Aalborg Universitet

AALBORG UNIVERSITY
DENMARK

Anytime decision making based on unconstrained influence diagrams

Luque, Manuel; Nielsen, Thomas Dyhre; Jensen, Finn Verner

Published in:
International Journal of Intelligent Systems

DOI (link to publication from Publisher):
10.1002/int.21780

Publication date:
2016

Document Version
Early version, also known as pre-print

Citation for published version (APA):
Luque, M., Nielsen, T. D., & Jensen, F. V. (2016). Anytime decision making based on unconstrained influence diagrams. International Journal of Intelligent Systems, 31(4), 379-398. DOI: 10.1002/int.21780

Anytime decision making based on unconstrained influence diagrams

Manuel Luque, Thomas D. Nielsen and Finn V. Jensen[*][†]

July 14, 2015

Abstract

Unconstrained influence diagrams extend the language of influence diagrams to cope with decision problems in which the order of the decisions is unspecified. Thus, when solving an unconstrained influence diagram we not only look for an optimal policy for each decision, but also for a so-called step-policy specifying the next decision given the observations made so far. However, due to the complexity of the problem, temporal constraints can force the decision maker to act before the solution algorithm has finished, and, in particular, before an optimal policy for the first decision has been computed. This paper addresses this problem by proposing an anytime algorithm that at any time provides a qualified recommendation for the first decisions of the problem. The algorithm performs a heuristic-based search in a decision tree representation of the problem. We provide a framework for analyzing the performance of the algorithm, and experiments based on this framework indicate that the proposed algorithm performs significantly better under time constraints than dynamic programming.

[*]Manuel Luque is with the Department of Artificial Intelligence, UNED, Madrid, 28040 Spain (e-mail: mluque@dia.uned.es).

[†]Thomas D. Nielsen and Finn V. Jensen are with the Department of Computer Science, Aalborg University, Aalborg, 9220 Denmark (e-mail: {tdn,fvj}@cs.aau.dk)

1 Introduction

An influence diagram (ID) is a framework for representing and solving Bayesian decision problems with a linear temporal ordering of the decisions.[6] However, in many domains, finding an ordering of the decisions is an integral part of the decision problem, and in these situations the use of IDs would require all decision orderings to be explicitly specified in the model.

Unconstrained influence diagrams (UIDs) were introduced to represent and solve decision problems of this type.[8] An optimal strategy in this framework consists not only of an optimal policy for each decision, but also of a step-strategy that prescribes the next decision to consider given the observations and decisions made so far. Such strategies are computable using dynamic programming in a way similar to that for traditional IDs.[7,11,17,18]

Unfortunately, many real world decision problems have an inherent complexity that makes evaluation through exact methods intractable when time is scarce. Moreover, even if you had the time for solving the problem, storing the solution as a simple lookup table may be a problem: the number of possible past scenarios to consider in a policy may be intractably large.

When working under time constraints, the user may need to take the first decision and cannot wait for the algorithm to terminate before deciding on what to do first. For example, Ictneo, an ID model for jaundice management in infants,[3] requires a maximum storage capacity of 1.66×10^{14} positions, thus forcing the authors to devise methods to make the ID tractable. However, slight modifications to the problem, such as adding new medical tests or calculating the best test ordering, could easily make the problem intractable again. Even if the model could be solved, a doctor may not have the time to wait until the end of the evaluation to make the first decision. Similar tight temporal constraints can appear in problem domains such as the response to natural disasters, to failures in industrial production plants, or to uncertain events in autonomous navigation.

The current evaluation algorithm for UIDs[8] is a dynamic programming algorithm that starts computing an optimal policy for the last decision and moves backwards in time until it reaches the first decision. The algorithm starts spending effort on calculating a policy for a distant decision with an enormous space for the past, a task which will decrease considerably in size when you actually approach the point of the decision. If the algorithm is stopped prematurely, it is because the decision maker needs to take the first decision. In that moment the prescription given by the algorithm for the first decision does not give any clue on what to do first. Thus, the result is not satisfactory for a decision maker impatiently awaiting advice.

Although several authors (see, e.g.,[2,4]) have proposed algorithms for finding a trade-off between the computational cost and the quality of the solution in decision making situations under time constraints, to the best of our knowledge none of these contributions address the problem of providing informed advice on the *first* decision(s)[1].

