Solving Limited-Memory Influence Diagrams Using Branch-and-Bound Search

Arindam Khaled, Changhe Yuan and **Eric A. Hansen**
Department of Computer Science and Engineering
Mississippi State University
Mississippi State, MS 39762
ak697@msstate.edu,{cyuan,hansen}@cse.msstate.edu

Abstract

A limited-memory influence diagram (LIMID) is a generalization of a traditional influence diagram in which the assumptions of regularity and no-forgetting are relaxed. Lauritzen and Nilsson (2001) introduced this model and proposed an iterative algorithm, called single policy updating (SPU), that converges to a locally optimal solution. In this paper, we describe a branch-and-bound algorithm that finds a globally optimal solution. Initial experiments suggest that the branch-and-bound approach scales better than SPU.

1 Introduction

An influence diagram is a graphical model that provides a framework for decision making in uncertain environments (Howard and Matheson 1981). In its traditional form, an influence diagram makes the assumptions of "regularity" and "no-forgetting," which means that decisions are ordered and each decision is conditioned on all previous observations and decisions. Lauritzen and Nilsson (2001) introduced a more general framework, called a *limited-memory influence diagram* (LIMID), that allows both the regularity and the no-forgetting assumptions to be relaxed. That is, LIMIDs do not require decisions to be ordered and they allow a decision to be conditioned on only a subset of previous decisions and observations. This generalization has several advantages. It allows modeling of (cooperative) multi-agent decision making in which one agent is not aware of the simultaneous or previous decisions of another agent. It allows modeling of situations in which only a subset of previous observations are relevant or available; in this case, conditioning decisions on only relevant previous observations reduces the complexity of finding an optimal strategy. Finally, in cases where conditioning a strategy on all previous relevant information is computationally prohibitive, it allows a decision to be conditioned on a subset of the relevant previous observations, offering a tradeoff between the quality of a decision strategy and the complexity of finding it.

For LIMIDs that satisfy special structural properties, called *soluble LIMIDs*, Lauritzen and Nilsson (2001) describe a variable elimination algorithm that finds an optimal strategy. They do not describe an algorithm that finds optimal solutions for LIMIDs in the general case, however. Instead, they propose an iterative algorithm, called *single policy updating* (SPU), that converges to a locally optimal solution. In this paper, we introduce a branch-and-bound graph-search algorithm that finds globally optimal solutions for LIMIDs, even in the general case.

Our approach builds on the work of Yuan et al. (2010), who describe a branch-and-bound search algorithm for solving a traditional influence diagram and show that it outperforms other approaches to solving multi-stage influence diagrams. We follow the same branch-and-bound approach in this paper, but with an important difference. The branch-and-bound algorithm for a traditional influence diagram is a tree-search algorithm in which each path through the tree represents perfect memory of a particular history of decisions and observations, in keeping with the no-forgetting assumption of a traditional influence diagram. By contrast, the branch-and-bound algorithm for LIMIDs presented in this paper searches in a much smaller *graph* in which different paths to the same node of the graph represent different histories with differences that are not "remembered." By collapsing the search tree into a much smaller search graph in which fewer histories are distinguished, the branch-and-bound approach can solve the optimization problem for LIMIDs much more efficiently than it can solve the optimization problem for a traditional influence diagram. That is, the new graph-search techniques we introduce leverage the opportunities for faster strategy computation provided by the LIMID model. We illustrate the advantages of the branch-and-bound approach in solving a simple maze-navigation example. Surprisingly, our initial experiments suggest that the branch-and-bound approach scales better than SPU in solving LIMIDs, even though SPU is an approximation algorithm.

The remainder of this paper is structured as follows. Section 2 provides a brief introduction to limited-memory influence diagrams and existing methods for solving them. Section 3 describes the details of our branch-and-bound search algorithm for LIMIDs. Section 4 describes the results of a simple empirical evaluation. Section 5 summarizes our contributions and discusses possible future work.

2 Background

We begin with a review of influence diagrams and limited-memory influence diagrams. We also introduce a simple example of a multi-stage decision problem that willnserve as a running illustration in this paper.

Influence Diagrams

An influence diagram is a directed acyclic graph G containing variables $\mathbf{V}$ of a decision domain. The variables can be classified into three groups, $\mathbf{V} = \mathbf{X} \cup \mathbf{D} \cup \mathbf{U}$, where $\mathbf{X}$ is the set of oval-shaped *chance* variables that specify the uncertain decision environment, $\mathbf{D}$ is the set of square-shaped *decision* variables that specify the possible decisions to be made in the domain, and $\mathbf{U}$ are the diamond-shaped *utility* variables that represent a decision maker's preferences. As in a Bayesian network, each chance 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 information states, where an information state is an instantiation of the variables with arcs leading into D_j; the selected action is conditioned on the information state. Incoming arcs to a decision variable are called *information arcs*; variables at the origin of these arcs are assumed 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, which means all the information variables of earlier decisions are also information variables of later decisions. We call these past information variables the *history*, and, for convenience, we assume that there are *explicit* information arcs from history information variables to decision variables. Finally, each utility node $U_i \in \mathbf{U}$ represents a function that maps each configuration of its parents to a utility value the represents the preference of the decision maker. (Utility variables typically do not have other variables as children, with the exception of multi-attribute utility/super-value variables.)

