A Factorization Approach To Evaluating Simultaneous Influence Diagrams

Weihong Zhang and Qiang Ji *Senior Member, IEEE*

***Abstract*— Evaluating an influence diagram (ID) is a challenging problem because its complexity increases exponentially in the number of decision nodes in the diagram. In this paper, we examine the problem for a special class of IDs where multiple decisions must be made simultaneously. We describe a brief theory that factorizes out the computations common to all policies in evaluating them. Our evaluation approach conducts these computations once and uses them across all policies. We identify the ID structures for which the approach can achieve savings. We show that the approach can be used to efficiently re-compute the optimal policy of an ID when its structure or parameters change. Finally, we demonstrate the superior performance of the approach by simulation studies and a military planning example.**

***Index Terms*— Decision making under uncertainty, graphical model, influence diagram, algorithm, military analysis.**

I. Introduction

An Influence diagram (ID) is a plausible graphical model for decision making under uncertainty [1]. An ID comprises of decision nodes, random nodes, value nodes and the probabilistic relations among these nodes. An ID is a more compact representation of a decision tree, which is a simple tool for decision analysis [2].

Given an ID, a policy prescribes an action choice for each decision node. Evaluating a policy is to compute the expected value of the ID under the policy. Evaluating an ID is to find the optimal policy that maximizes the expected value of the ID. A generic approach to evaluating an ID has to enumerate all policies, compare the expected utilities under them and choose the optimal one. However, the number of policies grows exponentially with the number of decision nodes. This renders the approaches for general ID evaluation very inefficient and infeasible for large IDs. Consequently, it is advisable to study efficient algorithms for special IDs.

Most of the previous approaches assume that there exists a linear ordering among the decision nodes. This ordering implies that the choice of a decision node is known to the decision maker when he/she chooses the actions for the successive decision nodes (e.g., see Zhang 1998). For a decision node, this linear ordering usually can be exploited to decompose the ID into one fraction prior to the node and the other fraction posterior to the node. The choice for the decision node can be made using the fraction posterior to the node. The procedure repeats for each decision node.

Manuscript received January 20, 2002; revised November 18, 2002.

Weihong Zhang and Qiang Ji are with Department of Electrical, Computer and Systems Engineering, Rensselaer Polytechnic Institute, Troy, NY 12180-3590, USA.

In this paper, we examine the ID evaluation problem for a special class of IDs in which decision nodes have no parents. Essentially an ID with this property assumes no precedence relationship among decision nodes. In other words, one has to determine the choices for all decision nodes simultaneously. For this reason, such an ID is said to be *simultaneous*. The simultaneity assumption prevails in real-world problem domains. For instance, a military planner must select among a number of available actions to achieve his/her overall goal success; a business owner must consider multiple elements in order to maximize his/her monetary profit.

In evaluating a simultaneous ID, we exploit the assumption and divide the ID into two fractions, calling them *the up-stream* and *down-stream*. Roughly, the up-stream consists of decision nodes and their children nodes through which the decisions propagate their impacts on the ID. Informally these child nodes are called *interface nodes*. The down-stream consists of the interface nodes and their succeeding nodes. We present a *representation theorem*, showing that the expected value of a value node under a policy can be represented as the sum of some intermediate quantities weighted by the probabilities determined by the policy. These intermediate quantities involve only the down-stream. The factorization approach we proposed computes them once but uses them across all policies. The computational gain brought by the approach depends on the size of the down-stream. Usually larger down-stream size implies more savings.

We organize the paper as follows. In the next section, we discuss related work to this research. We then introduce influence diagrams and the evaluation problem. In Section IV, we describe the representation theorem and develop the factorization approach. In Section V, we discuss two extensions of the approach: how it can be adapted to network structure/parameter changes and how it can be used in planning over time. We report empirical results on simulation studies and a military planning example in Section VI. Finally, we conclude the paper in Section VII.

II. Related Work

Since IDs were introduced by Howard and Matheson (1984), a variety of approaches have been proposed to find the optimal policy of a given ID. To mitigate the exponential growth problem of the policy number in the number of decision nodes, researchers have studied several special ID classes and proposed efficient approaches exploiting their specific problem characteristics. We give a brief survey of these IDs and their solutions.

A. Regular and No-forgetting IDs

To some extent, most IDs that have been studied assume a precedence ordering of the decision nodes. A regular ID assumes that there is a directed path containing all decision nodes; a no-forgetting ID assumes that each decision node and its parents are also parents of the successive decision nodes; and a stepwise decomposable ID assumes that the parents of each decision node divide the ID into two separate fractions. These assumptions are different from ours, which requires the actions to be chosen simultaneously. There exist *direct* and *indirect* approaches evaluating a regular no-forgetting ID. A direct approach works on the ID and evaluates it directly. Shachter (1986) proposed an algorithm that evaluates an ID by applying a series of value-preserving reductions. A value-preserving reduction is an operation that can transform an ID into another one with the same expected value. Specifically, Shachter identified four reductions – arc reversal, barren node removal, random node removal and decision node removal. An indirect approach first transforms an ID into an intermediate structure whose optimal policy (or value) remains the same as in the original ID. It then evaluates the intermediate structure and obtains the optimal policy. For instance, Howard and Matheson discussed a way to transform an ID into a decision tree network and to compute an optimal policy from the decision tree. In transforming an ID into a decision tree network, a basic operation is arc reversal [1], [3]. Since an no-forgetting ID must be stepwise decomposable, stepwise decomposability is more general than no-forgetting.

In most ID evaluation approaches, the ordering of decision nodes is an important information source in decision making and therefore is exploited to evaluate the optimal decision for decision nodes [4], [5], [6]. A stepwise decomposable ID can be evaluated by a divide-and-conquer approach. The approach deals with one decision node at a time [7]. For each decision node, its parental set separates an ID into two parts – a *body* and a *tail*. The tail is a simple ID with only one decision node. The body's value node is a new value node whose value function is obtained by evaluating the tail. In evaluating a stepwise decomposable ID, the approach begins with a leaf decision node and repeats the decomposition / evaluation procedure for the preceding decision nodes. In evaluating the tail with only one single decision node, the problem is reduced to that of computing posterior probabilities in a Bayesian network. Hence the approach uses probabilistic inference techniques to evaluate an ID. Cooper (1988) initiated the research in this direction. He gave a recursive formula for computing the maximal expected utilities and optimal policies of IDs. Shachter and Peot (1992) showed that the problem of ID evaluation can be reduced to a series of probabilistic inferences. Zhang (1998) described an algorithm that induces much easier probabilistic inferences than those in [8], [9].

B. Partial IDs

There also exists research work that relaxed the regularity or no-forgetting assumption. The specific ID types include partial IDs, unconstrained IDs and LIMID which is a compact representation of IDs. A partial ID is an ID that allows a non-total ordering of decision nodes [10]. Because the solution to a partial ID depends on the temporal ordering of the decisions, it is of interest to find the conditions identifying a class of partial IDs whose solution is independent of the legal evaluation ordering. Based on the concept of d-connectivity, Nielsen and Jensen presented an algorithm determining whether or not a partial ID represents well-defined scenarios, and they also addressed the problem of whether all admissible orderings yield the same optimal strategy.

An unconstrained ID is an ID where the order of decision nodes and the observable random nodes is not determined [11]. For an unconstrained ID, it is of interest to determine the order of decision nodes and information on which set of nodes is necessary for decision making in a decision node. For this purpose, a set of rules have been developed in order to determine the choice of the next decision node given the current information. Such a decision choice may be dependent on the specific information from the past.