In this paper we address the problem of providing informed any-time advice on the first decisions in IDs and UIDs. The algorithm provides a solution whenever it is stopped, and given sufficient time it will eventually provide a correct solution. To address this problem, the proposed anytime algorithm starts with the first decision and works its way forward in time. Due to the nature of the problem, you cannot be sure of the policy for the first decision before the entire problem has been solved. However, the algorithm will over time gradually improve the probability of choosing the best decision. We also provide a framework for analyzing the performance of the algorithm as well as experimental results using randomly generated UIDs, which demonstrate the feasibility of the approach.

2 Unconstrained influence diagrams

UIDs[8] generalizes IDs[6] in order to represent and solve decision problems in which the order of decisions is not linear, and for which the decision maker is interested in the best

[1]In Section 4 we will explain why we study the first decisions instead of not only the first decision.

ordering as well as an optimal choice for each decision. Since UIDs contains IDs as a subclass, we will in this section focus on describing the representation and solution of UIDs.

2.1 The representation language

We start by considering an example adapted and simplified from MEDIASTINET, a real-world ID for the mediastinal staging of non-small cell lung cancer (NSCLC).[9, 10] A physician is trying to decide on a policy for treating patients (*Tr*) suffering from NSCLC. After an initial CT scan of the patient (*CT*), the physician has to classify the tumor (*N2_N3*) to find out whether there is metastasis. The physician can decide to perform a transbronchial needle aspiration test (*TBNA?*) and/or a positron emission tomography test (*PET?*) and/or an endobronchial ultrasound test (*EBUS?*), which will produce the test results *TBNA*, *PET*, and *EBUS*, respectively. Note that the order in which the tests are performed is not specified, that the result of a test is only available if the physician decides to perform the corresponding test, and that each test can be performed at most once.

To represent this simplified problem by an ID we have to represent the unspecified ordering of the tests as a linear ordering of decisions. Thus, the ID has to include three test decisions (T_1, T_2, and T_3) and three result-nodes (R_1, R_2, and R_3). Each test decision has four states (*tbna*, *pet*, *ebus* and *no-test*) and each of the result nodes has seven states, pos_{tbna}, neg_{tbna}, pos_{pet}, neg_{pet}, pos_{ebus}, neg_{ebus}, and *no-result.* Fig. 1 shows the resulting ID representation. Not only does the ID representation obscure the structure of the decision problem, but for large decision problems this modeling technique will also be prohibitive as all possible scenarios should be explicitly encoded in the model.

In the UID framework, the combinatorial problem of representing non-sequential decision problems is postponed to the solution phase. A UID representation for the problem above is shown in Fig. 2. It is clear that this representation is more compact and un-

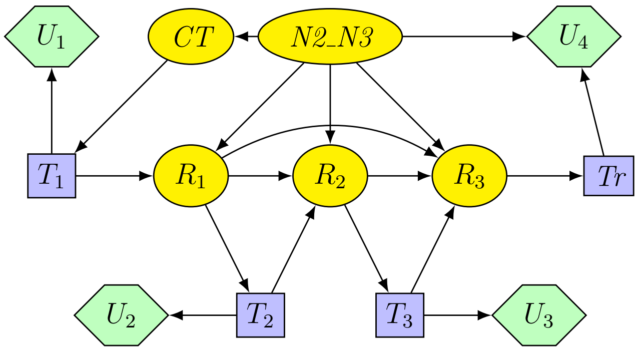


Figure 1: An ID representation for a simplified and adapted problem of mediastinal staging of non-small cell lung cancer.

derstandable than the ID in Fig. 1. Moreover, if we had not simplified the problem of Mediastinet and we had included all the tests in the UID, then the representation would still be affordable because for each possible new test to be performed we only need to add one decision node (representing the choice of performing the test) and one node representing the result of the test.

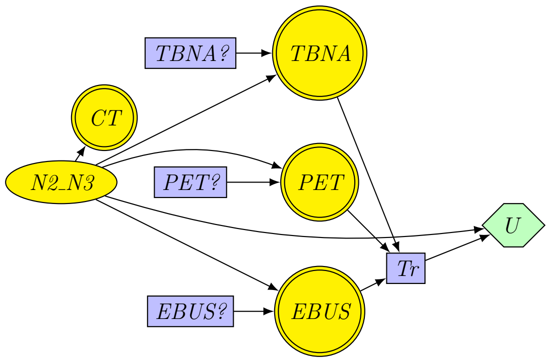


Figure 2: A UID for a simplified and adapted problem of mediastinal staging of non-small cell lung cancer.

