Dynamic Programming and Influence Diagrams

JOSEPH A. TATMAN AND ROSS D. SHACHTER

Abstract — The concept of a super value node is developed to extend the theory of influence diagrams to allow dynamic programming to be performed within this graphical modeling framework. The operations necessary to exploit the presence of these nodes and efficiently analyze the models are developed. The key result is that by representing value function separability in the structure of the graph of the influence diagram, formulation is simplified and operations on the model can take advantage of the separability. From the decision analysis perspective, this allows simple exploitation of separability in the value function of a decision problem which can significantly reduce memory and computation requirements. Importantly, this allows algorithms to be designed to solve influence diagrams that automatically recognize the opportunity for applying dynamic programming. From the decision processes perspective, influence diagrams with super value nodes allow efficient formulation and solution of nonstandard decision process structures. They also allow the exploitation of conditional independence between state variables. Examples are provided that demonstrate these advantages.

I. INTRODUCTION

Within decision analysis, a graphical modeling language known as influence diagrams have been used for several years as an aid for formulation of decision analysis problems in both the academic environment and professional practice [6], [10]. Recently, the development of computer tools have made it feasible not only to formulate, but also to analyze and solve decision models as presented in Shachter [16].

Influence diagrams are becoming an effective modeling framework for a diverse array of problems involving probability. This effectiveness has many different aspects. First, influence diagrams capture both the structural and qualitative aspects of the decision problem and serve as the framework for an efficient quantitative analysis of the problem. There is no need to translate the model from a framework that is effective for modeling and formulation to one that is effective for analysis, since influence diagrams are effective for both. This dual role of influence diagrams also allows partial solution of an influence diagram resulting in a simplified but meaningful intermediate model. Influence diagrams allow efficient representation and exploitation of the conditional independence in a decision model. Finally, influence diagrams have proven to be an effective tool for not only communicating decision models among decision analysts and decision makers, but also for communicating between the analyst and the computer.

The power of the influence diagram derives from the fact that it represents the most important information in the decision model, the probabilistic and functional dependencies and the information flow as a graph. It is straightforward to develop software to process graphs, and so it is straightforward to develop software to process the most important information in the model. Also, graphical representations are natural and intuitive for the decision maker [13], and so the most important information in the model is in a form that lends itself to communication among the decision makers and experts.

A shortcoming of the traditional influence diagram is that the separable nature, if any, of a value function is not revealed in the graphical structure and thus cannot be exploited. In this paper we define separable as follows. This use of the word is similar to its use in Luenberger [9] and Larson [7].

Definition: Let \( x \) be an \( n \)-vector and let there be \( q \) subvectors, \( x_1, x_2, \ldots, x_q \), such that the components of each \( x_i \) are a subset of the components of \( x \). A value function \( g(x) \) is separable if

\[
g(x) = \sum_{i=1}^{q} g_i(x_i) \quad \text{or} \quad g(x) = \prod_{i=1}^{q} g_i(x_i).
\]

The inability to represent such separability of the value function prevents us from performing dynamic programming in influence diagrams. This problem was first addressed in Olmsted [11].

The objective of this paper is to bring the separability information out of the single value node and represent it explicitly in the graph. This is accomplished by allowing multiple value nodes, including sum and product nodes,
collectively called super value nodes. The separable nature of the value function is represented in the structure of the graph using these multiple value nodes, and the decision problem can be solved (carrying out the dynamic programming operations) via influence diagram reductions.

The Markov decision process example at the end of this paper demonstrates how the multiple value node concept is used to formulate and solve this standard stochastic dynamic programming problem. The standard steps of dynamic programming are seen to correspond very naturally to the influence diagram reductions. Also, the important structural characteristics of the Markov decision process are readily identifiable in the influence diagram graph. The capability to perform dynamic programming easily within the influence diagram framework should increase its accessibility.

The second example is meant to show that dynamic programs with nonstandard problem structures can be naturally solved within the influence diagram framework, without reformulation and state augmentation. In general, one need not aggregate the state variables of a Markov decision process into a single state variable, but may treat them separately, possibly taking advantage of conditional independence between them. The example shows that a Markov decision process with time lag can be solved directly in the influence diagram framework. Solving this problem as an influence diagram decreases from $n^2$ to $n^2$ both the size of the largest operation and the largest array storage requirement.

The satellite power example demonstrates that the multiple value node concept is useful for decision models that are not dynamic programming problems, but do have separable value functions. In the satellite power problem, the size of the largest operation and the largest array required to solve the model was reduced from $3^{10}$ to $3^2$. The total number of operations was reduced from $1.3 \times 10^3$ to $1.2 \times 10^3$.

We begin our development with a review of influence diagram fundamentals. The special properties of the expectation and maximization operators when applied to separable value functions are then developed in terms of influence diagrams with multiple value nodes. This motivates the development of the value node structure necessary for representing separability. Reductions of the influence diagram and an algorithm are then developed that allow us to exploit this new structure in solving decision problems. Finally, the three examples are presented.

II. INFLUENCE DIAGRAMS

A. Definitions

Influence diagrams are a graphical modeling language that can represent probabilistic inference and decision analysis models. They are hierarchical. The top level is a graph, and the second level is a frame of data associated with each node. Influence diagrams are mathematically precise in that the structure of the graph is a simple, but fundamental representation of the relationships among a set of variables. Because they are mathematically precise, influence diagrams can be used in both the formulation and analysis of a decision problem. This distinguishes the influence diagram from many other graphical modeling languages.

More formally, an influence diagram is a directed graph with no cycles. The nodes in the graph correspond to variables in the model; decision nodes correspond to variables under the control of the decisionmaker, and chance nodes correspond to random variables or random events. The decision nodes must be totally ordered; that is, there must be a directed path that contains all the decision nodes. The value node represents the objective function of the model, to be maximized in expectation. An influence diagram need not contain a value node. If it does, the value node can have no successors and the influence diagram is said to be oriented. In this paper we give a node and the variable that it corresponds to the same name.

Arcs in the graph represent the relationships among the nodes, that is, the associated variables. Arcs into a decision node indicate what information will be known to the decisionmaker at the time the decision is made. Arcs into a chance node indicate which variables condition the probability distribution for the associated random variable. Arcs into a value node indicate which variables condition the associated expected value. Importantly, the absence of an arc between two nodes indicates that the associated variables are conditionally independent. The influence diagram as a whole represents a specific expansion of the joint probability distribution of the random variables in the problem.

An example of an influence diagram is shown in Fig. 1. The graph may be interpreted as follows. Nodes $x$, $y$, and $z$ are chance nodes, $d$ is a decision node, and $J$ is the value node. Variables $x$ and $z$ are independent given $y$. The decisionmaker will know the outcome of random variable $y$ before decision $d$ must be made but will not know the outcomes of $x$ and $z$. Our objective is to maximize the expected value of $J$, which is conditioned on $d$, $x$, $y$, and $z$.

The set of all nodes in an influence diagram is designated by $N$. The set of all chance nodes is designated by $C$, the set of all decision nodes by $D$, and the set containing the value node by $V$. We will need the following definition.

**Definition:** If nodes $n_1$ and $n_2$ are elements of $N$ and there is a directed arc from $n_1$ to $n_2$, then $n_1$ is called a direct predecessor of $n_2$ and $n_2$ is called a direct successor of $n_1$. 