Another recently proposed ID is called *LImited Memory Influence Diagram*(LIMID) that violates the no-forgetting assumption [12]. In contrast to the regular and non-forgetting assumption, the assumption behind a LIMID is that only requisite information for the computation of optimal policies is depicted in the graphical representation. Two properties pertaining to LIMIDs are (1) any ID can be converted to a LIMID; (2) the converted LIMID is more compact than the original ID in the sense that only requisite information is depicted in the LIMID for computing an optimal policy. By these properties, one may convert an ID to its LIMID version and solve the LIMID instead of the original ID. This optimal policy is also optimal in the original ID. The algorithm solving a LIMID exploits the fact that the entire decision problem can be partitioned into a set of smaller decision problems, each of which has one decision node only. This is analogous to the divide-and-conquer approach [13].

C. Simultaneous IDs

From its root definition, an ID does not impose a precedence ordering of the decision nodes. As an example, there are military applications that need to choose multiple actions simultaneously. A simultaneous ID is suitable for this situation. We exploit this assumption and divide a simultaneous ID into the up-stream and the down-stream fractions. The decomposition takes the random and value nodes as interface nodes between the up-stream and the down-stream. The computations involving the down-stream fraction can be pre-computed and re-used across all policies in evaluating them. This computation sharing schema can greatly accelerate the procedure of finding the optimal policy for a given ID, as indicated in our theoretical and empirical analysis.

Technically, the factorization approach has some conceptual similarities to the probabilistic inference-based algorithm [13]. Both algorithms divide the ID into two fractions. However, there are apparent differences. In the cited article, the separation of an ID relies on a single decision node. With respect to a decision node, roughly the body contains the predecessors

of the decision nodes, while the tail contains the successors. The choice of the decision node is evaluated by the tail part. This is quite different from our factorization approach, where the separation relies on the set of interface nodes. The set of the interface nodes separates an ID into two fractions: roughly the up-stream contains the predecessors of all interface nodes, while the down-stream contains the successors. This difference in solution techniques stems from the difference in assumption – the probabilistic inference algorithm works with a regular ID that specifies a linear order among decision nodes, whereas the factorization algorithm works with a simultaneous ID that assumes no ordering among decision nodes.

III. Influence Diagram

Mathematically, an ID $\mathcal{I}$ is a directed acyclic graph consisting of three types of nodes and the links among these nodes [1].

- Its node set is partitioned into a set of random nodes $\mathcal{Y}$, a set of decision nodes $\mathcal{X}$ and a set of value nodes $\mathcal{U}$. A value node cannot have children. The links characterize the conditional dependence among the nodes in the ID. Specifically, links to a random node indicate the probabilistic dependence of the node on its parents; links to a decision node indicate the information available to the planner at the time the planner must choose a decision for it; links to a value node indicate the functional dependencies.
  We will adopt the following notational conventions. We will use bold-typed letters such as $\mathcal{Z}$ to denote a set of variables and capital letters such as Z to denote a variable in the set. Each random or decision node Z is associated with a set Ω_Z, denoting the set of its possible states. The set Ω_Z is called *the domain* of node Z. An element in Ω_Z is denoted by a low-case letter z. For any node Z, we use $\pi(Z)$ to denote its parent set. For any subset $\mathcal{Z}' \subset \mathcal{Y} \cup \mathcal{X}$, we use $\Omega_{\mathcal{Z}'}$ to denote the Cartesian product $\Pi_{Z \in \mathcal{Z}'} \Omega_Z$. For convenience, we shall interchangeably use a *node* and a *variable*. Without loss of generality, we assume that all the nodes are binary throughout this paper.
- For each decision or random node Z, given an assignment of $\pi(Z)$, the distribution $P(Z|\pi(Z))$ specifies the probability of Z being in each state of the node Z. Such a distribution is called a Conditional Probability Table (CPT) in the case that the domain of the variable Z is a finite set.
- For each value node U, g_U is a value function $g_U : \Omega_{\pi(U)} \rightarrow R$, where R denotes the set of the real numbers.

To avoid unnecessary notations, we define the (optimal) policy concept only for a simultaneous ID [1]. A *policy*, denoted by δ, specifies one action choice for each decision node in $\mathcal{X}$. Hence, a policy δ can be denoted by $(\delta_1, \ldots, \delta_n)$, where δ_i belongs to the domain of X_i for each i.

Given a policy δ, a probability P_δ can be defined over the random nodes and decision nodes as follows.

$$P_\delta(\mathcal{Y}, \mathcal{X}) = \Pi_{Y \in \mathcal{Y}} P(Y|\pi(Y)) \Pi_{i=1}^{n} P_\delta(X_i) \quad (1)$$

[1]For general IDs, the definition of an (optimal) policy can be found in, e.g., Qi and Poole 1995.

where $P(Y|\pi(Y))$ is specified in the definition of $\mathcal{I}$, while $P_\delta(X_i)$ equals to 1.0 if $X_i = \delta_i$ and 0.0 otherwise.

The *expectation of the value node U under policy δ*, denoted by $E_\delta[U]$, is defined as

$$E_\delta[U] = \sum_{\pi(U)} P_\delta(\pi(U)) g_U(\pi(U)). \quad (2)$$

The *expected value E_δ of $\mathcal{I}$ under the policy δ* is the sum $E_\delta[U]$ over all value nodes U in $\mathcal{U}$, i.e.,

$$E_\delta = \sum_{U \in \mathcal{U}} E_\delta[U]. \quad (3)$$

For simplicity, E_δ is also called *the expected value of policy δ*. *Evaluating a policy δ* means to compute its expected value. The maximum of E_δ over all policies is the *optimal (expected) value of $\mathcal{I}$*. An *optimal policy* is the policy that achieves the optimal expected value. To *evaluate an ID* is to find an optimal policy and to compute its optimal expected value.

IV. The Factorization Approach

In this section, we describe the representation theorem and the factorization approach.

A. The Idea

From its definition, an ID is a network structure consisting of decision nodes, random nodes and value nodes. Among them, in determining the expected value of the ID, a decision node plays a different role from a random or a value node. The choices of a decision node can affect the expected value of the ID through changing the CPTs of its child random nodes, or through changing the value functions of its child value nodes (note that a decision node cannot have another decision node as child in a simultaneous ID). In this sense, a node, if it is a child of a decision node, serves as an *interface* through which the choices of decision nodes may affect the value of the ID. Such a node is called an *interface node*. All interface nodes constitute an *interface set*. Collectively, an interface set serves as an interface of an ID through which policies can affect the expected value of the ID. Consequently, an ID can be divided into two fractions: the *up-stream fraction* which includes the interface nodes and the nodes "preceding" them, and the *down-stream fraction*, which includes the interface nodes and the nodes "succeeding" the interface nodes.

Example We use the ID in Fig. 1 to informally illustrate these concepts. The ID has two decision nodes $\{X_1, X_2\}$, five random nodes $\{A, B, C, D, H\}$ and one value node U. The interface set $\mathcal{Y}_{in}$ is $\{A, C\}$ since they have parental decision nodes. The up-stream is $\{X_1, X_2, A, B, C\}$, which consists of two interface nodes A and C, node X_1 preceding node A, and nodes B and X_2 preceding node C. The down-stream is $\{A, C, H, D, U\}$, which consists of interface nodes A and C, the nodes H, D, U succeeding to the interface nodes.

□

Interestingly, corresponding to the structural separation that an ID can be divided into two fractions, the expected value of a value node under a policy breaks into two fractions, each of which involving only the up-stream or the down-stream of the ID.

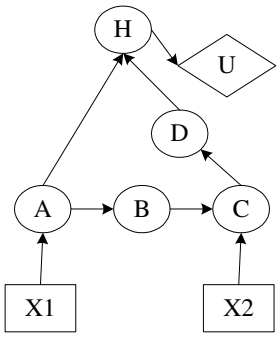

Fig. 1. An ID to illustrate the representation theorem

B. The Theorem

We formalize the above idea in this subsection. For the sake of simplicity, throughout the paper, unless explicitly stated, we assume that (1) the ID has only one value node, and (2) the value node has no decision node as its parent. We also note that our results in this paper generalize to the IDs with multiple value nodes and with value nodes having parental decision nodes. We relax these assumptions at the end of this subsection.