The decision variables in an influence diagram are traditionally assumed to be temporally ordered; this is the regularity assumption. 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, where $0 < k < n$, $\mathbf{I_k}$ 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, a partial order $\prec$ is defined on the influence diagram over $\mathbf{X} \cup \mathbf{D}$, as follows:

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

A solution to the decision problem defined by an influence diagram is a series of decision rules for the decision variables. A *decision policy* for a decision D is a mapping from each configuration of its parents to a probability distribution over the states of D. If the range of these probability distributions for a policy is $(0, 1)$, it is called a *pure* policy. A policy in which each action of its decision variable is equally likely is called a *uniform policy*. A *strategy* is a set $s = \{\delta_D : D \in \mathbf{D}\}$ of policies, one for each decision variable. A *pure strategy* is composed of only *pure policies*. The expected utility of a *strategy* is given by

$$EU(s) = \sum_{x \in \Omega_{\mathbf{U}}} (\prod_{X \in \mathbf{X}} P_X \prod_{D \in \mathbf{D}} \delta_D \sum_{u \in \mathbf{V}} (U_u(x))), \quad (2)$$

where $\Omega_{\mathbf{U}}$ is the possible set of states of $\mathbf{U}$. The goal is to find a *strategy* that returns the maximum expected utility.

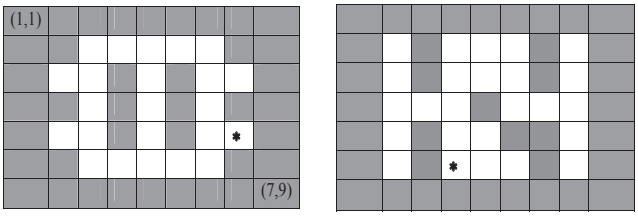


Figure 1: Two mazes. A star in a cell represents the goal.

In a *limited-memory influence diagram*, the traditional assumptions of regularity and no-forgetting are relaxed, allowing a wider range of decision problems to be modeled (Lauritzen and Nilsson 2001). Note that this requires all information arcs to be explicitly drawn in the diagram. To illustrate a LIMID, we use the following example.

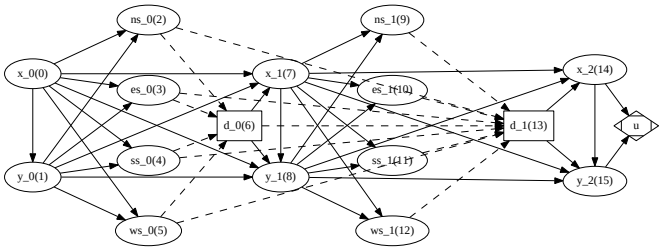
Example

Consider a maze navigation problem that is modeled as a multi-stage influence diagram (Horsch and Poole 1998; Nilsson and Hohle 2001; Yuan, Wu, and Hansen 2010). Figure 1 shows two instances of such a maze. White cells represent spaces where navigation is possible, and shaded cells represent walls. A robot is placed at one of the white cells in a maze. Its goal is to reach the goal state displayed by a star. The robot is equipped with sensors to detect walls and with motors to travel from one cell to another. Based on the sensor information, it can take one of five possible actions. It can either move to one neighboring white cell, or it stays at the same location. The movement of the robot is not fully deterministic. It moves to the intended white cell with a probability of 0.89 and fails to move with probability 0.089. The probability of sideways movement is 0.02 and of backward movement is 0.001. A move towards a wall has a probability of zero to succeed, and the remaining non-zero probabilities are normalized. The sensors are noisy – they are able to detect the walls correctly with a probability of .9 and find walls where there is none with a probability of .05. The robot selects an action at each stage. If the robot is in the goal state at the final stage, it receives an utility value of 1; otherwise, it receives nothing.