![Fig. 1. Simple influence diagram with data frame for one node.](image-url)
For decision node $d$, $I(d)$ designates the information available to the decisionmaker when decision $d$ must be made; that is, $I(d)$ is the set of direct predecessors of $d$. We call these informational predecessors. For chance or value node $x$, $C(x)$ designates the set of variables that condition $x$; that is, the direct predecessors of $x$. We call these the conditional predecessors of $x$. Similarly, if $X$ is a set of nodes, we use $C(X)$ to denote the set of all nodes which are direct predecessors of some node in $X$. The indirect predecessors of a node $y$, $C_{\text{ind}}(y)$, are all nodes in the diagram on a directed path to $y$. We use $S(y)$ to designate the direct successors of $y$ and $S_{\text{ind}}(y)$ to designate the indirect successors of $y$, that is, all nodes which are on a directed path from $y$.

With each node we associate a frame of data. For a chance node $x$, this includes the outcome space of $x$, $\Omega_x$, and the conditional probability distribution of $x$, $\pi_x$. For each decision node $d$, this includes the alternatives of the associated decision variable, $\Omega_d$. Finally, the data frame for value node $r$ contains the conditional expected value of $r$ conditioned on the predecessors of $r$; that is, we have the following:

$$E[r|C(r)] = g(C(r)).$$

(2)

The conditional expectation of $r$ is actually a deterministic function of the conditioning variables. The outcome space for $r$ is $\Omega$, and is assumed to be a bounded set of real numbers for this paper.

As an example, Fig. 1 shows the data frame for chance node $x$ (the two sets of probabilities for $x$ correspond to the two outcomes of $y$ which conditions $x$). Note that the information stored in the frame of data for each node, along with the information in the graph, completely defines the model. The complete information for a chance node $x$ in the model might be stored on the computer as

$$\{x \text{ (kind chance)},$$

$$(\text{preds } y),$$

$$(\text{outcomes } 1 \ 2 \ 3),$$

$$(\text{probs } (0.1 \ 0.2 \ 0.7),$$

$$(0.3 \ 0.1 \ 0.6)).$$

Note that this includes the graphical as well as the non-graphical information for node $x$.

We use $\Omega_x$, where $X$ is the set of chance or decision nodes $\{x_1, x_2, \ldots, x_n\}$, to mean the cross product space $\Omega_{x_1} \times \Omega_{x_2} \times \cdots \times \Omega_{x_n}$. We use $|\Omega_x|$ to denote the number of elements in $\Omega_x$.

To be well defined, the decisions in an influence diagram must be totally ordered. One of the assumptions in decision theory, implicit in the decision tree representation, is that a decisionmaker making decision $d$ knows the chosen alternative for all previous decisions and all information known for those previous decisions. This "no forgetting" assumption implies that every decision node $d$ should have as direct predecessors all previous decision nodes plus all direct predecessors of all previous decision nodes. Since this would clutter up the influence diagram, some of these arcs may be implied without being explicitly drawn. Only those arcs into $d$ that represent information not previously known for any decision and those arcs necessary to establish a total ordering of the decisions need to be shown explicitly in the diagram. The others, "no forgetting" arcs, may be omitted in a drawing of the diagram, but must be included when performing the analysis. Finally, if a node in an oriented influence diagram is not the value node and has no successors, then no matter what value it assumes no other node in the diagram is affected. Such nodes are called barren and can be deleted from the diagram without affecting the optimal policy or the maximal expected value.

B. Reductions

Each influence diagram with a value node may be solved for a maximum expected value and an optimal policy [16]. To accomplish this we use a set of reductions, or value preserving transformations, of the influence diagram. We define a reduction as follows.

Definition: An operation that transforms influence diagram $A$ to influence diagram $B$ is a reduction of influence diagram $A$ (or value preserving transformation) if $B$ has a joint probability distribution that is either equal to the joint probability distribution of $A$ or the marginal of it, and if the optimal policy and expected value of $B$ is the same as that of $A$.

A sufficient set of reductions for the influence diagram with a single value node is shown in Fig. 2. The four basic reductions are arc reversal using Bayes theorem, summing a variable out of the joint, removing a chance node by expectation and removing a decision node by maximization [6], [11], [16].

Arc reversal between two chance nodes, as in Fig. 2(a), corresponds to applying Bayes theorem and can be performed whenever there is no other path between them. As can be seen in the influence diagram, we begin with conditional probability distributions $P(x|\cdot)$ and $P(y|x, \cdot)$. We would like to apply Bayes theorem to obtain conditional probability distributions $P(x|y, \cdot)$ and $P(y|\cdot)$. In the mathematics, we have the following two step operation:

$$P(y|a, b, c) \leftarrow \sum_x P(x, y|a, b, c)$$

$$= \sum_x P(y|x, a, b) \cdot P(x|a, b) \quad (3)$$

$$P(x|y, a, b, c) \leftarrow \frac{P(x|y, a, b, c)}{P(y|a, b, c)}$$

$$= \frac{P(y|x, a, b) \cdot P(x|a, b)}{P(y|a, b, c)}. \quad (4)$$

Note that as a result of calculating the conditional joint, as an intermediate step, that chance nodes $x$ and $y$ inherit each others predecessors.
Removal of a chance node by summing it out of the joint, as in Fig. 2(b), can be performed whenever a chance node has a single successor and that successor is a chance node. In the mathematics, this operation corresponds to

\[
P(y|a, b, c) = \sum_x P(x, y|a, b, c)
\]

\[
= \sum_x P(y|x, a, b, c) \cdot P(x|a, b).
\]  

(5)

Note that as a result of the calculations, chance node \( y \) inherits the predecessors of chance node \( x \).

Removal of a chance node by expectation, as in Fig. 2(c), can be performed whenever the only successor of a chance node is the value node. In the mathematics this corresponds to

\[
E[J|a, b, c] = E[E[J|x, b, c]|a, b].
\]  

(6)

The validity of this version of expectation expansion is discussed after Lemma 4 in this paper. Note that as a result of the expectation operation that value node \( J \) representing the conditional expected value of \( J \), inherits the predecessors of chance node \( x \).

Finally, the removal of a decision node by maximization, as in Fig. 2(d), can be performed whenever the decision node has the value node as its only successor and all conditional predecessors of that value node, other than the decision node, are informational predecessors of the decision node. In the mathematics, decision node removal corresponds to

\[
E[J|b] = \max_d [E[J|b, d]]
\]

and

\[
d^* = \arg \max_d [E[J|b, d]].
\]  

(7)

Note that value node \( J \) does not inherit any predecessors of \( d \) as a result of this operation. Nodes such as chance node \( a \) in Fig. 2(d) represent information available to the decisionmaker, that does not effect the values. The arcs from such nodes can be ignored when performing the maximization.

It might require a sequence of reductions to remove a node from the influence diagram. For example, a chance node with only chance node successors can be removed from the influence diagram by a series of arc reversals and a summation.

This concludes our summary of the fundamentals of influence diagrams. Again, for a formal presentation of these fundamentals see [16], [17]. We are now ready to explore the relationship between separability of the value function and solving decision problems in influence diagrams.

III. SEPARABILITY PROPERTIES AND SUPER VALUE NODE STRUCTURES

A. Super Value Nodes

Solving a decision problem involves applying a sequence of maximization and expectation operators to the value function. In the influence diagram this corresponds to removing decision and chance nodes at the value node by performing maximizations or expectations as in Fig. 2(c) and (d). The maximization and expectation operators have special properties when applied to value functions which are separable.