We begin by defining several concepts. A random node Y is an interface node if its parent set has at least one decision variable, i.e., $\pi(Y) \cap \mathcal{X} \neq \emptyset$. The interface set of an ID is the set of all interface nodes. Due to the above assumptions, the interface set contains only random nodes; for this reason, we denote the set by $\mathcal{Y}_{in}$. The *up-stream* of the ID includes the interface set and all ancestors of the nodes in the interface. By this definition, in addition to the interface random nodes and decision nodes, the up-stream may contain the random-node ancestors of the interface nodes. These ancestral nodes must be included because they, together with decision nodes, determine the CPTs of the interface nodes. These ancestral random nodes form a set denoted by $\mathcal{Y}_0$.

Given an ID, we can efficiently identify its up-stream using a queueing mechanism. We initialize a queue to be the interface set $\mathcal{Y}_{in}$ (it can be readily built by checking whether there is a parental decision node for every node in the ID) and the up-stream $\mathcal{I}_{up}$ to be empty. At each step, a node is removed from the queue and added to $\mathcal{I}_{up}$ if it is not in $\mathcal{I}_{up}$. The parents of the node, if not present in $\mathcal{I}_{up}$ thus far, are added to the queue. The procedure terminates when the queue is empty. When it terminates, the set $\mathcal{I}_{up}$ becomes the up-stream set. The procedure must terminate after a finite number of steps because an ID is a directed acyclic graph.

The up-stream can be partitioned into three sets: the set $\mathcal{X}$ of decision nodes, interface set $\mathcal{Y}_{in}$ and the set $\mathcal{Y}_0$ of random-node ancestors of interface nodes. Given a policy δ, we define a function f_δ from $\Omega_{\mathcal{Y}_{in}}$ to the real line R. For notations, we let m be the number of nodes in the set $\mathcal{Y}_{in}$, $Y_{in}^{1:m}$ be a short notation of $\{Y_{in}^1, \ldots, Y_{in}^m\}$, and $y_{in}^{1:m}$ be an assignment to all interface variables, i.e., an element of $\Omega_{Y_{in}^{1:m}}$.

$$f_\delta(y_{in}^{1:m}) = \sum_{Y \in \mathcal{Y}_0} \Pi_{Y \in \mathcal{Y}_0 \cup \mathcal{Y}_{in}} P_\delta(Y|\pi(Y)) \Pi_{i=1}^n P_\delta(X_i) \quad (4)$$

where $\Pi_{Y \in \mathcal{Y}_0 \cup \mathcal{Y}_{in}} P_\delta(Y|\pi(Y)) \Pi_{i=1}^n P_\delta(X_i)$ is the joint probability distribution of the variables in $\mathcal{X}$, $\mathcal{Y}_0$ and $\mathcal{Y}_{in}$ given policy δ. Hence, $f_\delta(Y_{in}^{1:m})$ is the conditional probability that the interface $Y_{in}^{1:m} = y_{in}^{1:m}$ occurs upon the policy δ. For convenience, we call them *interface probabilities*.

In contrast to the up-stream, *the down-stream of an ID* is the set consisting of all the interface nodes and their descendants. The down-stream contains the value node, the interface nodes and the random nodes that do not belong to the up-stream. We use $\mathcal{Y}_1$ to denote the set of non-interface random nodes in the down-stream. Note that the random nodes in the interface set belong to both the up-stream and the down-stream. For an assignment $y_{in}^{1:m}$ of the set $Y_{in}^{1:m}$ and the value node U, we can define a function as follows.

$$f_{in,U}(y_{in}^{1:m}) = \sum_{Y \in \mathcal{Y}_1} \Pi_{Y \in \mathcal{Y}_1} P_\delta(Y|\pi(Y)) g_U(\pi(U)). \quad (5)$$

To see that $f_{in,U}$ is a function of $Y_{in}^{1:m}$, we note that the interface variables may appear in $\pi(Y)$ for $Y \in \mathcal{Y}_1$. Given an assignment $y_{in}^{1:m}$ of $Y_{in}^{1:m}$, $f_{in,U}(y_{in}^{1:m})$ is the expected utility conditioned on the assigned interface $y_{in}^{1:m}$. These quantities are called *interface utilities* for convenience. Since $\pi(Y)$ for Y in $\mathcal{Y}_1$ must belong to the down-stream, $P_\delta(Y|\pi(Y))$ is independent of policy δ. Consequently, these utilities are independent of policy δ.

Theorem 1: Given a policy δ and a value node U, the expected value of the node U under policy δ

$$E_\delta[U] = \sum_{y_{in}^{1:m} \in \Omega_{Y_{in}^{1:m}}} f_\delta(y_{in}^{1:m}) \cdot f_{in,U}(y_{in}^{1:m}). \quad (6)$$

Proof: We show that $E_\delta[U]$ can be rewritten as the sum of the interface utility $f_{in,U}$ weighted by the probability f_δ over all interfaces.

$$\begin{aligned}
& E_\delta[U] \\
&= \textstyle\sum_{\pi(U)} P_\delta(\pi(U)) g_U(\pi(U)) && \text{(a)} \\
&= \textstyle\sum_{\pi(U)} [\sum_{\mathcal{Y}/\pi(U)} \Pi_{Y \in \mathcal{Y}} P(Y|\pi(Y)) \Pi_{i=1}^n P_\delta(X_i)] \\
&\quad g_U(\pi(U)) && \text{(b)} \\
&= \textstyle\sum_{Y \in \mathcal{Y}_{in}} \sum_{Y \in \mathcal{Y}_0} \sum_{Y \in \mathcal{Y}_1} \Pi_{Y \in \mathcal{Y}_0 \cup \mathcal{Y}_{in}} P(Y|\pi(Y)) \\
&\quad \Pi_{Y \in \mathcal{Y}_1} P(Y|\pi(Y)) \Pi_{i=1}^n P_\delta(X_i) g_U(\pi(U)) && \text{(c)} \\
&= \textstyle\sum_{Y \in \mathcal{Y}_{in}} \left[\sum_{Y \in \mathcal{Y}_0} \Pi_{Y \in \mathcal{Y}_0 \cup \mathcal{Y}_{in}} P(Y\pi(Y))\right] \\
&\quad \left[\sum_{Y \in \mathcal{Y}_1} \Pi_{Y \in \mathcal{Y}_1} P(Y|\pi(Y)) g_U(\pi(U))\right] && \text{(d)} \\
&= \textstyle\sum_{y_{in}^{1:m} \in \Omega_{Y_{in}^{1:m}}} f_\delta(y_{in}^{1:m}) \cdot f_{in,U}(y_{in}^{1:m}) && \text{(e)}
\end{aligned}$$

Step (a) is true by Equation (2). At Step (b), $\mathcal{Y}/\pi(U)$ is the difference set of $\mathcal{Y}$ and $\pi(U)$. This step is true by inserting Equation (1) into Equation (2). Step (c) follows from the fact that $\{\pi(U), \mathcal{Y}/\pi(U)\}$ and $\{\mathcal{Y}_0, \mathcal{Y}_{in}, \mathcal{Y}_1\}$ are two partitions of the set $\mathcal{Y}$. At Step (d), we break the distribution $\Pi_{Y \in \mathcal{Y}} P(Y|\pi(Y)) \Pi_{i=1}^n P_\delta(X_i)$ into two fractions $\Pi_{Y \in \mathcal{Y}_0 \cup \mathcal{Y}_{in}} P(Y|\pi(Y))$ and $\Pi_{Y \in \mathcal{Y}_1} P(Y|\pi(Y)) g_U(\pi(U))$. At Step (e), we replace the two fractions with the definitions of the interface utilities and interface probabilities. □

By the theorem, given a policy δ and a value node U, the expected value of the node under the policy can be represented as the sum of the multiplications of the interface utilities and corresponding interface probabilities.