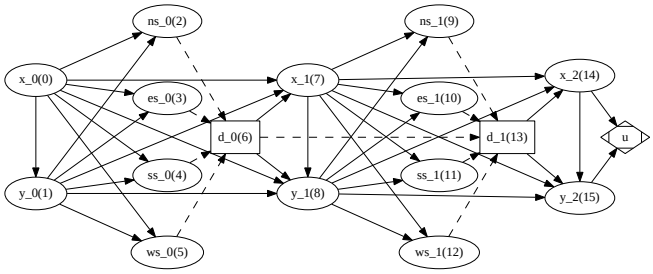
A two-stage influence diagram is shown in Figure 2(a). The variables x_i and y_i represent the coordinates of the location of the robot at time i. The variables $\{ns_i, es_i, ss_i, ws_i\}$ are the sensor readings in four directions at time i. The decision variable d_i represents one of the possible actions taken by the robot at that same time. A LIMID version of this decision making problem is displayed in Figure 2(b); it assumes that a decision is conditioned on all past decisions as well as the present states of the information variable when the decision is made, but not on the information variables for any previous stages. In other words, the robot makes decisions based on its current sensor readings only, without considering any previous sensor readings.

Approaches to Solving LIMIDs

Algorithms for solving traditional influence diagrams, such as the jointree algorithm, cannot be used to solve LIMIDs.



(a)



(b)

Figure 2: (a) An influence diagram with no-forgetting and (b) a LIMID, both for the maze navigation problem.

Instead, Nilsson and Lauritzen (2001) propose an iterative solution procedure, called single policy updating (SPU), that converges to a locally-optimal solution. SPU begins by creating an initial random policy for each of the decision variables. Then, in order from the last decision to the first, SPU tries to improve the policy for the current decision variable conditioned on all the other policies. Note that by fixing the other policies, the other decision variables are effectively transformed into random variables. Only the current decision policy is left, and can be optimized straightforwardly. One traversal of all the decisions is called one iteration of the algorithm. SPU iterates the policy updating process multiple times until a convergence criterion is met. When seeking a pure strategy, the criterion for convergence is that the maximum expected utility does not increase for two consecutive iterations. In that case, the decision strategy (the set of all policies) has reached a local optimum. de Campos and Ji (2008) propose an exact approach to solving LIMIDs that reformulates the LIMID as a credal network inference problem and uses mixed integer programming to solve the inference problem. We discuss their approach later in the paper.

3 Branch-and-Bound Search for LIMIDs

In this section, we describe how to formulate the problem of solving a LIMID as an AND/OR graph search problem. We show how to compute upper bounds for the OR nodes of the search graph in order to prune the search graph while still ensuring that an optimal solution is found. To perform efficient probabilistic inference when computing bounds, we use the incremental approach originally proposed by Yuan et al. (2009) for solving the MAP search problem in Bayesian networks, and also used by Yuan et al. (2010) in solving traditional influence diagrams.

AND/OR Graph Search

We represent our search space as an AND/OR search graph. AND nodes in the search graph represent the random variables of the LIMID; OR nodes represent decision nodes. Any arc emitting from an AND node has a probability attached to it; the sum of all the probabilities associated with the arcs of an AND is 1.0. Each arc emitting an OR node represents a decision alternative. The leaf nodes of the AND/OR graph have utility values attached to them that can be derived from the utility nodes of the LIMID.

The *valuation function* for each node is defined as follows: (a) for a leaf node, the value is its utility value, (b) for an AND node, the value is computed by multiplying the probabilities associated with the outgoing arcs to the utility values of its corresponding children and then summing these values, and (c) for an OR node, the value is the maximum of the utility values of each child node. We use this valuation function to determine the optimal strategy for the LIMID. Given an AND/OR search graph that represents all possible strategies for a LIMID, the LIMID is solved by finding a strategy that returns the maximum value at the root node.

We represent a strategy for a LIMID as a *policy graph*, which is a subgraph of an AND/OR graph that is defined as follows: (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 are 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. Given an AND/OR graph that represents all possible histories and strategies for a LIMID, the LIMID is solved by finding a policy graph with the maximum value at the root, where the value of the policy graph is computed based on the valuation function. Note that since all children of an AND node need to be evaluated and our search algorithm starts out with an AND node, we only obtain a solution (the optimal one) once the whole search is completed.

Yuan et al. (2010) proposed a depth-first branch-and-bound AND/OR tree-search algorithm to solve traditional influence diagrams. There are two main differences between that algorithm and our depth-first branch-and-bound AND/OR graph search algorithm for solving LIMIDs. The first difference is that we search in an AND/OR graph instead of an AND/OR tree. Because the no-forgetting assumption is relaxed in a LIMID, multiple decision histories can share the same information that is remembered by a decision in a LIMID and thus lead to a same decision scenario. In other words, there may be more than one incoming arc to an OR node in an AND/OR graph, in contrast to an AND/OR tree. The second difference lies in the upper bound calculation used to prune the search graph. Both algorithms coordinate the search with the join tree of a *relaxed* influence diagram in order to perform probabilistic inference and compute bounds. But since a LIMID allows information forgetting, the message passing on the incremental-bounds jointree has to be revised appropriately to solve a LIMID. We will discuss these two differences in detail in the next two subsections. But first we describe how to compute an upper bound for IDs.