The special properties of the maximization and expectation operators when applied to separable functions are important in that, under certain conditions, they allow maximizations and expectations to be performed over an addend or factor in the value function instead of over the entire value function. In those cases, only a subspace of the value function needs to be examined. This may significantly reduce the dimensionality of the operations necessary to solve a decision problem. These special properties are the foundation of dynamic programming.

In order to take advantage of these properties in the influence diagram framework, it is first necessary to represent explicitly the separable nature of the value function in the influence diagram. A natural technique for doing this, that is consistent with the essential nature of the influence diagram, is to decompose the single value node into a set of value nodes.

Consider the influence diagram in Fig. 3(a) with value function \( J \) written in terms of conditional expected values...
Thus taking expectation at value node as expectation of the expectation of a sum is the sum of the expectations, and associated with value node \( E[\text{uly}, \text{rlx}, \text{sld}] \) and value node \( \text{w} \) represents the factors of \( w \). In removing \( x \) from the influence diagram by expectation we now observe readily from the graph that \( w \) is a product of \( r \) and \( s \) and that \( x \) is not a function of \( x \). Thus taking expectation at value node \( w \) with respect to \( x \) is equivalent to taking expectation at value node \( r \) with respect to \( x \).

There is an analogous situation for maximization. In the influence diagram in Fig. 3(c) we note that value node \( J \) is the sum of \( u \) and \( w \) and that \( u \) is not a function of \( d \). Therefore, maximizing \( J \) with respect to \( d \) is equivalent to maximizing \( w \) with respect to \( d \). We then see that \( w \) is the product of \( r \) and \( s \) and that \( r \) is not a function of \( d \). So, maximizing \( w \) with respect to \( d \) is equivalent to maximizing \( s \) with respect to \( d \). The sum and product nodes represent the necessary information in the graph for taking advantage of the separability in the value function.

Thus in our framework there are two kinds of value nodes. A super value node is either a sum node or product node, and a nonsuper value node is any other value node. A general set of value nodes, \( V \), has a particular structure. There is exactly one value node, the terminal value node, which has no successors. This represents the objective function for the model. If there is more than one value node, then the terminal value node must be a super value node. Super value nodes can only have value nodes (either super or nonsuper) as conditional predecessors. Nonsuper value nodes, on the other hand, can only have chance and decision nodes as conditional predecessors.

It is convenient to restrict all value nodes to have no more than one successor. This forces the value node structure to be that of a tree. The “root” is the terminal value node and the “leaves” are nonsuper value nodes that have only chance and decision node predecessors. This facilitates the development of the reductions and algorithm necessary to solve multiple value node influence diagrams.

It does not restrict the generality of the influence diagram in representing decision models with separable value functions, though it may increase the number of value nodes necessary to model the value function. This assumption is used in proving Lemmas 2 and 3.

We formalize the previous definitions.

**Definition:** A sum node \( r \) with predecessors \( r_1, r_2, \ldots, r_n \) is a value node representing the conditional expected value

\[
E[r | r_1, r_2, \ldots, r_n] = g(r_1, r_2, \ldots, r_n) = r_1 + r_2 + \cdots + r_n.
\]  

(9)

**Definition:** A product node \( r \) with predecessors \( r_1, r_2, \ldots, r_n \) is a value node representing the conditional expected value

\[
E[r | r_1, r_2, \ldots, r_n] = h(r_1, r_2, \ldots, r_n) = r_1 r_2 \cdots r_n.
\]  

(10)
Definition: A super value node is either a sum node or product node. A super value node has only value node predecessors. Value nodes which are not super value nodes, nonsuper value nodes, have only chance and decision node predecessors. Super and nonsuper value nodes have at most one successor.

Definition: In each oriented influence diagram there is exactly one value node, the terminal value node, which has no successors. This node represents the objective function of the model.

Note that the predecessors of a super value node (which will all be value nodes) could be collapsed into the super value node at any point. For example, in Fig. 3(b), value nodes w and v can be removed into J. This corresponds to rewriting the composite function $E[J[u, w] = g(u, w)$, where u is a function of y and z, and w is a function of d, x, and y, as $E[J[u, s, y, z] = g_2(r, s, y, z)$. Extending this reasoning, the entire set of multiple value nodes can be reduced into the terminal value node at any point reducing any multiple value node influence diagram to a single value node influence diagram. These ideas are formalized as follows.

Lemma 1: If r is a super value node in an influence diagram, then $C(r)$ may be reduced into r.

Proof: Let $C(r) = \{r_1, r_2, \ldots, r_n\}$ where $r_i \in V$ for all i because all predecessors of super value nodes must be value nodes. Let the predecessors of the value nodes in $C(r)$ be $C(C(r)) = \{x_1, x_2, \ldots, x_m\}$. Then $E[r_1, r_2, \ldots, r_n] = g_1(r_1, r_2, \ldots, r_n)$ for some function $g_1$ and $E[r, x_1, x_2, \ldots, x_m] = h_i(x_1, x_2, \ldots, x_m)$ for each i and some functions $h_i$. Thus r is a composite function of the $x_i$'s and can be rewritten as $E[r, x_1, x_2, \ldots, x_m] = g_2(x_1, x_2, \ldots, x_m)$.

Theorem 1: The set of all value node indirect predecessors of r, $C_{Ind}(r) \cap V$, for some super value node r, may be reduced into r.

Proof: Either $C(r)$ is contained in V or $C(r)$ is contained in $C \cup D$ by definition. If $C(r)$ is contained in $C \cup D$, then $C_{Ind}(r) \cap V$ is empty (for elements of V cannot have chance or decision node successors) and we are done. If $C(r)$ is contained in V, then $C(r)$ may be reduced into r by Lemma 1. There are a finite number of value nodes, so this step may be repeated until $C_{Ind}(r) \cap V$ is empty.

The next step is to examine the separability properties of the maximization and expectation operators in terms of the influence diagram with multiple value nodes. It will then be possible to develop influence diagram reductions, and an algorithm based on these, that allow us to exploit the super value node structure and thus the separable nature of value functions.

It is necessary to define the concept of functional predecessor. Simply, node x is a functional predecessor of value code r if the function associated with node r could be written as a deterministic function for which x is an argument. Consider again the example in Fig. 3(c). We have the functions $E[J[u, w] = g_3(u, w)$, $E[w, r, s] = g_4(r, s)$, $E[r, x] = g_3(x)$, $E[y, d, y] = g_4(d, y)$, and $E[u, y, z] = g_4(y, z)$. Value node J is directly a function of u and w and a composite function of d, x, y, and z, since it could in fact be written as $E[J[d, x, y, z] = g_4(d, x, y, z)$. Nodes d, x, y, and z are value nodes r, s, u, and w are all functional predecessors of J. We denote this set as $C_f(s)$ for value node s. To contrast this concept to that of indirect predecessor, look ahead to Fig. 8(c). Consider sum node $v_2$ in that influence diagram. The functional predecessors of $v_2$, $C_f(v_2)$, are $x_2$ and $d_2$. Node $v_2$ can be written as a function of these. On the other hand, the indirect predecessors of $v_2$ are $x_0$, $d_0$, $x_1$, $d_1$, $x_2$, and $d_2$. Value node $v_2$ is dependent on the nodes in $C_{Ind}(v_2)$, $C_f(v_2)$, but $v_2$ cannot be written as a function of these nodes. It is sometimes convenient to talk about the functional successors of a node x. Simply, node s is a functional successor of x if $C_f(s)$ contains x; that is, if s can be written as a function of x.