Example (Continued) For the ID in Fig. 1, we show how to represent $E_\delta[U]$ for the value node U and a given policy δ. Let the policy δ be (δ_1, δ_2) where δ_i is the decision choice of X_i for $i = 1, 2$. By definition,

$$\begin{aligned} E_\delta[U] \quad &= \textstyle\sum_H \sum_{ABCD} P(A|\delta_1\delta_2)P(B|A)P(C|B\delta_2) \\ &\quad P(D|C)P(H|AD)\Pi_{i=1}^2 P_\delta(X_i)g_U(H). \end{aligned}$$

The two functions are defined as follows.

$$\begin{aligned} &f_\delta(A, C) = \textstyle\sum_B P(A|\delta_1\delta_2)P(B|A)P(C|B\delta_2)\Pi_{i=1}^2 P_\delta(X_i) \\ &f_{in,U}(A, C) = \textstyle\sum_{HD} P(H|AD)P(D|C)g_U(H). \end{aligned}$$

It can be verified that $E_\delta[U] = \sum_{A,C} f_\delta(A, C) \cdot f_{in,U}(A, C)$. □

We examine the assumptions we made at the beginning of this subsection. First, we have assumed that there is only one value node. In case of multiple value nodes, we may apply the representation theorem to each node. The expected value of a policy is the additive sum of the expected values of all value nodes under the policy.

Second, we have assumed that the value node has no decision nodes as its parents. In the other case that the value node has a decision node as its parents, the functions $f_{\delta,U}$ and f_{in} can be defined as follows such that the theorem holds.

$$\begin{aligned} f_{\delta,U}(Y_{in}^{1:m}) \quad &= \textstyle\sum_{Y \in \mathcal{Y}_0} \Pi_{Y \in \mathcal{Y}_0 \cup \mathcal{Y}_{in}} P_\delta(Y|\pi(Y)) \\ &\quad \Pi_{i=1}^n P_\delta(X_i)g_U(\pi(U)) \\ f_{in}(Y_{in}^{1:m}) \quad &= \textstyle\sum_{Y \in \mathcal{Y}_1} \Pi_{Y \in \mathcal{Y}_1} P_\delta(Y|\pi(Y)). \end{aligned}$$

Therefore we can lift the assumption that the value node has no decision node as parents. In this case, U is also called an interface node, but it belongs to the up-stream only. The reason is that by definition a value node cannot have children and therefore cannot produce impact on the down-stream [2]. Note that f_δ changes to $f_{\delta,U}$ since the value node is considered in computing the quantities relevant to the up-stream. Interestingly, it can be proven that $f_{in}(= 1.0)$ is a constant. To see why, let us assume that the size of $\mathcal{Y}_1$ be k. We enumerate the set $\mathcal{Y}_1$ as $\{Y_1^1, \ldots, Y_1^k\}$ such that a node's parents appear after the node in the set. In computing $f_{in}(Y_{in}^{1:m})$, we can sequentially sum out the variables in $\mathcal{Y}_1$ in the enumerated order. Ultimately, we have $f_{in} = 1.0$.

C. The Algorithm

By the representation theorem, the expected value of a policy is represented as the sum of interface utilities weighted by the corresponding interface probabilities. The interface utilities are independent of the individual policies, whereas the interface probabilities are dependent on the policies. Therefore, the interface utilities can be factored out, i.e., they can be calculated once and re-used across all the policies.

[2]Note that this is different from random nodes having parental decision nodes. Such a random node belongs to both the up-stream and the down-stream.

This is the idea behind our factorization approach, which is described in Table I. The factored-out computations are calculated once at line 1. They are used for all policies at line 2.2. Note that the procedure below generalizes to IDs with multiple value nodes.

1. Pre-compute the quantities $f_{in,U}(y_{in}^{1:m})$ for all $y_{in}^{1:m}$ in $\Omega_{Y_{in}^{1:m}}$
2. For each policy δ
   2.1 compute $f_\delta(y_{in}^{1:m})$ for each assignment of $Y_{in}^{1:m}$
   2.2 compute $E_\delta[U]$ by Equation (6)
3. Return the policy that maximizes $E_\delta[U]$

TABLE I
THE FACTORIZATION APPROACH TO ID EVALUATION

D. Complexity Analysis

It is of interest to compare the approach and the generic brute-forced approach that evaluates a policy directly by combining Equation (1) and (2). Let n be the number of decision nodes. Thus the size of the policy space is 2^n. Let the complexity of evaluating one policy be C. The complexity C breaks into three pieces: computing f_δ, computing $f_{in,U}$, and computing $E_\delta[U]$ by Equation (6). We denote them respectively by C_1, C_3 and C_2. To evaluate all policies, the generic approach has complexity $2^n C$, i.e., $2^n(C_1+C_2+C_3)$. In contrast, since the factorization approach computes $f_{in,U}$ only once but uses them 2^n times, its complexity is $2^n(C_1+C_2)+C_3$. A good measure to predict the computational gain is the size of the down-stream, i.e., the number of nodes in it. In one extreme, if the down-stream contains only the interface nodes and the value nodes (thus C_3 is a constant), the two approaches have the same complexity. In the other extreme, if the down-stream contains far more nodes than the up-stream (i.e., $C_3 >> C_1+C_2$), the computational gain is significant.

E. Bounding the Optimal Expected Value

We show that the interface utilities computed in the factorization approach can be used to derive both an upper and a lower bound of the optimal value of the ID. These bounds have significant implications in practical planning.

We define $f_{in,U}^+$ and $f_{in,U}^-$ to be the largest and the smallest one among all interface utilities, i.e.,

$$\begin{aligned} f_{in,U}^+ &= \max_{y_{in}^{1:m} \in \Omega_{Y_{in}^{1:m}}} f_{in,U}(y_{in}^{1:m}) \\ f_{in,U}^- &= \min_{y_{in}^{1:m} \in \Omega_{Y_{in}^{1:m}}} f_{in,U}(y_{in}^{1:m}) \end{aligned}$$

where the $\max$ and $\min$ are taken over the domain of the variables in $Y_{in}^{1:m}$. From the theorem, we see that $f_{in,U}^+(f_{in,U}^-)$ is the upper (lower) bound of the optimal value of the ID. These bounds have significant importance in practice. Suppose, for instance, that these bounds are available to a planner. In one extreme, if the planner expects a utility that is larger than the upper bound, he never bothers to evaluate all the policies and finds the optimal one because even the best policy provides less than he expects. In this case, he

needs to redesign the network structure or parameters such that the performance of the ID can be improved. In the other extreme, if the planner expects an utility that is less than the lower bound, again he never bothers to evaluate all the policies and chooses the optimal one because any policy can provide more than he expects. In this case, he can pick any policy and execute it.

We note that from the computational point of view, computing these bounds is easier than evaluating the ID. There are two reasons. First, as discussed earlier, computing these bounds involves only the down-stream of the ID, whereas evaluating an ID involves its entire structure. If the down-stream contains much fewer variables than the up-stream, the interface utilities (and also the bounds) can be obtained efficiently. Second, computing these bounds avoid enumerating all the policies and calculating their expected values.

Finally, the tightness of the bounds depends on the structure of an ID, the CPTs of random nodes and the value functions of value nodes. It is difficult to characterize a general condition to determine the tightness of the bounds. In our experiments, we empirically show that these bounds are reasonably tight for the tested problems.

V. Extensions to The Factorization Approach

In this section, we discuss two extensions to the factorization approach. These extensions deal with reconstructing the policies as the network structure/parameters undergo changes. There are two perspectives. First, at one decision step, the network might change such as more actions being available for a planner's choice, more value nodes needed consideration and so on. Second, the network might dynamically alter its structure or parameters as time goes by. For example, if a subgoal is successfully accomplished at one step, it can be removed from the network in the subsequent steps.