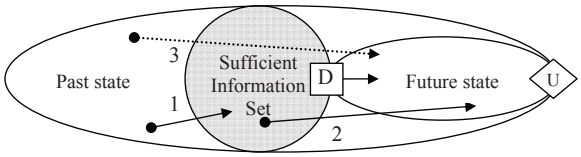


Figure 3: Relations between past and future information states and the minimum sufficient information set.

An upper bound for IDs

The branch-and-bound algorithm needs upper bounds on the values of OR nodes in order to prune the AND/OR search graph. Our approach to computing these bounds is adopted from the earlier work of Yuan et. al (2010), which in turn builds on the work of Nilsson and Höhle (2001). In this approach, we generate a *relaxed influence diagram* that is guaranteed to give upper bounds on the utility of the original ID, and is also easier to solve.

The relaxed ID is created by adding information variables for a decision, which in turn simplifies the ID by allowing *non-requisite* arcs to be removed. An information arc is called *non-requisite* (Lauritzen and Nilsson 2001; Nielsen and Jensen 1999) for a decision node D if

$$I_i \perp (\mathbf{U} \cap de(D))|D \cup (Pa(D) \setminus \{I_i\}), \qquad (3)$$

where $de(D)$ and $Pa(D)$ are the descendants and parents of D, respectively. A *reduction* of an influence diagram is obtained by deleting all the *non-requisite* information arcs (Nilsson and Hohle 2001).

Ideally, we want to add information variables that make some or all of the information arcs for a decision node *non-requisite* and also make the influence diagram easier to solve. The following theorem is proved by Nilsson and Höhle (2001). Let $\triangle_j$ be $\{D_1, ...D_j\}$ be a sequence of decision variables from stage 1 to stage j.

Theorem 1 *Let $fa(X) = Pa(X) \cup \{X\}$ be the family of variable X (i.e., the variable and its parents), and let $de(D)$ be the set non-descendants variables of D. For an influence diagram with the constrained order in Equation (1), if we add to each decision variable D_j the following new information variables in the order of $j = n, ..., 1$,*

$$N_j = \arg\min_{B \subseteq (B_j \cap nd(D_j))} \{|B||fa(\triangle_j) \perp (\mathbf{U} \cap de(D_j))|(B \cup \{D_j\})\}, \qquad (4)$$

where

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

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

$$(de(D_j) \cap \mathbf{U}) \perp fa(\triangle_{i-1})|fa(D_j). \qquad (5)$$

This theorem tells us that the optimal policy for each decision variable, D_j, in an ID depends on a set of information variables, N_j, or it is history-independent. This set N_j can be described as the current state of the decision problem, such that if the decision maker is informed of this state, it does not need to know of any past history to find an optimal optimal policy; in this way, it fulfills the Markov property.

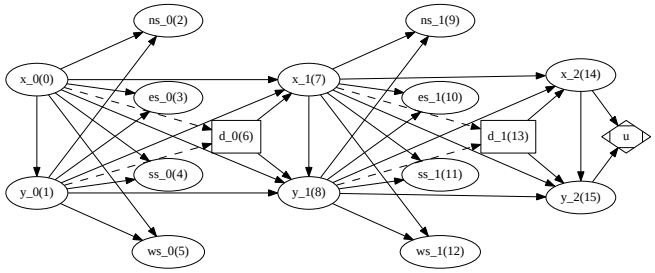


Figure 4: The relaxed influence diagram.

As an example, consider the maze problem described in Section 2. The robot has noisy sensors that provide imperfect information about its current location. If the robot knew its location with certainty before selecting an action, the utility it could achieve would be an upper bound on the utility it can achieve when its decision is based on imperfect information about its location. Adding information arcs from from the location variables to the decision variable makes the sensor readings *non-requisite* and the new influence diagram that is created can serve as an upper-bound influence diagram.

Figure 3 illustrates the idea behind this approach. N_j is called a *sufficient information set* (SIS) of D_j. The SIS for decision D, N_D, is shaded in the figure. Arc 1 shows that past information variables influence N_D and D, and arc 2 shows that $N_D \cup \{D\}$ influence the future information variables. $N_D \cup \{D\}$ d-separates the past states from the future states which is shown by arc 3.