B. Separability Properties

Now, we are ready to present the properties of the expectation and maximization operators that will serve as the basis for developing reductions for influence diagrams with super value nodes. First, we have that if r is a sum super value node with only one predecessor that is a function of x, $r_0$, then taking expectation at r with respect to x is equivalent to taking expectation at $r_0$ with respect to x. Fig. 4 displays the conditions for this property. Note in the figure that since x precedes value node s and s is a functional predecessor of $r_0$, value node $r_0$ is actually a composite function of x.

Property 1: Given the value nodes $r, r_0, r_1, \ldots, r_n$ and chance node x such that r is a sum node, $C(r) = \{r_0, r_1, \ldots, r_n\}$ and $C(r) \cap S_f(x) = r_0$, applying the conditional expectation operator $E_{r_0|S_f(x)}$ at r is equivalent to applying that operator at $r_0$. 

Fig. 4. Expectation with respect to x at r is equivalent to expectation at $r_0$ (Property 1).
Proof: Applying the conditional expectation operator \( E_{n(x)} \) at value node \( r \) is equivalent to taking the expectation:

\[
E \left[ E \left[ E \left[ r \mid r_0, r_1, \ldots, r_n \right] C(x) \right] \right] = E \left[ E \left[ g \left( r_0, r_1, \ldots, r_n \right) C(x) \right] \right]
\]

because \( r \) is a sum node, \( C(r) = \{ r_0, r_1, \ldots, r_n \} \) and the linearity of the expectation operator. Finally, since \( C(r) \cap S_i(x) = r_n \) and so \( r_0, r_1, \ldots, r_{n-1} \) are constant with respect to \( x \), the right-hand side above

\[
e = r_0 + r_1 + \cdots + r_{n-1} + E \left[ r_n C(x) \right]
\]

which is equivalent to applying \( E_{n(x)} \) at \( r_n \).

There is a similar property for a product super value node.

Property 2: Given the value nodes \( r, r_0, r_1, \ldots, r_n \) and chance node \( x \) such that \( r \) is a product node, \( C(r) = \{ r_0, r_1, \ldots, r_n \} \) and \( C(r) \cap S_i(x) = r_n \), applying the conditional expectation operator \( E_{n(x)} \) at \( r \) is equivalent to applying that operator at \( r_n \).

Proof: Applying the conditional expectation operator \( E_{n(x)} \) at value node \( r \) is equivalent to taking the expectation:

\[
E \left[ E \left[ r \mid r_0, r_1, \ldots, r_n \right] C(x) \right] = E \left[ E \left[ h \left( r_0, r_1, \ldots, r_n \right) C(x) \right] \right]
\]

because \( r \) is a product node and \( C(r) = \{ r_0, r_1, \ldots, r_n \} \). Since \( C(r) \cap S_i(x) = r_n \) and so \( r_0, r_1, \ldots, r_{n-1} \) are constant with respect to \( x \),

\[
e = r_0 + r_1 + \cdots + r_{n-1} + E \left[ r_n C(x) \right]
\]

which is equivalent to applying \( E_{n(x)} \) at \( r_n \).

In other words, if \( r \) is a product super value node and only one of its value node predecessors, \( r_n \), is a function of \( x \), then taking expectation at \( r \) with respect to \( x \) is equivalent to taking expectation at \( r_n \) with respect to \( x \).

The next two properties are those of the maximization operator when applied to a sum or a product. The first states that if \( r \) is a sum super value node with value node predecessors and only one of these predecessors, \( r_n \), varies with \( d \), then maximizing \( r \) over \( d \) is equivalent to maximizing \( r_n \) with respect to \( d \). The second is the equivalent idea when \( r \) is a product node. It is slightly more restrictive in that each predecessor of \( r \) must be greater than or equal to zero for every combination of inputs. Fig. 5 illustrates the conditions for Property 3.

Property 3: Given the value nodes \( r, r_0, r_1, \ldots, r_n \) and decision node \( d \) such that \( r \) is a sum node, \( C(r) = \{ r_0, r_1, \ldots, r_n \} \) and \( S_{n(d)} \cap C(r) = r_n \), applying the maximization operator \( \text{max}_d \) at \( r \) is equivalent to applying that operator at \( r_n \).

Proof: Applying the maximization operator \( \text{max}_d \) at value node \( r \) is equivalent to taking the maximization:

\[
\text{max}_d \left[ E \left[ r \mid r_0, r_1, \ldots, r_n \right] \right] = \text{max}_d \left[ g \left( r_0, r_1, \ldots, r_n \right) \right]
\]

\[
= \max_d \left[ r_0 + r_1 + \cdots + r_n \right]
\]

\[
= \text{max}_d \left[ r_0 + r_1 + \cdots + r_{n-1} + \text{max}_d \left[ r_n \right] \right]
\]

because \( r \) is a sum node, \( C(r) = \{ r_0, r_1, \ldots, r_n \} \) and \( S_{n(d)} \cap C(r) = r_n \) and so \( r_0, r_1, \ldots, r_{n-1} \) do not vary with \( d \). The last equality is equivalent to applying \( \text{max}_d \) at \( r_n \).

Likewise:

\[
d^* = \text{arg max}_d \left[ r_n \right]
\]

Property 4: Given the value nodes \( r, r_0, r_1, \ldots, r_n \) and decision node \( d \) such that \( r \) is a product node, \( C(r) = \{ r_0, r_1, \ldots, r_n \} \), \( S_{n(d)} \cap C(r) = r_n \), and the value nodes in \( C(r) \) take on only nonnegative values, applying the maximization operator \( \text{max}_d \) at \( r \) is equivalent to applying that operator at \( r_n \).

Proof: Applying maximization operator \( \text{max}_d \) at value node \( r \) is equivalent to taking the maximization:

\[
\text{max}_d \left[ E \left[ r \mid r_0, r_1, \ldots, r_n \right] \right] = \text{max}_d \left[ h \left( r_0, r_1, \ldots, r_n \right) \right]
\]

\[
= \max_d \left[ r_0 + r_1 + \cdots + r_n \right]
\]

\[
= \text{max}_d \left[ r_0 r_1 \cdots r_{n-1} \cdot \text{max}_d \left[ r_n \right] \right]
\]

Fig. 5. Maximization with respect to \( d \) at \( r \) is equivalent to maximization at \( r_n \) (Property 3).
because \( r \) is a sum node, \( C(r) = \{r_0, r_1, \ldots, r_n\} \), and \( S_{\text{ind}}(d) \cap C(r) = r_n \) and so \( r_0, r_1, \ldots, r_{n-1} \) do not vary with \( d \). The last equality is equivalent to applying \( \max_d \) at \( r_e \). Likewise:

\[
d^* = \arg \max_{d} \{r_n\}.
\]  

(16)

Note that these properties can be applied recursively in various combinations to allow an operator to propagate through several sum or product nodes, at each step requiring an operation over a successively smaller sample space. Thus in the influence diagram in Fig. 3(c), the maximization operator over \( d \) can propagate from \( J \) to \( w \) and then to \( s \). Likewise the expectation operator over \( x \) can propagate from \( J \) to \( w \) then to \( r \). This concept will be important in allowing efficient solutions to influence diagrams that arise in applications. We formalize it in the following lemmas.

Lemma 2: If a chance node \( x \) has a single successor, value node \( r \), then applying the conditional expectation operator \( E_{x|C(x)} \) at \( J \) is equivalent to applying that operator at \( r \).