A. Network Structure/Parameter Changes

The principle for the factorization approach to accommodate structure or parameter changes is as follows. First, if the changes involve only the up-stream of an ID, the interface utilities do not need to be re-computed and can still be shared in evaluating the ID. Specifically, these changes include addition or removal of decision/random/value nodes and also the alternation of CPTs and value functions in the up-stream. Second, if the changes involve only the down-stream of the ID, the approach needs re-construct the interface utilities. Fortunately, the interface probabilities are preserved and the calculations for them can be saved. Third, if the changes involve not only the up-stream but also the down-stream, the approach needs to re-compute both the interface utilities and the interface probabilities.

B. Planning Over Time

In realistic applications, network parameters may change over time. In this case, we can use a *dynamic ID* to model the conditional dependencies among nodes over time. In this subsection, we show how the factorization approach can be used to reconstruct the policies on a step by step basis for dynamic IDs.

To facilitate our discussions, we extend the example in Fig. 1 to a dynamic ID. We assume that the variable H evolves over time and let H_t denote H at step t. The dynamic ID is drawn in Fig. 2. In contrast, the ID in Fig. 1 is said to be *static* since the multiple decisions are made at one time step.

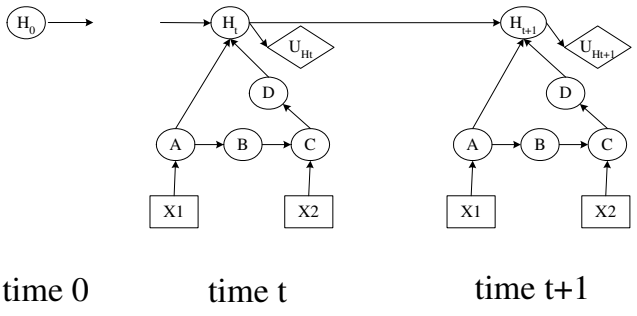

Fig. 2. A dynamic ID model

The dynamic ID has two prominent features. First, at a single step, the decision problem can be modelled as a static ID. In addition to the nodes and links in Fig. 1, the node H_{t+1} at step $t+1$ has one more parent node H_t. Second, the inter-temporal link between two consecutive nodes carries the historic information about the sequence of performed policies. For step $n+1$, the information can be summarized by a probability distribution of H_t conditioned on the history [14].

For a dynamic ID, we are interested in optimal planning on the step-by-step basis. The problem is formulated as: given an initial probability distribution $P(H_0)$, at step $t+1$, how to efficiently find the policy $\delta(= (\delta_1, \ldots, \delta_n)$ where n is the number of decision nodes) that maximizes $E_\delta[U_{H_{t+1}}]$? To solve the problem, we show (1) how to select the optimal policy at step $t+1$ given the probability distribution $P(H_t)$, and (2) how to sequentially update the probability distribution $P(H_{t+1})$ from $P(H_t)$ given a policy δ at the previous step. After these two questions are settled, we may choose the optimal policy as follows. At step $t+1$, we first choose the optimal policy for the step and then update the probability $P(H_{t+1})$ from $P(H_t)$. The procedure repeats at each step.

To answer the first question, we introduce the concept of *augmented interface node* and *augmented interface set*. We call the node H_t *an augmented interface node of the ID at step* $t+1$ since the node H_t can produce impact on the network via altering its probability distribution. In this sense it is an interface node [3]. The *augmented interface set* consists of $\mathcal{Y}_{in}$ as before and the node H_t. The down-stream of the ID at step $t+1$ remains the same as that of the static ID. Likewise, we may define the two functions f_δ and $f_{in,U_{H_{t+1}}}$. Therefore we can use the factorization approach to solve the planning problem over time. The computations involving $f_{in,U_{H_{t+1}}}$ are

[3]Previously, we defined an interface node to be a node that has parental decision nodes since the choices of decision nodes can affect its CPTs and in turn the expected value of the ID. In contrast, the node H_t is called *an augmented interface node* since it can change its probability distribution and thus affect the expected value of the ID.

factored out. Note that these interface utilities are shared for all policies at each decision step. For the ID in Fig. 2, we can define the following functions for the ID.

$$f_\delta(A,C,H_t) = P(H_t)\sum_B P(A|\delta_1\delta_2)P(B|A)P(C|B\delta_2)\Pi_{i=1}^2 P_\delta(X_i)$$
$$f_{in,U_{H_t}}(A,C,H_t) = \sum_{H_{t+1}D} P(H_{t+1}|ADH_t)P(D|C)g_U(H_{t+1}).$$

It can be verified that $E_\delta[U_{H_{t+1}}] = \sum_{A,C,H_t} f_\delta(A,C,H_t)\cdot f_{in,U_{H_t}}(A,C,H_t)$.

To answer the second question, we show how to efficiently compute $P(H_{t+1})$ given a distribution $P(H_t)$ and the policy δ performed at step t. We introduce a technique such that the procedure of computing $P(H_{t+1})$ can be conducted similarly to that of computing $E_\delta[U_{H_{t+1}}]$. Suppose that H_{t+1} can take on two values h(true) and $\neg h$(false). We first show how to calculate the probability of H_{t+1} being true. Let V be a value node that differs from $U_{H_{t+1}}$ only in its value function. Specifically, g_V is 1.0 if its parent H_{t+1} is true; it is 0.0 otherwise. For simplicity, let $H_{t+1} = h(\neg h)$ denote the event that the hypothesis H_{t+1} is true (false). We prove that $E_\delta[V] = P_\delta(H_{t+1} = h)$.

Proposition 1: $E_\delta[V] = P_\delta(H_{t+1} = h)$.

Proof:

$$\begin{aligned} E_\delta[V] &= \sum_{H_{t+1}} P_\delta(H_{t+1})g_V(H_{t+1}) \\ &= P_\delta(H_{t+1}=h)g_V(H_{t+1}=h) \\ &\quad + P_\delta(H_{t+1}=\neg h)g_V(H_{t+1}=\neg h) \\ &= P_\delta(H_{t+1}=h). \end{aligned}$$

In the last step, we use the definition of the value function g_V. □

To calculate the probability of H_{t+1} being false, we may define g_V as follows: it is 1.0 if its parent H_{t+1} is false; it is 0.0 otherwise. If we define two functions f_δ and $f_{in,V}$, we see that the computational steps for $E_\delta[V]$ are the same as those for computing $E_\delta[U_{H_{t+1}}]$. Hence, computing $P(H_{t+1})$ does not add much overhead to ID evaluation.

It is interesting to compare the generic approach and the factorization approach in the context of the dynamic ID. Let the number of decision steps be T. Recall that the complexity of computing f_δ is C_1, the complexity of computing $f_{in,U_{H_{t+1}}}$ is C_3, and the complexity of computing $E_\delta[U_{H_{t+1}}]$ is C_2. Since C_1 takes constant time, it can be ignored. For one decision step, the factorization approach has the complexity $2^nC_2+C_3$ while the generic approach has the complexity $2^n(C_2+C_3)$. For T steps, the complexity of the factorization approach is $2^nTC_2+C_3$ (note that this does not include the overhead of computing $P(H_{t+1})$), while the complexity of the generic approach is $2^nT(C_2+C_3)$. If $C_3>>C_2$, the factorization approach can be extremely efficient.

VI. Experiments

In this section, we first report our experiments on both simulation studies and a military planning example. In our experiments, we wrote Matlab-V6.5 codes and ran them on a laptop with a 2.0 GHz CPU under Windows XP. We compare the factorization approach against the generic brute-forced approach. We chose the generic approach because we are not aware of specific algorithms for evaluating simultaneous IDs. For convenience we refer to the two algorithms as `evalCS` (named after Computation Sharing) and `evalBF` (named after Brute-Forced).