We find the SIS for each decision in reverse time order, $D_n, ..., D_1$, making each SIS the information variables for its corresponding decision variable. We then obtain a relaxed ID by deleting the *non-requisite* arcs. Consider the LIMID in Figure 2 as an example. The original information set for decision d_1 is $\{d_0, ns_1, es_1, ss_1, ws_1\}$, and we find that its SIS is $\{x_1, y_1\}$. We also find that the SIS for d_0 is $\{x_0, y_0\}$. By making $\{x_1, y_1\}$ and $\{x_0, y_0\}$ information variables for d_1 and d_0, respectively, and after removing the non-requisite arcs, we obtain a relaxed influence diagram which is shown in Figure 4. The join tree for this ID shown in Figure 5. The advantage of having this relaxed influence diagram is that the largest clique-size of its strong join tree remains constant; for our example, it is 5. In the next subsection, we describe how to compute upper bounds for LIMIDs.

Calculating Probabilities for AND Nodes

We use the relaxed join tree of a LIMID to compute the probabilities and values needed by the AND/OR graph. It is important to do so in a manner that allows the calculation of probabilities and values to be synchronized with the AND/OR graph search. This is best illustrated using the

LIMID introduced in the background section. The relaxed join tree for the LIMID is shown in Figure 5. The join tree does not have a clique that contains all four variable; they are contained in different cliques. So we consider the expansion of these variables one by one. This will generate four layers of AND nodes and then one layer of OR nodes for our AND/OR graph search. A partial AND/OR graph is shown in Figure 6.

The AND/OR search graph is generated on-the-fly during the branch-and-bound search, and it is important to do so in a manner that allows probabilities and values to be computed efficiently. If we want to start by expanding ns_0, we need to compute the probabilities of ns_0, $P(ns_0)$, to label the outgoing arcs. $P(ns_0)$ can readily be looked up from clique $(0, 1, 2)$ after the initial join tree evaluation. Note that we make use of the relaxed join tree to compute the probabilities. We can do that since the same set of actions will transform both influence diagrams into the same Bayesian network. Adding information arcs in order to create the relaxed influence diagram only changes the final expected utility of the network.

Once ns_0 is generated, we can generate any of the $\{es_0, ss_0, ws_0\}$. If we want to generate es_0, we then set the current state of ns_0 as evidence in the clique $((0, 1, 2))$ containing ns_0, pass a message from that clique to the clique $((0, 1, 3))$ containing es_0 and obtain the probabilities for es_0. We can generate the rest of the probabilities in a similar way.

After the AND nodes of $\{ns_0, es_0, ss_0, ws_0\}$ are generated, we need to generate the OR node, d_0, and we need the upper bounds. Here comes the second major difference between our AND/OR graph search and the AND/OR tree search described by Yuan et al. (2010). Due to information forgetting, we do not need to set the states of $\{ns_0, es_0, ss_0, ws_0\}$ as evidence during the computation of expected utility values of d_0 since the states of information variables for d_0 are not remembered for future decisions. We can use the utility values calculated by an initial collection phase in the relaxed join tree as the upper bounds to select the most promising decision alternative to expand first. The exact utility value will be obtained once the subgraph rooted at the OR node is fully searched. If the upper bound of any remaining alternative is smaller than the exact value of the current best decision alternative, the remaining alternative is immediately pruned. Therefore, the basic idea for handling information forgetting in a LIMID is to only send a message to a future decision for calculating probabilities and utility values if the message contains information variables that are remembered by the future decision. Not sending some of the messages can slightly improve the efficiency of computing the probabilities and utility values. In general, we reduce the total number of messages passed in the relaxed join tree by a number equal to the number of stages on each evaluation of the search tree from its root to a leaf node for a LIMID.

Incremental Join Tree Evaluation

It is evident that repeated join tree evaluation is required to obtain the required probabilities. A naive way to perform this is to evaluate the full join tree each time an instantiated variable is set as evidence. But this is too costly. Inspired

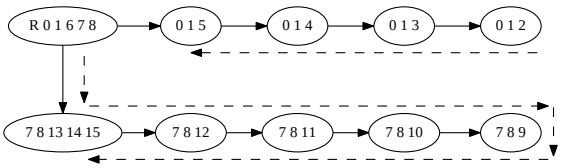


Figure 5: The strong join tree of the influence diagram in Figure 4

by the incremental join tree evaluation method proposed by Yuan and Hansen (2009) for solving the MAP problem, we use a similar incremental join tree evaluation method to compute probabilities and upper bounds.

Note that one of the partial orders for the LIMID shown in Figure 2(b) is

$$\{ns_0, es_0, ss_0, ws_0\} \prec \{d_0\} \prec \{ns_1, es_1, ss_1, ws_1\} \prec \{d_1\} \prec \{x_0, y_0, x_1, y_1, x_2, y_2, u\}, \quad (6)$$

If we can find an order of variable expansion that satisfies the constraints in Equation 6, then we can use that order for the *incremental join tree evaluation*. Given such an order, once a variable is searched, we know which variable to search next. For example, after we search ns_0, the only message that needs to be sent to obtain $P(es_0|ns_0)$ is the message from clique $(0, 1, 2)$ to $(0, 1, 3)$. If we choose the following order for our 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 scheme in the direction of the dashed arc in Figure 5 in one downward pass of the depth-first search to compute probabilities and upper bounds.

