department.

Park, J. D., and Darwiche, A. 2003. Solving MAP exactly using systematic search. In *Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence (UAI–03)*, 459–468.

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

Shachter, R. D. 1986. Evaluating influence diagrams. *Oper. Res.* 34(6):871–882.

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

Yuan, C., and Hansen, E. A. 2009. Efficient computation of jointree bounds for systematic MAP search. In *Proceedings of 21st International Joint Conference on Artificial Intelligence (IJCAI-09)*.

Zhang, N. L. 1998. Probabilistic inference in influence diagrams. In *Computational Intelligence*, 514–522.