A. Simulation Studies

To thoroughly evaluate the performance of the factorization approach, we conducted simulated studies on the ID in the left chart of Fig. 3, which is similar to the military planning examples in [15]. It is referred to as the static ID in the rest of this section. The CPTs are randomly generated. The value functions for value nodes are manually specified.

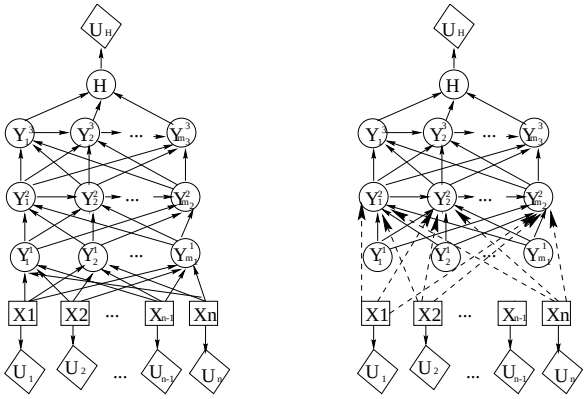

Fig. 3. The left chart is a test example, while the right is its variant for comparative studies.

Specifically, our experiments are designed to (1) evaluate the performances of the factorization approach for static and dynamic IDs, (2) show the tightness of bounds derived from the interface utilities, (3) demonstrate how the computational gain achieved by the approach varies with different network structures, and (4) demonstrate the computational gain by adapting the approach to account for newly added decisions and value nodes.

1) Performances of the Factorization Approach: To see how the performances of the algorithms vary with the number of decision nodes, we fix the number of random nodes at each level at four and vary the number of decision nodes. Thus the static ID with n decision nodes has additionally 10 random nodes and $n+1$ value nodes. We ran `evalBF` and `evalCS` for seven problems with $n = 3, 5, ..., 13$. The timing data are presented in the left chart of Fig. 4. The chart gives the total CPU seconds that the algorithms took for each of the problems. Note that the vertical direction is drawn in log-scale. The solid (dashed) curve is for `evalCS`(`evalBF`). It can be seen that `evalCS` is considerably more efficient than `evaBF`. For instance, from our data, for $n = 9$, to evaluate 512 policies, `evalCS` took 3.51 seconds while `evalBF` 646.74 seconds; for $n = 13$, to evaluate 8192 policies, `evalCS` used 46.32 seconds while `evalBF` 9284.05 seconds.

To quantitatively characterize how much savings the factorization procedure can bring about, we use the timing results of `evalCS` to predict the performance of `evalBF`. Recall that the complexity of evaluating a policy breaks into three

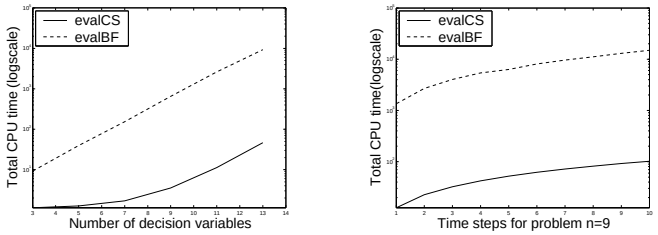

Fig. 4. Performance comparison of evalBF and evalCS

fractions C_1, C_3 and C_2. We ignore C_1 since it is a constant. For each problem, we estimate C_3 by the actual seconds $\hat{C}_3$ of computing f_{in,U_H}, and C_2 by $\hat{C}_2$ as $\frac{the\ total\ CPU\ time - \hat{C}_3}{the\ number\ of\ policies}$. The complexity of `evalBF` is predicted by $2^n(\hat{C}_2+\hat{C}_3)$. We found that these estimations are almost the same as the actual timing results of `evalBF`. This suggests the effectiveness of our complexity analysis.

We also tested the algorithms over the dynamic ID in Fig. 5, which is an extension of the left chart of Fig. 3.

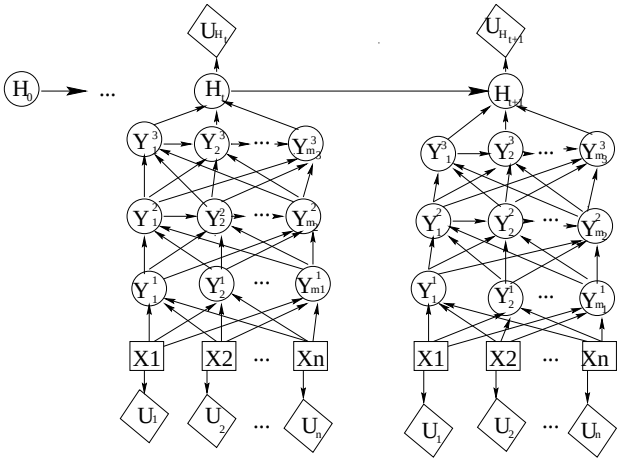

Fig. 5. A dynamic influence diagram

Initially, the probability of the node H being false is set to 1.0. Its probability is updated at each decision step. We ran both algorithms for up to 10 decision steps. We showed the total CPU time for the ID with 9 decision nodes in the right chart of Fig. 4. The chart gives the total CPU seconds for both algorithms against the time steps. Note that again the vertical direction is drawn in log-scale. It can be seen that the CPU time linearly increases with the elapsed time for `evalBF` while its increase is negligible for `evalCS`. This is not a surprising observation. In `evalBF`, all policies are evaluated at each step. The time cost for all steps remains the same. Hence the increase is linear. From our data, `evalBF` uses about 1,300 seconds to evaluate all 512 (2^9) policies at each step. However, in `evalCS`, the interface utilities are computed only once at the first step. So we observe that at the first step, `evalCS` takes about 2.66 seconds to compute these utilities; thereafter, each step takes about only 10 seconds to evaluate all 512 policies. The increase is negligible when compared against that in `evalBF`.

2) Tightness of Bounds: To show the tightness of the upper and lower bounds of the optimal expected value, in Fig. 6, we plot the optimal value (the middle curve) and these bounds (the upper and lower curves) for the static IDs with 3,5,...,13 decision nodes. We see that these bounds are reasonably tight for the tested problems. For example, for $n = 7$, the optimal value is 871.47 while the bounds are 722.91 and 936.637. Although it is difficult to quantitatively analyze the properties of these bounds, these experiments show they can be tight at least for these tested problems.

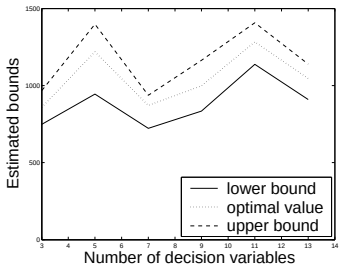

Fig. 6. The lower and upper bounds obtained from the interface utilities

3) Computational Gain Under Network Structure Changes: To demonstrate how the computational gain of `evalCS` varies with different network structures, we run `evalCS` over the static ID and a modified version of it. The modified ID is obtained as follows: every link from X_i to Y_j^1 is re-directed to Y_j^2. The resulted ID is shown in the right chart of Fig. 3. Its up-stream is $\mathcal{X} \cup Y^1_{1:m_1} \cup Y^2_{1:m_2} \cup U_{1:n}$, whereas its down-stream is $Y^3_{1:m_3} \cup \{H\} \cup \{U_H\}$, where $U_{1:n}$ means the set of value nodes, and $Y^i_{1:m_i}$ means the set of nodes Y^i_j, i.e., $Y^i_{1:m_i} = \{Y^i_1, \ldots, Y^i_{m_i}\}$ for $i = 1, 2, 3$. Compared with that of the static ID, the down-stream of the modified ID contains fewer random nodes. We expect (1) `evalCS` is still more efficient than `evalBF` in the modified ID since its down-stream contains a number of random nodes, and (2) `evalCS` achieves less savings in modified ID than it does in the original ID.