Adopting a depth-first-search strategy ensures that the search process does not need to jump to a different search branch before backtracking is required. We backtrack only when a search branch is searched to its end or when we realize the branch is not promising and should be pruned. When backtracking, we need to retract the evidence introduced since we generated that node and restore the join tree to its previous state. Instead of reinitializing the full join tree with evidence and reevaluating it, we cache the clique and separator potentials along the path of the message propagation before we set the evidence or override it with new messages. During backtracking, we simply restore the separator and clique potentials with the most recent respective cached potentials and remove these from the caches. For example, after evaluating $P(es_0 = 0|ns_0 = 0)$, we need to evaluate $P(es_0 = 1|ns_0 = 0)$. Instead of sending a message from $(0, 1, 2)$ to $(0, 1, 3)$ once again, we can just retrieve the last cached potential of $(0, 1, 3)$ when the last message was sent from $(0, 1, 2)$ and set the evidence $es_0 = 1$ to get the probability $P(es_0 = 1|ns_0 = 0)$.

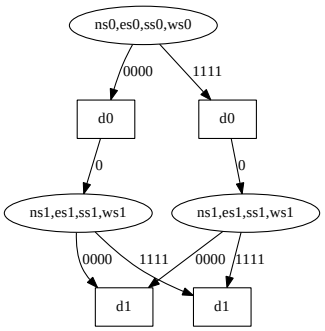


Figure 6: AND/OR graph (partial)

Context of OR Nodes

To solve a LIMID, we construct an AND/OR graph by merging multiple OR nodes (or decision nodes) that represent the same decision scenario into a single OR node; it means that the AND nodes corresponding to that decision scenario all emit arcs that point to the same OR node. To facilitate the discussion, we first define the concept of the context of an OR node. The *context* of an OR node is defined as the joint state of the information variables remembered by the corresponding decision variable in the LIMID, including both past random variables and decision variables. More formally, the context of an OR node is a set $C_D = S_{I_{i-1}} \cup S_{D_{past}}$, where $S_{I_{i-1}}$ is the set of states of the past random variables remembered by D and $S_{D_{past}}$ is the set of states of the past decision variables remembered by D. For example, Figure 6 shows a partial AND/OR graph for the LIMID in Figure 2(b). Note that the AND/OR graph is condensed for ease of illustration as the individual random variables for the sensors are grouped together into one AND layer. In our actual AND/OR graph, they correspond to multiple AND layers. The context of the bottom-right OR node in the Figure is $\{1, 1, 1, 1, 0\}$, where $\{1, 1, 1, 1\}$ is the set of present information states, and $\{0\}$ is the last action taken. The previous sensor readings are totally forgotten. That is why the two paths starting from the root converge to this OR node. Similar concepts of context have been proposed before, e.g. (Marinescu 2009; Smith, Holtzman, and Matheson 1993).

Merging multiple OR nodes can reduce the amount of computation as well as the size of the search graph. For any pair of decision variables in the same OR layer, if they have the same contexts, these two decisions are expected to share the same optimal action since their information sets are identical. Consequently, the utility values returned by these actions will also be the same. Therefore, it makes sense to store the context of an OR node along with the expected utility value it receives whenever the subgraph rooted at this node is completely expanded and the expected utility value is calculated. Whenever an OR node is ready to be generated, its context is calculated and then checked against the contexts of the other OR nodes at the same layer. If another previously expanded OR node has the same context, the new OR node is not generated. Instead, the existing OR node receives an additional arc from the parent of the new OR node, and the search can immediately jump to another branch or backtrack to a previous layer. Otherwise if no OR node has the same context, the new OR node is generated, and the search continues into deeper layers. The more duplicate OR nodes there are, the more savings in computation time. Because OR nodes can be merged, the AND/OR graph has a much smaller size than the corresponding (and equivalent) AND/OR tree, which lets our algorithm solve larger decision problems.

4 Empirical Evaluation

We evaluated the new search algorithm by comparing its performance to the performance of the single policy updating method of Lauritzen and Nilsson (2001) and the branch-and-bound AND/OR tree search algorithm for solving traditional influence diagrams (Yuan, Wu, and Hansen 2010), in terms of both the quality of the policies and the scalability of the algorithms. We performed the experiments on a PC with a Pentium-Quad-Core processor (2.66 GHz), 4GB of RAM, and a 64-bit Windows XP operating system. All the algorithms were implemented in C++.