Proof: If we can prove there is a single path from \( r \) to \( J \), and that each node on that path is a sum or product node with only one predecessor that is a function of \( x \), then we can successively apply Properties 1 and 2 to obtain the desired result. By definition, node \( r \) and all of its successors are value nodes and all value nodes have a single successor which must be a super value node. This implies that all nodes that succeed \( r \) are sum or product nodes with only one predecessor that is a successor of \( r \). Since \( r \) is the only successor of \( x \), all nodes which succeed \( r \) will have only one predecessor that is a function of \( x \). Therefore, by Properties 1 and 2, applying \( E_{x|C(x)} \) at \( J \) is equivalent to applying the operator at \( r \).

Lemma 3: If a decision node \( d \) has a single successor, value node \( r \), and all value nodes take on only nonnegative values, then applying the maximization operator \( \max_d \) at \( J \) is equivalent to applying the operator at \( r \).

Proof: If we can prove there is a single path from \( r \) to \( J \) and that each node on that path is a sum or product node with only one predecessor that has \( d \) as an indirect predecessor, then we can successively apply Properties 3 and 4 to obtain the desired result. By definition, node \( r \) and all of its successors are value nodes and all value nodes have a single successor which must be a super value node. This implies that all nodes that succeed \( r \) are sum or product nodes with only one predecessor that is a successor of \( r \). Since \( r \) is the only successor of \( d \), all nodes which succeed \( r \) will have only one predecessor that is an indirect successor of \( d \). Therefore, since all value nodes take on only values greater than or equal to zero, applying the \( \max_d \) operator at \( J \) is equivalent to applying it at \( r \) by Properties 3 and 4.

With these definitions and results, we are now ready to develop more general reduction operations for the influence diagram with super value nodes. These reductions allow us to exploit the separable nature of the value function in solving decision problems in the influence diagram framework.

IV. Reductions

As discussed in Section II, in order to analyze the influence diagram of a decision problem, four reductions are required: arc reversal, chance node removal by summarization over the joint, chance node removal by expectation of the value function, and decision node removal by maximization of the value function. The first two of these do not involve value nodes and so are unaffected by replacing the single value node with a set of value nodes. The last two clearly involve the value nodes. The purpose of this section is to develop maximization and expectation reductions for influence diagrams with multiple value nodes.

A. Expectation and Maximization

We first state a useful form of expectation expansion and then use this to prove the basic theorem for removing chance nodes from an influence diagram with super value nodes. We then prove a more operational version of the theorem.

Lemma 4: For any \( n, m \geq 1 \), and random variables \( Y, x_1, x_2, \ldots, x_{n+m} \):

\[
E \left[ E \left[ E \left[ x_1, x_2, \ldots, x_{n+m} \right] \mid x_1, x_2, \ldots, x_n \right] \right] = E \left[ E \left[ x_1, x_2, \ldots, x_m \right] \mid x_1, x_2, \ldots, x_n \right].
\]  

(17)

This lemma is a special case of a theorem proved in Billingsly [3]. We use this expectation expansion result to prove the basic chance node removal theorem.

Theorem 2 (Simple Chance Node Removal): If \( x \) is a chance node in an influence diagram with terminal value node \( J \), and \( x \) directly precedes a value node \( r \) and nothing else, chance node \( x \) may be removed by conditional expectation over \( r \). Value node \( r \) inherits the conditional predecessors of \( x \) and \( x \) is removed from the diagram.

Proof: Taking expectation of \( J \) with respect to \( x \) is equivalent to applying the \( E_{x|C(x)} \) operator to \( J \). This operator propagates to \( r \) by Lemma 2 since \( x \) has a single successor, value node \( r \). Applying the \( E_{x|C(x)} \) operator to \( r \) is equivalent to taking the expectation:

\[
E \left[ E \left[ r \mid C(r) \right] \mid C(x) \right] = \sum_x E \left[ r \mid C(r) \right] \cdot P(x \mid C(x)) \]

\[
= \sum_x E \left[ r \mid C(r) \cup C(x) \right] \cdot P(x \mid C(x)) \]

\[
= \sum_x E \left[ r \mid C(r) \cup C(x) \right] \cdot P(x \mid C(r) \cup C(x) \\setminus \{x\}) \]

\[
= \sum_x E \left[ r \mid C(r) \cup C(x) \right] \cdot P(x \mid \{C(r) \cup C(x) \\setminus \{x\}) \]

(18)
which is in a form suitable for application of Lemma 4 and we have:

\[ E \left[ E \left[ r \mid C(r) \right] \mid C(x) \right] = E \left[ r \left\mid (C(r) \cup C(x)) \setminus \{x\} \right. \right]. \]

(19)

The second and third equalities hold because both \( r \) and \( x \) are independent of any other nodes in the influence diagram given their predecessors.

As an example of the use of this theorem consider the influence diagram in Fig. 3(c). A valid reduction of this influence diagram, per Theorem 2, is to take expectation of \( r \) with respect to \( x \).

A chance node can be removed from an influence diagram under more general conditions than those in Theorem 2, but the removal process is more complex. The following theorem allows us to remove a chance node whenever it does not have any decision node successors, that is, has only chance and value node successors. We first prove a lemma stating that whenever a chance node has multiple value node successors these can always be reduced into a single value node successor of \( x \).

For this lemma consider Fig. 6. The lemma merely states that since node \( d \) has value node successors \( r \) and \( s \) we can take value nodes \( r \) and \( x \) and remove them into \( w \) resulting in an influence diagram in which \( d \) has only one value node successor. We are just rewriting \( w \), a composite function of \( d \) as a function of \( d' \).

**Lemma 5:** If a node \( x \) in an influence diagram has multiple value node successors then there exists a value node \( r \) such that all paths from \( x \) to terminal value node \( J \) contain \( r \). The set of all value node predecessors of \( r \), may be reduced into \( r \) resulting in \( x \) having a single successor, value node \( r \).

**Proof:** We show that \( J \) is such a node. There are no barren nodes in the influence diagram and so all directed paths from \( x \) contain \( J \). All value node predecessors of \( J \) can be removed by Theorem 1.

It will usually not be advantageous to reduce all value node successors of such a chance node \( x \) into \( J \) but rather into some other value node instead. We are now ready to state the more general chance node removal theorem.

**Theorem 3 (Chance Node Removal):** If \( x \) is a chance node in an influence diagram with no decision successors, then \( x \) can be removed from the diagram.

**Proof:** Since \( x \) has no decision node successors, it has only chance and value node successors. The arcs from \( x \) to its chance node successors can all be reversed [16]. The node \( x \) will then have only value node successors. We can reduce the value node successors of \( x \) such that \( x \) has a single value node successor by the preceding lemma. The requirements for Theorem 2 are now met and \( x \) may be removed.

These two theorems provide us with the theory we need to remove chance nodes from influence diagrams with super value nodes under very general conditions. In order to solve decision problems we also need to perform maximizations over the value function; that is, we need a theory for removing decision nodes. In parallel with the development of the chance node removal theorems, we first present an elementary decision node removal theorem and then a more useful version with more general conditions.

**Theorem 4 (Simple Decision Node Removal):** If \( d \) is a decision node in an influence diagram with terminal value node \( J \) such that:

a) all value nodes take on only nonnegative values;
b) \( d \) directly precedes a value node \( r \) and nothing else;
c) all conditional predecessors of \( r \), besides \( d \), are informational predecessors of \( d \); that is, \( C(r) \setminus \{d\} \) is contained in \( I(d) \)

then \( d \) may be removed by maximization of \( r \) with respect to \( d \). The new predecessors of \( r \) will be \( C(r) \setminus \{d\} \).