The experiments presented in Fig. 7 confirm these expectations. First, the left chart plots the CPU seconds (in log-scale) that `evalCS` and `evalBF` take for the modified IDs with different number of decision variables. It can be seen that `evalCS` is more efficient. Second, the right chart plots the magnitudes of the savings brought by `evalCS`. For each approach, the saving magnitude is measured by the quotient of the total time of `evalBF` and that of `evalCS`. The magnitudes are drawn in the vertical direction. For a modified ID, `evalCS` is about 14 times faster than `evalBF`. For the original IDs, the magnitudes vary with the number of their decision nodes. We see that the computational savings brought by `evalCS` are more significant for IDs whose down-stream contains more nodes.

4) Computational Gain Under Network Parameter Changes: We also conducted experiments to show how the factorization approach achieves computational savings as the network changes. For this purpose, given the static ID, we first evaluate it (the planning phase), then add more nodes

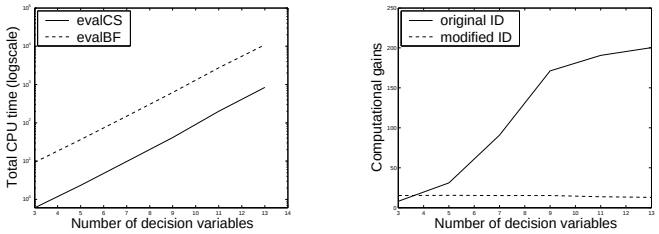

Fig. 7. Computational gains versus network structure

to the ID and re-evaluate it (the replanning phase). We like to compare both the replanning time and total time of the factorization approach against that of the generic approach.

In one experiment, we first evaluate a static ID with n decision nodes. We then add two decision nodes to the ID and evaluate the modified ID. Every newly added node has a link from itself to every Y_j^1 node. The timing results in log-scale are presented in the left chart of Fig. 8. In the chart, the curve corresponding to `CS1`(`BF1`) depicts the replanning time for the factorization (generic) approach, whereas the curve corresponding to `CS`(`BF`) depicts the total time similarly. We see that for the tested problems, the factorization approach achieves considerable savings in replanning when more decision nodes are added. For instance, for the ID with nine decision nodes, the factorization and generic approach respectively takes 9.80 seconds and 2313.94 seconds. These savings are achieved through sharing the interface utilities computed during evaluating the original ID. Since the factorization approach takes much less time in both evaluating and re-evaluating the ID, its total time is considerably less than that used by the generic approach.

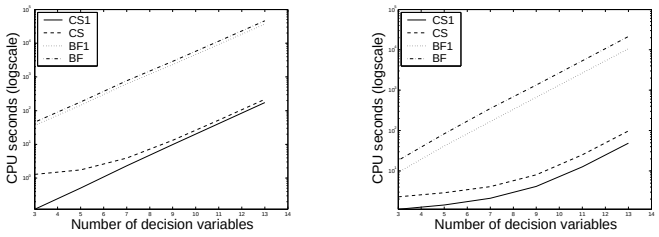

Fig. 8. Replanning for added action/value nodes. In both charts, the curves from the top to bottom plot the total/replanning time for the generic approach, and the total/replanning time for the factorization approach.

In the other experiment, we evaluate the ID and then add one more value node for replanning. The added value node has a link from every node Y_j^2 for $j = 1, \ldots, 4$. In computing the expected value of the added value node, we still use the factorization approach. In re-evaluating an ID, we do not re-compute the expected value of the existing value node. The timing results in log-scale are collected in the right chart of Fig. 8. The legends read similarly to those in the left chart. It can be seen that the factorization approach can achieve great savings in replanning. The reason is obvious: the factorized computations are saved in computing the expected value of the newly added value node. By taking advantage of shared computations in evaluating two value nodes, the total time used by the factorization approach is considerably less than that by the generic approach.

B. A Military Planning Example

We applied the factorization approach to a hypothetical military planning example, which is illustrated in Fig. 9. The overall military goal is to win a war or to bring a tyrant to justice. The goal is represented by a `Hypothesis` node, which is on the top of the figure. There are 12 primitive actions, namely `destroy_C2`, `destroy_Radars`,..., `operate_special_force`, which are on the bottom side. Performing an action has direct effects of specific purpose. For instance, if the action `destroy_Radars` is performed, the `EW/GCIRadars` is destroyed with a high probability. These effects alter the overall goal through altering the low-level subgoals. For instance, the status of `C2` (Command and Control), `EW/GCIRadars` and `Communications` facilities and `Air_strike` determine the workability of Integrated Air Defense System (IADS) and the strength of the enemy air force. In turn, the workability of IADS system and the strength of the enemy air force determine the loss of `Air_superiority`. The `Air_superiority`, `Territory_occupation`, `Commander_surrender` are three subgoals determining the overall goal success. Without loss of generality, we assume all nodes are binary. In the example, each decision node is associated with a value node encoding the cost of performing the action, and the hypothesis node is associated with a value node encoding the utility of goal success. The optimal policy needs to balance the utility of goal success and cost of performing actions.

We designed a reasonable set of CPTs and value functions. For the `Hypothesis` node, if all the subgoals `Air_superiority`, `Territory_occupation` and `Command_surrender` are achieved, the overall goal is successfully achieved; if one of the subgoals is to be achieved, the probability of the overall success is decreased by 0.3; however, if none of the subgoals is achieved, the overall goal fails with certainty. Similarly, for the subgoal `Air_superiority`, the two influencing factors are `IADS` and `Air_force`. If the IADS system works well and `Air_force` is strong, `Air_superiority` is true for the enemy air force; if either the IADS system works poorly or `Air_force` is weak, the probability of `Air_superiority` being true is decreased by 0.5. Other CPTs for `Territory_occupation` and `Command_surrender` are set analogously to those for `Air_superiority`. A similar strategy is used in parameterizing the nodes `IADS`, `Air_force`, `Artillery`, `Ground_force`, `Morale` and `Commander_in_custody`. In determining the CPTs for random nodes that are immediate children of the decision nodes, we assume that an action achieves its intended effect with probability 0.9. For example, a `destroy_Radars` decision will destroy the EW/GCI radars with probability 0.9. To complete the ID definition, we also assigned value functions. If the goal is successfully achieved, the reward is 1000; otherwise, the cost, i.e., a negative reward, is 500. For

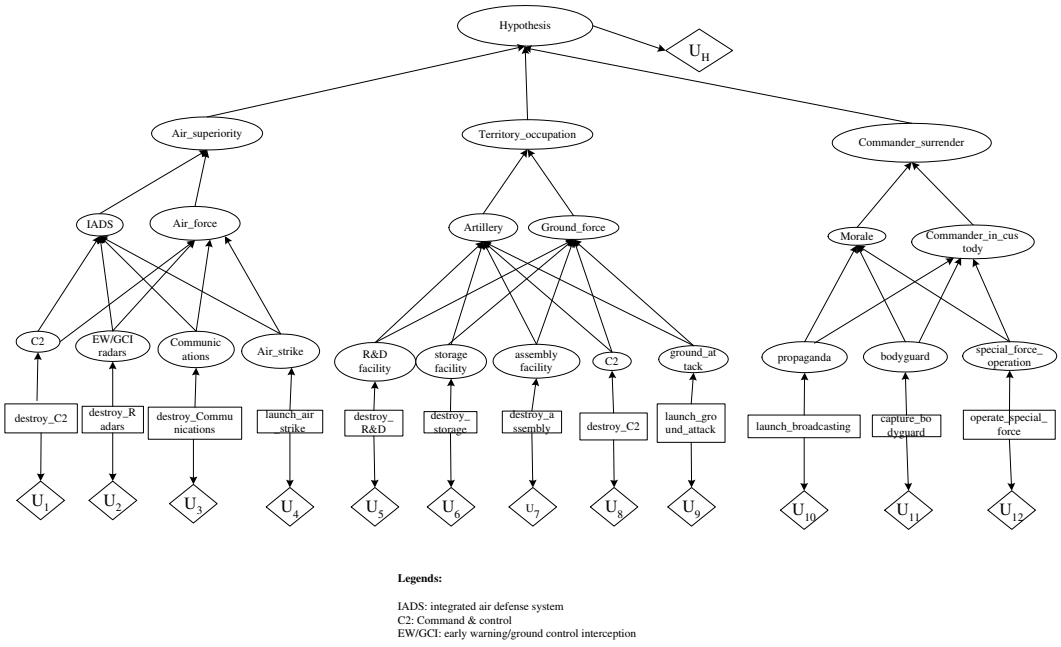

Fig. 9. A static ID illustrating a military planning problem

other decision nodes, if a ground attack is launched, the cost is 150; if the special force operation is performed, its cost is 100; if the commander decides to capture the bodyguards of the tyrant, the operating cost is 80; if an air strike is launched, the cost is 50; for any other actions, their operating cost are 20.