In (Madsen and Nilsson 2001), the SPU method for solving LIMIDs was implemented on HUGIN, Shafer-Shaney, and Lazy propagation architectures. We implemented SPU on the HUGIN architecture. The Hugin architecture cannot be used to solve general LIMIDs (Madsen and Nilsson 2001) since the policies are multiplied onto the clique potentials of the jointree. Once a policy is retracted, the earlier clique potentials cannot be obtained. However, we are able to circumvent this problem by caching the clique potentials. We start out with a *uniform policy* for each decision variable, but we find a *pure strategy* once the algorithm finishes its execution.

Table 1 shows the results for the two maze problems for different numbers of stages. For all of the LIMIDs, we assume memory of all previous decisions but access to only the current sensor readings, with no memory of previous sensor readings. The last column shows the possible number of policies for the last decision node. Since the last decision node contains the largest information set of all decision nodes and the domain sizes of these variables do not decrease over time, the possible number policies is the highest for this node. The results of the branch-and-bound search for the traditional influence diagram are not shown for more than five stages because the algorithm did not finish running in the allotted time (1.5 hours). As expected, the expected utilities returned by the LIMID algorithm are smaller than the expected utilities returned by the algorithm for traditional influence diagrams, which remembers the entire history. Information loss typically results in worse decision policies. However, the LIMID algorithm runs considerably faster than the algorithm for traditional IDs. This happens for two reasons: less message passing is required during the expansion of the AND/OR graph and duplicate OR nodes are merged based on their $contexts$. The sizes of the final policy graphs for LIMIDs are much smaller than the policy trees for traditional influence diagrams. Consequently, the LIMID algorithm can solve the maze problem for up to 9

maze	stages	DFBnB ID time	DFBnB ID size	DFBnB ID EU	DFBnB LIMID time	DFBnB LIMID size	DFBnB LIMID EU	SPU LIMID time	SPU LIMID EU	SPU LIMID size	Min. # of Strategies
a	2	312ms	783	0.098	125ms	543	0.083	46ms	0.063	320	4^{4^3}
	3	4.5s	12,559	0.137	281ms	1,039	0.106	63ms	0.074	1,344	4^{4^4}
	4	2m25s	200,975	0.187	500ms	1,535	0.148	188ms	0.126	5,440	4^{4^5}
	5	1h19m58s	3,215,631	0.259	2s	2,031	0.194	13.3s	0.073	21,824	4^{4^6}
	6	-	-	-	6s	2,527	0.241	-	-	-	4^{4^7}
	7	-	-	-	18s	3,023	0.287	-	-	-	4^{4^8}
	8	-	-	-	1m8s	3,519	0.336	-	-	-	4^{4^9}
	9	-	-	-	4m	4,015	0.405	-	-	-	$4^{4^{10}}$
b	2	390ms	783	0.206	125ms	543	0.194	46ms	0.099	320	4^{4^3}
	3	5s	12,559	0.273	421ms	1,039	0.232	78ms	0.153	1,344	4^{4^4}
	4	1m48s	200,975	0.339	1s	1,535	0.278	297ms	0.127	5,440	4^{4^5}
	5	43m56s	3,215,631	0.408	3s	2,031	0.328	20.9s	0.229	21,824	4^{4^6}
	6	-	-	-	11s	2,527	0.348	-	-	-	4^{4^7}
	7	-	-	-	52s	3,023	0.362	-	-	-	4^{4^8}
	8	-	-	-	3m18s	3,519	0.370	-	-	-	4^{4^9}
	9	-	-	-	13m26s	4,015	0.388	-	-	-	$4^{4^{10}}$

Table 1: Comparison of three algorithms (DFBnB for regular influence diagram, DFBnB for LIMID, and SPU for LIMID) in solving maze problems a and b for different numbers of stages. Note that "size" is the count of nodes in the part of the search tree (graph for LIMID) containing the optimal policy tree (graph for LIMID), and the total size of all the policies combined for SPU. EU shows the expected utility of each algorithm.

stages in the allotted time, while the traditional ID algorithm can only solve the maze problem for up to 5 stages.

The SPU algorithm is a local search method for solving LIMIDs, so it is not surprising that its expected utilities are always lower than for our optimal AND/OR graph search algorithm. However, it is surprising at first that the SPU algorithm failed to solve the maze problems for more than five stages. The reason for its failure is the substantial increase in the size of the join trees used by the SPU algorithm to perform the local search. Note that the family of nodes of any decision must be contained in at least one of the cliques in the join tree. Since we assume that all previous decisions are remembered (though not the previous observations), the largest clique size still grows exponentially. As a result, the SPU algorithm ran out of memory and could not solve the problems for more than five stages.

Table 2 shows our results when the maze problem is modified by removing some but not all noise from the sensors. The scalability of the two search-based algorithms is improved because zero-probability branches can be removed from the search graph. The algorithm for solving traditional IDs can handle up to 7 stages now, while our LIMID algorithm can handle up to 10 stages within the allotted time (35 minutes). However, the scalability of the SPU algorithm is not improved because the SPU algorithm uses a tabular representation for the potentials and policies and cannot exploit zero probabilities in the domain. The minimum sizes of possible strategies remain the same as shown in Table 1 due to the same domain sizes of variables.