An *unconstrained influence diagram* (UID) is a directed acyclic graph (DAG) over three sets of nodes: decision nodes (rectangles) $\mathbf{V}_D$, chance nodes $\mathbf{V}_C$, and utility nodes (diamonds) $\mathbf{V}_U$. Chance nodes can be of two types, *observable* (double-circles) and *non-observable* (single-circles). We require that utility nodes have no children. We will use the terms node and variable interchangeably when this does not cause any confusion.

The quantitative information associated with a UID is given by (1) assigning to each chance node C a probability distribution $P(C \mid \text{pa}(C))$ for C given each configuration of its parents $\mathrm{pa}(C)$, and (2) assigning to each utility node U a utility function ψ_U that maps each configuration of the parents of U into a real number. We assume that the utility functions are combined additively into a joint utility function ψ.

The semantics of the links are similar to the semantics for IDs. However, as opposed to IDs, a total ordering of the decision nodes is not required. While non-observable variables are variables that will never be observed, an observable variable will be observed when all its antecedent decision variables have been decided.

The structural specification of a UID yields a partial temporal order. If the partial order is extended to a linear order we get an ID. Such an extended order is called an *admissible order*.

2.2 UID Solution

Solving a UID means calculating an optimal strategy, formed by a set of *step-policies* and a set of *decision-policies*. To organize the computations, we work with a secondary computational structure called an S-DAG, which is a DAG representing relevant admissible orderings (see Fig. 3).

A *step-policy* for a node N in an S-DAG is a rule that based on the current history $\mathrm{hst}(N)$ specifies which of its children $\mathrm{ch}(N)$ to go to. As the policy needs not be deterministic, we formally define a step-policy for node N as a conditional probability distribution $P(\mathrm{ch}(N) \mid \mathrm{hst}(N))$. A *decision-policy* for a decision node D in an S-DAG is a probability distribution $P(D \mid \mathrm{hst}(D))$. A *strategy* for an S-DAG consists of a step-policy for each node and a decision policy for each decision.

An optimal strategy for an S-DAG is a strategy that maximizes the expected utility. To be sure that an optimal strategy for a particular S-DAG is in fact a solution to the decision problem specified by a UID the S-DAG must contain every ordering that can be part of an optimal strategy. Such S-DAGs are called GS-DAGs (see Fig. 3).

Jensen and Vomlelová[8] proposed to solve a UID by constructing a GS-DAG and to

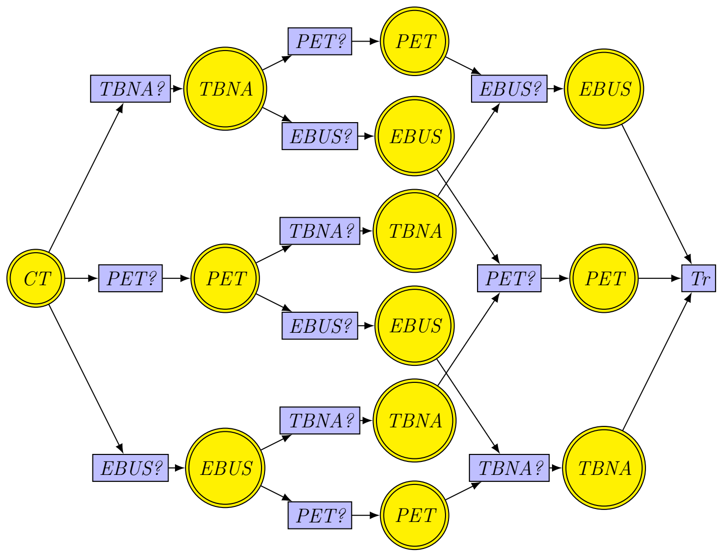


Figure 3: A GS-DAG for the UID of Fig. 2.

solve it through dynamic programming, eliminating the variables in reverse temporal order. The evaluation starts by allocating the sets of probability and utility potentials into the *sink* node and proceeds backwards by following the links in the S-DAG. Chance and decision variables are eliminated as in variable-elimination for IDs:[11] chance variables are eliminated by sum-marginalization, and decision variables are eliminated by max-marginalization. When the algorithm finishes the computations on a node then it transfers the sets of potentials to its parents. When several branches meet, the probability tables will be identical and the utility tables will be combined through maximization.