Our primary interest is in the performance of the factorization algorithm. From our data, to evaluate the ID, the factorization algorithm took 45 seconds, while the brute-forced algorithm took 9012 seconds. Hence, the computational saving is tremendous. We can explain the performance difference by the ID structure – its down-stream contains a large number of nodes: all random nodes and the value node associated with the goal. Since its down-stream contains far more random nodes than its up-stream, the approach is expected to be significantly more efficient. Our secondary interest is concerned with the optimal policy. The optimal policy is the one that performs only air strike and special force operation. The expected value of the ID is 561.98, and the probability of goal success is 0.81. We note that the optimal policy excludes "launching a ground attack", although it is the action that is most likely to lead to goal success. One possible reason, as explained earlier, is that the action is excluded due to the high operating cost of performing the action.

VII. Conclusions and Future Work

In this paper, we studied a special ID class, namely simultaneous IDs, where multiple decisions need to be made at one time step. We intended to make two contributions. First, we examined a simultaneous ID and studied its theoretical properties. We showed that such an ID can be decomposed into an up-stream fraction and a down-stream fraction, and that the expected value of a value node under a policy can be represented as the sum of interface utilities that involve only the down-stream fraction, weighted by the corresponding interface probabilities that involve only the up-stream fraction. The interface utilities naturally provide an upper and lower bound of the optimal value of the ID. Second, we proposed a novel factorization algorithm to evaluate a simultaneous ID. The interface utilities are independent of the individual policies; therefore, they can be calculated once but used across all policies in evaluating them. We also extend the the factorization approach to a dynamic ID. The algorithm has been tested on simulation studies and a military planning example. Our experiments showed that the factorization algorithm is significantly more efficient than the generic algorithm in evaluating a simultaneous ID.

To further speed up ID evaluating, one future direction is to combine the factorization approach with the approaches of reducing the search space. In this paper, we addresses one difficulty in ID evaluation, i.e., evaluating individual policies. Another difficulty in ID evaluation is that the policy space contains exponentially many polices and one needs to evaluate all of them in order to find the optimal one. The ID evaluation process can be accelerated if the technique in this paper can be integrated with the approaches of reducing the search space.

Acknowledgement

The work is supported in part by a grant from AFOSR under the grant number F49620-03-1-0160. Part of the work for this project is also supported by the AFRL/Rome summer visiting faculty program. We would like to thank John Lemmer, Don Gossink, and Jerry Dussault from AFRL/Rome for introducing the problem to us and for numerous technical exchanges on the related issues. The authors are grateful to three anonymous reviewers for their insightful comments in improving the paper.

The authors are also grateful to Wenhui Liao and Zhiwei Zhu for insightful discussions.

REFERENCES

[1] R. Howard and J. Matheson, "Influence diagrams," in *The Principles and Applications of Decision Analysis*, R. Howard and J. Matheson, Eds., 1984, pp. 719–762, strategic Decisions Group, Menlo Park, California, USA.

[2] H. Raiffa, *Decision analysis*. Addison-Wesley Publishing Company, 1968.

[3] R. D. Shachter, "Evaluating influence diagrams," *Operations Research*, vol. 34, no. 6, pp. 871–882, 1986.

[4] P. P. Shenoy, "Valuation-based systems for Bayesian decision analysis," *Operations Research*, vol. 40, no. 3, pp. 463–484, 1992.

[5] F. Jensen, F. V. Jensen, and S. L. Dittmer, "From influence diagram to junction trees," in *Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence*, 1994, pp. 367–373.

[6] R. Qi and D. Poole, "A new method for influence diagram evaluation," *Computational Intelligence*, vol. 11, no. 1, pp. 1–34, 1995.

[7] N. L. Zhang and D. Poole, "Stepwise decomposable influence diagram," in *Proceedings of the 4th International Conference on Knowledge Representation and Reasoning*, 1992, pp. 141–152.

[8] G. F. Cooper, "A method for using belief networks as influence diagrams," in *Proceedings of the 4th Workshop on Uncertainty in Artificial Intelligence*, 1988, pp. 55–63.

[9] R. Shachter and M. Peot, "Decision making using probabilistic inference methods," in *Proceedings of the 8th Annual Conference on Uncertainty in Artificial Intelligence (UAI-92)*. San Mateo, CA: Morgan Kaufmann Publishers, 1992, pp. 276–283.

[10] T. Nielsen and F. Jensen, "Well-defined decision scenarios," in *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence*, 1999, pp. 502–511.

[11] F. Jensen and M. Vomlelova, "Unconstrained influence diagrams," in *Proceedings of the 18th Eighteenth Conference on Uncertainty in Artificial Intelligence*, 2002, pp. 234–241.

[12] S. L. Lauritzen and D. Nilsson, "Representing and solving decision problems with limited information," *Management Science*, vol. 47, pp. 1238–1251, 2001.

[13] N. L. Zhang, "Probabilistic inferences in influence diagrams," in *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, 1998, pp. 514–522.

[14] K. J. Aström, "Optimal control of Markov decision processes with incomplete state estimation," *Journal of Mathematical Analysis and Applications*, vol. 10, pp. 403–406, 1965.

[15] U. Kuter, D. Nau, and J. F. Lemmer, "Interactive planning under uncertainty with causal modeling and analysis," Department of Computer Science, University of Maryland, Tech. Rep. CS-TR-4434, 2003.

Qiang Ji Qiang Ji received his Ph.D degree in electrical engineering from the University of Washington in 1998. He is currently an associate Professor with the Department of Electrical, Computer, and Systems Engineering at Rensselaer Polytechnic Institute. His areas of research include computer vision, probabilistic reasoning for decision making and information fusion, pattern recognition, and robotics.

Dr. Ji has published more than 70 papers in peer-reviewed journals and conferences. His research has been funded by local and federal government agencies including NSF, NIH, AFOSR, ONR, DARPA, and ARO and by private companies including Boeing and Honda. His latest research focuses on face detection and recognition, facial expression analysis, image segmentation, object tracking, user affect modeling and recognition, and active information fusion for decision making under uncertainty. Dr. Ji is a senior member of the IEEE.

Weihong Zhang Weihong Zhang received his Ph.D. degree in computer science from the Hong Kong University of Science and Technology in 2001. He then worked as a postdoc researcher at the Department of Computer Science and Engineering, Washington University in Saint Louis. He is currently a postdoc researcher at the Department of Electrical, Computer, and Systems Engineering, Rensselaer Polytechnic Institute.

Dr. Zhang has conducted research in artificial intelligence, probabilistic inferences and decision making under uncertainty, graphical models and their applications in sensor networks and human-computer interaction. He has published more than 10 papers in peer-reviewed journals and conferences.