Besides our branch-and-bound algorithm, the only other algorithm that can find optimal solutions for LIMIDs of which we are aware is the approach of de Campos and Ji (2008), which reformulates a LIMID as a credal network inference problem and uses mixed integer programming to solve the inference problem. They also compare their approach to the SPU algorithm and find that their algorithm runs two orders of magnitude slower than the SPU algorithm in solving their test problem exactly. Since our branch-and-bound algorithm runs significantly faster than the SPU algorithm, it suggests that our branch-and-bound approach can outperform the other exact approach of reformulating a LIMID as a credal network inference problem. The largest problem solved in (de Campos and Ji 2008) has approximately 2^{120} strategies, whereas the maze problem we solve has more than $4^{4^{11}}$ strategies.

5 Conclusion

A limited-memory influence diagram (LIMID) allows the "regularity" and "no-forgetting" assumptions of the traditional influence diagram to be relaxed, so that a wider and more realistic range of decision problems can be modeled and solved. With these changes, however, most exact algorithms for solving traditional influence diagrams cannot be used to solve LIMIDs. Single policy updating (SPU) is an approximation algorithm for solving LIMIDs that converges to a locally-optimal solution. In this paper, we have introduced a branch-and-bound approach to solving LIMIDs that not only finds optimal solutions, it scales better than the SPU algorithm, at least in solving the maze problem we have used as an example.

Our branch-and-bound approach performs as well as it does even though the bounds used in our implementation

		DFBnB ID			DFBnB LIMID			SPU LIMID		
maze	stages	time	size	EU	time	size	EU	time	EU	size
a	2	109ms	224	0.102	93ms	363	0.083	32ms	0.064	320
	3	250ms	849	0.141	219ms	687	0.106	62ms	0.070	1,344
	4	1.1s	2,843	0.195	360ms	1,011	0.148	203ms	0.106	5,440
	5	6.4s	9,076	0.273	1s	1,335	0.194	13.3s	0.039	21,824
	6	34.6s	28,413	0.366	4s	1,659	0.241	-	-	-
	7	3m22s	88,167	0.458	12s	1,983	0.287	-	-	-
	8	-	-	-	46s	2,307	0.336	-	-	-
	9	-	-	-	2m39s	2,631	0.405	-	-	-
	10	-	-	-	9m20s	2,955	0.468	-	-	-
b	2	93ms	261	0.209	94ms	363	0.194	31ms	0.102	320
	3	313ms	1,136	0.281	297ms	687	0.232	62ms	0.124	1,344
	4	1.1s	4,244	0.349	734ms	1,011	0.278	203ms	0.091	5,440
	5	3.8s	15,030	0.430	2s	1,335	0.328	13.3s	0.088	21,824
	6	14.8s	52,240	0.518	7s	1,659	0.348	-	-	-
	7	1m02s	180,795	0.578	35s	1,983	0.362	-	-	-
	8	-	-	-	2m13s	2,307	0.370	-	-	-
	9	-	-	-	8m55s	2,631	0.388	-	-	-
	10	-	-	-	34m28s	2,955	0.395	-	-	-

Table 2: Comparison of three algorithms (DFBnB for regular influence diagram, DFBnB for LIMID, and SPU for LIMID) in solving maze problems (with accurate sensors) a and b for different numbers of stages.

are quite simple. (They are equivalent to assuming perfect information.) These bounds can likely be significantly improved, allowing more pruning and faster search. We plan to implement improved bounds and evaluate the approach on a wider range of test problems.

Acknowledgements

This research was supported by the National Science Foundation grants IIS-0953723 and EPS-0903787. We thank the anonymous reviewers for their constructive comments.

References

de Campos, C. P., and Ji, Q. 2008. Strategy selection in influence diagrams using imprecise probabilities. In McAllester, D. A., and Myllymäki, P., eds., *UAI*, 121–128. AUAI Press.

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.

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

Madsen, A. L., and Nilsson, D. 2001. Solving influence diagrams using hugin, shafer-shenoy and lazy propagation. In Breese, J. S., and Koller, D., eds., *UAI*, 337–345. Morgan Kaufmann.

Marinescu, R. 2009. A new approach to influence diagrams evaluation. In Bramer, M.; Ellis, R.; and Petridis, M., eds., *SGAI Conf.*, 107–120. Springer.

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.

Smith, J. E.; Holtzman, S.; and Matheson, J. E. 1993. Structuring conditional relationships in influence diagrams. *Oper. Res.* 41:280–297.

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

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