**Proof:** Maximization of \( J \) with respect to \( d \) is equivalent to applying the maximization operator \( \max_{d'} \) to \( J \). This operator propagates to \( r \) by Lemma 3 since \( d \) has a single successor, value node \( r \). Applying the \( \max_{d} \) operator to \( r \) is equivalent to performing the maximization:

\[
\max_{d} E \left[ r \mid C(r) \right] = E \left[ r \mid C(r) \setminus \{d\} \right], \quad (20)
\]

for all settings of the variables in \( C(r) \setminus \{d\} \). [21]

Note that any direct predecessors of \( d \) that are not also direct predecessors of \( r \) are not involved in the maximization operation. Arcs from these nodes can be ignored in the maximization.

In Fig. 3(c), decision node \( d \) can be removed by maximizing \( s \) over \( d \) using this theorem. On the other hand, if \( d \) has many value node successors we would like to know...
if it is removable (value nodes will have to be removed first so that the conditions of the previous theorem hold).

**Theorem 5 (Decision Node Removal):** If \( d \) is a decision node in an influence diagram such that:

a) all value nodes take on only nonnegative values;

b) each successor of \( d \) is a value node;

c) there is a value node \( s \) such that \( C(s) \) is contained in \( I(d) \cup \{ d \} \);

d) all directed paths from \( d \) to terminal value node \( J \) contain \( s \) then \( d \) can be removed from the diagram by first removing \( C_{\text{ind}}(s) \cap V \) and then maximizing \( s \) over \( d \).

**Proof:** \( C_{\text{ind}}(s) \cap V \) can be removed by Theorem 1. After this reduction all conditions for Theorem 4 are satisfied and \( d \) can be removed from the diagram by maximizing \( s \) over \( d \).

Using this theorem, decision node \( d \) in Fig. 6 can be removed by maximizing over value node \( w \) after value nodes \( r \) and \( s \) have been removed into \( w \).

These four theorems give us the reductions necessary to solve decision problems expressed as influence diagrams with super value nodes. We see that there are times when it is necessary to reduce a portion of the multiple value node structure as in Theorems 3 and 5. In this case, we would like to retain as much of this structure as possible for as long as possible.

**B. Value Node Merging**

The Properties 1 through 4 in Section III all assumed that only one out of \( n \) value node predecessors of a super value node was dependent on a particular chance or decision node. Suppose that instead two out of \( n \) predecessors were dependent on a particular chance or decision node. We wish to combine the two predecessors into a single value node and then apply the property.

As an example, look ahead to the influence diagram in Fig. 8(b). Decision node \( d_2 \) cannot be removed because it has two successors, value nodes \( r_2 \) and \( \nu_3 \). Rather than reduce all value node predecessors into \( J \) to provide a single successor for \( d_2 \), we introduce new super value node \( \nu_2 \) as in Fig. 8(c) and then apply Theorem 5 which will reduce \( r_2 \) and \( \nu_3 \) into \( \nu_2 \). It is simple for the software implementing the algorithm to introduce such nodes as \( \nu_2 \) without assistance from the user. This concept is formalized in the following theorem. We think of this reduction as merging the two value nodes \( r_2 \) and \( \nu_3 \) into the new super value node, \( \nu_2 \), which represents a partial sum or partial product.

**Theorem 6 (Merging of Value Nodes):** Given the value nodes \( r, r_0, r_1, \ldots, r_m \) such that \( r \) is a sum or product node and \( C(r) = \{ r_1, r_2, \ldots, r_m, r_{m+1}, \ldots, r_n \} \) then the value nodes \( r_1, r_2, \ldots, r_m \) may be merged into a new value node \( s \) of the same type (sum or product) as \( r \). This results in \( C(s) = C(r_1) \cup C(r_2) \cup \cdots \cup C(r_m) \). Node \( r \) remains the same type and \( C(r) \) becomes \( \{ s, r_{m+1}, \ldots, r_n \} \).

**Proof:**

\[
\begin{align*}
  r &= r_1 + r_2 + \cdots + r_m + r_{m+1} + \cdots + r_n \\
  &= s + r_{m+1} + \cdots + r_n
\end{align*}
\]

where

\[
s = r_1 + r_2 + \cdots + r_m
\]

and

\[
\begin{align*}
  r &= r_1 \cdot r_2 \cdot \cdots \cdot r_n \\
  &= s \cdot r_{m+1} \cdot \cdots \cdot r_n
\end{align*}
\]

where

\[
s = r_1 \cdot r_2 \cdot \cdots \cdot r_m.
\]

Using this theorem, we can take maximum advantage of the separability of the value function represented by the super value node structure, reducing this structure only as necessary.

We now have all reductions needed to solve influence diagrams with value nodes. We next discuss how to combine these reductions in an algorithm to solve such influence diagrams.

**V. ALGORITHM**

In this section we present an algorithm that is able to reduce any influence diagram to provide the maximum expected value and optimal policy of the decision problem.

**A. Reducing Value Node Structure**

In influence diagrams, a decision problem is solved by reducing all nodes from the influence diagram except the terminal value node while recording the optimal policy choices as decision nodes are removed. The computational complexity of solving the influence diagram can be critically dependent on the order in which the nodes are removed. At any step, there may be many valid reductions possible. We will present some heuristics to guide the choice of the next reduction, but there may be a more efficient sequence of reductions [4], [14]. We offer the following two heuristics.

First, if two value nodes \( r_1 \) and \( r_2 \) have the same successor, a super value node \( r \), and \( C(r_1) \) is contained in \( C(r_2) \), then removing \( r_1 \) and \( r_2 \) (if they are the only predecessors of \( r \), or merging \( r_1 \) and \( r_2 \) into new value node \( r \) if they are not) will not increase the size of any operation necessary to solve the influence diagram and so we should remove them. We call this the *subset rule*. In Fig. 7 value nodes \( u \) and \( w \) would be removed into \( r \) by this rule. The rationale here is the following. Take any \( y \) which is an element of both \( C(u) \) and \( C(w) \). Node \( y \) cannot be removed until \( u \) and \( w \) are merged into a single value node. Now, take node \( z \) that is an element of \( C(u) \) alone. The size of the operation necessary to remove \( z \) is dependent on \( C(z) \) and \( C(u) \). If we merge \( u \) and \( w \) the predecessors of the new value node would be \( C(u) \cup C(w) \).
Fig. 7. Value nodes \( u \) and \( w \) would be reduced into \( r \) by subset rule.

\[ = C(u) \text{ because } C(w) \text{ is contained in } C(u) \] and so the size of the operation required to remove \( z \) would be the same.

Second, when a decision node \( d \) satisfies the conditions of Theorem 5, that is, \( d \) is removable by reducing the value node predecessors of some value node \( s \) and then reducing \( d \) into \( s \), then it should be removed along with the relevant value nodes. The rationale here is that if \( d \) is removable then there is nothing we can do to change the nature of the operation required to remove \( d \). The very conditions that make \( d \) removable prevent the removal of any node which decreases the dimensionality of the operation to remove \( d \). We can change neither the size nor the scope of this operation. Also, value nodes do not inherit the predecessors of decision nodes in the maximization operation, and so removing a decision node will not increase the size of any subsequent operations. For these reasons, if decision node \( d \) is removable, we remove it. An example is Fig. 6. Decision node \( d \) is removable since value nodes \( r \) and \( s \) can be reduced into \( w \) leaving \( d \) with a single successor. The only other predecessor of that value node successor is chance node \( y \) which is also a predecessor of \( d \). The size of the operation to remove \( d \) is \( |\Omega_y - \Omega_y| \). This cannot be reduced because \( d \) must be removed before \( y \).

These heuristics do not completely solve the problem of when to remove value nodes. They are sufficient, however, to allow us to solve influence diagrams with multiple value nodes in practice.

B. Algorithm

The algorithm is now presented. It will be shown that the algorithm always reduces an influence diagram to the terminal value node thus producing the optimal policy and maximum expected value for the problem.

```
DEFINE PROCEDURE EVALUATE_ID
    BEGIN
        add "no forgetting" arcs
        eliminate any barren nodes
        WHILE \( C(J) \neq \emptyset \) DO
            IF subset rule holds for any set of value nodes
                THEN remove these value nodes
                    (possibly adding a sum or product node)
            ELSE
                END IF
        END WHILE
    END EVALUATE_ID
```

We begin the iterative part of the algorithm with an influence diagram that has only a finite number of nodes. After each step of the algorithm, the net change in total number of nodes in the diagram will be at least one less. Thus the algorithm is guaranteed to reduce the influence diagram to the value node and thus provide the solution if there is always at least one removable node at each step.

Consider first the case in which all chance and decision nodes have been removed from the diagram. The influence diagram then contains only value nodes. As stated in Theorem 1 these can always be removed.

If there are chance or decision nodes in the diagram, then the following theorem guarantees that at least one of them is removable.

**Theorem 6 (Existence of a Node to Remove):** In an influence diagram there is always a removable node until only a single value node remains.

**Proof:** From Theorem 1 the influence diagram with multiple value nodes can always be reduced to an influence diagram with a single value node. In such a diagram there is always a node to remove until only the value node remains [16].

The algorithm concludes our development of the influence diagram theory necessary to formulate and analyze decision problems with separable value functions. We have introduced super value nodes to represent the separable nature of the value function. We have developed reductions and an algorithm for influence diagrams with multiple value nodes to allow us to solve such decision problems within the influence diagram framework. We are now prepared to consider applications of this theory.

VI. EXAMPLES

The influence diagram with multiple value nodes has been applied to several sample problems including the standard Markov decision process (MDP), an MDP with time lag, a partially observable MDP, an inventory prob-
Fig. 8. Solving a three stage MDP. (a) The initial influence diagram. (b) After expectation of \( r_3 \) with respect to \( x_3 \). (c) After introducing new value node \( v_2 \). (d) After summing \( r_2 \) and \( v_2 \) into \( r_2 \). (e) After maximization of \( v_2 \) with respect to \( d_2 \).

A. Standard Markov Decision Process

The graph of the influence diagram of a four stage MDP is shown in Fig. 8(a). All of the critical characteristics of the MDP are easily identified in the influence diagram. The state variable is represented by the chance node \( x_i \). The state at each stage depends only on the previous state and the previous decision (\( x_i \) has lone predecessors \( x_{i-1} \) and \( d_{i-1} \)). The decisionmaker at each stage knows the current value of the state (the arc from \( x_i \) to \( d_i \)). Also, at each stage there is a reward as a function of the current state and current decision (the \( r_i \) value node at each stage, each with predecessors \( x_i \) and \( d_i \)). There is a salvage value that is a function of the final state alone (value node \( v_3 \) with lone predecessor \( x_3 \)). The value function \( J \) is the sum of the stage rewards and the salvage value (sum node \( J \) with predecessors \( r_3 \) through \( r_3 \) and \( v_3 \)).

Remember that each value node represents a conditional expected value and that conditional expectations can be written as a function of the conditioning variables. For example, each value node \( r_j \) in Fig. 8(a) represents

\[
E[r_j|x_i,d_i] = g(x_i,d_i)
\]

for some function \( g \).

Now, consider solving this MDP in the influence diagram framework. There is no removable decision and the subset rule does not apply; therefore, the algorithm will choose a chance node to remove. The only removable chance node is \( x_3 \) which is removed producing the influence diagram in Fig. 8(b). This corresponds in the mathematics to taking expectation of \( v_3 \) with respect to \( x_3 \) conditioned on \( x_2 \) and \( d_2 \):

\[
E[v_3|x_2,d_2] = E[E[v_3|x_3]|x_2,d_2].
\] (25)

Now, the subset rule holds for \( r_2 \) and \( v_3 \), so they will be removed by introducing value node \( v_3 \) as in Fig. 8(c) and then merging \( r_2 \) and \( v_3 \) into \( r_2 \). Note that no additional chance or decision nodes are removable until \( r_2 \) and \( v_3 \) are merged. This yields the influence diagram in Fig. 8(d). In the mathematics we have

\[
E[v_2|x_2,d_2] = E[r_2|x_2,d_2] + E[v_3|x_2,d_2].
\] (26)

Back in the influence diagram, we see that there is a removable decision \( d_2 \) which is removed to produce the influence diagram in Fig. 8(e). In the mathematics this corresponds to

\[
E[v_2|x_2] = \max_{d_2} E[v_2|x_2,d_2].
\] (27)

This influence diagram is identical to that in Fig. 8(a) with one less stage. Thus the above steps may be repeated to remove each stage of the MDP. Note that the "no forgetting" arcs were not added. They were unnecessary in removing \( d_2 \) because \( d_2 \) was removed by maximizing over
Fig. 9. Formulation of the MDP per Bertsekas [2].

\[ E[v_2|x_2] = \max_{d_2} \left( E[v_3|x_2, d_2] + E[v_3|x_3|x_2, d_2] \right). \]

Note that this is precisely the recurrence relation of stochastic dynamic programming as stated in Howard [5].

Another version of the basic stochastic dynamic programming problem is found in Bertsekas [2]. The influence diagram for this formulation is in Fig. 9. The state variable in this case is deterministic (as indicated by the double circle in the influence diagram) and uncertainty is modeled with the noise variables \( w_i \). This problem can also be solved in the influence diagram framework as presented here; however, it would require the introduction of “deterministic nodes” that are drawn as double circles [17].

The recurrence relations for this formulation would be the following:

\[ E[v_2|x_2] = \max_{d_2} E \left( E[v_3|x_2, d_2, w_2] + E[v_3|x_2, d_2, w_2] \right). \]

For the standard MDP, as in Howard [5], the influence diagram steps required to solve it correspond exactly to the dynamic programming operations. The influence diagram algorithm is performing dynamic programming, taking advantage of the special structure of the problem while the user needed only to represent the problem as an influence diagram and explicitly represent the sums and products in the value function by use of super value nodes. The user did not need to set up the recursive equations of dynamic programming nor the expectation variables \( v_i \). Indeed, software has been written that constructs the super value node structure for the user by parsing the user’s value function stored in the value node of a single value node influence diagram [1]. The value function may be stored in standard Lotus 123 notation. The software then automatically solves the resulting MDP.

B. MDP with Time Lag

The super value node influence diagram algorithm is robust with regard to problem structure. As an example of this point, consider the influence diagram in Fig. 10. This is a common deviation from the standard MDP in which the next state depends not only on the current state and decision, but also on some subset of previous states and decisions. This common variation of the problem could be dealt with using standard dynamic programming techniques by defining an augmented state variable as in

Fig. 10. Solving three stage MDP with time lag. (a) The initial influence diagram. (b) After expectation of \( v_3 \) with respect to \( x_3 \). (c) After summing \( r_1, r_2, \) and \( r_3 \) into \( r_2 \). (d) After maximization of \( v_2 \) with respect to \( d_2 \).
Bertsekas [2]. For the case in Fig. 10 the augmented state variable would be the vector \( y_k = [x_k, x_{k-1}, d_{k-1}] \). The decision process with this new state variable would be similar to the standard MDP in Fig. 8(a). Solving this influence diagram is equivalent to solving the problem by traditional dynamic programming. Using state augmentation means forcing the problem into the standard MDP format.

On the other hand, the problem can be solved directly as originally formulated in the influence diagram by applying the techniques presented here. The first few steps of the solution process are shown in Fig. 10. Reformulating the problem to one with an augmented state variable is unnecessary. This is advantageous in that it saves the user the work of reformulation and the original structure of the problem is maintained.

Another important advantage of solving this problem directly as an influence diagram is that doing so significantly decreases the data storage requirements and computational complexity. In the current example, let each \( d_k \) have \( n \) alternatives and each \( x_k \) have \( n \) outcomes. Then the augmented state variable, \( y_k = [x_{k-1}, d_{k-1}, x_k] \), has \( n^3 \) outcomes. Thus since \( y_k \) is conditioned on \( y_{k-1} \) and \( d_{k-1} \), we must store \( n \cdot n^3 \cdot n^3 = n^7 \) probabilities for \( y_k \). If the problem is formulated as an influence diagram as in Fig. 10, each \( x_k \) is conditioned on \( x_{k-1}, x_{k-2}, d_{k-1}, \) and \( d_{k-2} \). Only \( n \cdot n \cdot n = n^3 \) probabilities must be stored. Likewise, to roll back each stage of the decision process with the augmented state variable requires an operation of order \( n^7 \). The largest operation necessary in order to solve the problem directly as an influence diagram is of order \( n^3 \).

To understand this advantage of the influence diagram, consider the following statements

\[
E[v_3|y_2, d_2] = E[E[v_3|y_3]|y_2, d_2] \tag{30}
\]

\[
E[v_3|x_1, d_1, x_2] = E[v_3|x_2, x_3] \tag{31}
\]

\[
E[v_3|x_1, d_1, x_2] = E[v_3|x_3] \tag{32}
\]

Statements (30) and (31) illustrate taking the first expectation in the solution procedure using the augmented state variable approach. They are equivalent due to the definition of the augmented state variable, \( y_k = [x_{k-1}, d_{k-1}, x_k] \). Statement (32) represents the first expectation in the solution process using the influence diagram. Comparing (31) and (32) illustrates the key distinction between the augmented state variable approach and the influence diagram approach. This distinction is that in the augmented state variable approach we must treat the variables \( [x_{k-1}, d_{k-1}, x_k] \) as a group. In the influence diagram technique we can treat these variables separately. For example, in (31), the \( x_2 \) in \( [x_2, d_2, x_3] \) and the \( x_3 \) in \( [x_1, d_1, x_2] \) are the same variable obviously but count as two variables because we cannot “look inside” the augmented stated variable.

The influence diagram, on the other hand, allows us to treat each variable individually.

Thus for this problem the influence diagram is quite valuable. By automating the process not only is the user saved the work of reformulation, but the user does not even need to know how to reformulate. It allows the problem to be solved in a framework that captures the structure of the original problem. Finally, it reduces the number of probabilities stored for each state variable and reduces the size of the largest operation in the solution procedure from \( n^7 \) to \( n^3 \). Of course, these same efficiencies could theoretically be obtained by solving the problem algebraically without benefit of the influence diagram. However, in practice, this would require solving on a case by case basis problems that do not fit into the standard MDP format without the benefit of general tools. Instead, the influence diagram provides a framework for automating this process and thus routinely delivers the efficiencies discussed above when the structure of the problem permits.

### C. Satellite Power Decision Analysis

A final example involves research and development decisions for a new class of satellite power generators. The influence diagram for the resulting pilot decision model is shown in Fig. 11. The variable definitions for the various nodes are not important, only the basic structure of the problem. Note that node Value is a sum super value node and that it is the sum of five value nodes representing the various costs and benefits involved in the decision. There is the common development-production sequence of decisions, but the model would not be classified as a decision process.

In solving this model, the analysts came across a serious size problem. Solving for the optimal policy required an operation of size \( 3^{30} \) in terms of both number of operations and array size. This prevented them from solving the model with the influence diagram software available. At this point it was noticed that the value function had a separable structure and the influence diagram in Fig. 11 was developed using the supervalue node ideas. The largest operation and the largest array required to solve the influence diagram with super value nodes was decreased from...
3^{10} \text{ to } 3^5. \text{ This reduction in array size and number of operations allowed the model to be solved easily along with sensitivity analyses and value of information calculations. The total number of operations required to solve the model was reduced from } 1.3 \times 10^3 \text{ to } 1.2 \times 10^3.

VII. CONCLUSION

Super value nodes appear to be a natural and effective technique for representing the separable nature of a value function in the influence diagram structure. The influence diagram with this added construct provides a language for representing decision problems with separable value functions that emphasizes the critical aspects of these problems. These aspects can be exploited in an algorithm which analyzes the problem. The algorithm presented here is able to recognize the opportunity for applying dynamic programming and to use dynamic programming in solving the decision problem to the extent possible given problem structure.

Extending the influence diagram framework to incorporate dynamic programming has many advantages. In order to solve the problem by dynamic programming, the user need only represent the separable nature of the value function. It is not necessary to set up the traditional recursive equations of dynamic programming. The explicit representation of the conditional independencies in the problem by the influence diagram allows the algorithm to exploit this, as well. This can decrease data storage and computational complexity by orders of magnitude for certain Markov decision processes. Finally, many of the critical characteristics of various decision processes are made explicit by representing the decision process as an influence diagram.

The concept of a super value node is both consistent with previous influence diagram theory and expands the application potential of that theory to include a rather large and important class of decision problems, namely decision processes.

ACKNOWLEDGMENT

The authors wish to thank Ron Howard and Jim Matheson for many helpful discussions. Also, the comments and questions of two anonymous reviewers led to important improvements in the paper.

REFERENCES


Joseph A. Taitman was born in Indianapolis, IN. He received the B.S. degree in electrical engineering from the University of Notre Dame in 1978; the M.S. degree from the Air Force Institute of Technology, Wright-Patterson AFB, Ohio, in 1979 and the Ph.D. degree in engineering-economic systems from Stanford University in 1986. He is currently on the faculty of the Department of Mathematics and Computer Science at the Air Force Institute of Technology. He has developed a decision analysis program there and leads research and applications in the analysis of defense decisions.

Dr. Taitman is a member of the Operations Research Society of America, the Military Operations Research Society and Tau Beta Pi. He was named the Tau Beta Pi Outstanding Teacher of 1988 for the Air Force Institute of Technology.

Rox D. Shachter received an S.B. from the Massachusetts Institute of Technology in 1976 and the M.S. and Ph.D. degrees in operations research from the University of California, Berkeley, in 1979 and 1982. Since 1982 he has been an Assistant Professor in the Department of Engineering-Economic Systems at Stanford University. His research has been in decisionmaking under uncertainty, with emphasis on medical decisionmaking and on the representation and analysis of decision models with influence diagrams. During the academic years 1986-1988, he was a Visiting Professor at the Center for Health Policy Research and Education at Duke University, where he developed interactive analytical tools to assist in medical technology assessment.

Dr. Shachter is an active participant and organizer of the Workshops on Uncertainty in Artificial Intelligence, and an officer of the Operations Research Society's Special Interest Group on Decision Analysis. He is a member of ORSA, TIMS, AAAI, and the Society for Medical Decision Making.