3 An anytime algorithm for unconstrained influence diagrams

In general, the basic idea with an anytime algorithm is that time constraints may cause the user to be unable to wait for the standard solution algorithm to finish. Thus, it should be possible to stop the algorithm at any time, and the algorithm should then provide an approximate solution. With this requirement we may settle for an algorithm that may take longer than the standard algorithm, but which in the mean time can provide a better approximate solution than the standard algorithm.

Unfortunately, the decision maker cannot accept an anytime strategy where the policy for the first decision is completely uninformed. The decision maker needs to know what to do first. Therefore, the aim of an anytime algorithm for solving UIDs (or decision graphs in general) is to provide more and more informed advice on what to do first.

The standard evaluation algorithm for UIDs[8] provides a strategy by solving the problem in reverse temporal order. If the algorithm is stopped prematurely, it can provide a strategy, which consists of choosing completely randomly for the decisions which have not yet been dealt with, and to follow the calculated optimal policies for the last decisions. In this way, it can be said that you have an anytime algorithm: it provides a strategy whenever it is stopped, the expected utility of the strategy never decreases over time, and, eventually, the algorithm provides an optimal strategy. However, this is not satisfactory. If the user stops the algorithm prematurely, it is because she needs to take the first decision, but the algorithm does not give her any clue on what to do first.

To address the problem of providing informed advice on the first decision, we propose a forward search performed in a decision tree[16] representation of the UID. The tree is built from the root towards the leaves. Our algorithm is inspired by AO* heuristic search algorithms.[13] The search process can be accommodated by considering the decision tree as an AND/OR (AO) graph, where chance nodes are AND nodes, and decision nodes and branch nodes are OR nodes. Thus, a solution is a hyper-graph G such that every node X in G satisfies: (1) if X is an AND node, then all the children of X are in G; (2) otherwise (X is an OR node) exactly one child of X is in G. We will use the terms decision tree and AO graph interchangeably in this section.

The algorithm keeps a list of triggered nodes (the current leaves in the tree constructed so far) as candidates for expansion. A triggered node X is *expanded* by adding its children $\mathrm{ch}(X)$ to the tree and calculating the expected utility of the path from the root to X using a *heuristic function* to estimate the MEU obtainable at the children of X. The heuristic function described in Section 3.3 involves belief updating in the BN underlying the UID. However, as our experiments show, although BN belief updating is a considerable part of the time complexity of our algorithms, it does not cause any complexity problems.

The main results of our experiments can be summarized as follows. Dynamic programming is faster than tree search when the purpose is to compute the optimal strategy for the entire decision problem. However, the tree search algorithm fulfills the purpose for which it is designed: it improves the recommendation for the first decisions of the problem when time increases, as opposed to dynamic programming, which provides completely uninformed recommendations until it has terminated.

3.1 A Search Based Solution Algorithm

A UID can be converted into a decision tree, which in turn can be used as a computational structure for solving the corresponding decision problem. A decision tree is a rooted tree in which the leaves are utility nodes and the non-leaf nodes are either decision nodes (square shaped) or chance nodes (circular shaped). The decisions on the possible orderings are made explicit in the model by partitioning the decision nodes into either ordinary decisions or branching point decisions.

The past of a node X, denoted by $\text{past}(X)$, is the configuration specified by the labels associated with the arcs on the path from the root to X; if X is a value node then $\text{past}(X)$ is called a *scenario*. The decision tree represents each scenario in the decision problem explicitly; hence the size of the tree can grow exponentially in the number of variables. The size can, however, be reduced by collapsing identical sub-trees, a procedure also known as *coalescence*.[14]

The quantitative part of the decision tree consists of probabilities and utilities. Each arc from a chance node A is associated with a probability $P(A = a \mid \text{past}(A))$, where $A = a$ is the label of the arc. The probabilities can be found from the underlying Bayesian network model encoded by the UID. With each value node V in the decision tree, we associate the utility $\psi(\text{past}(V))$ of the scenario $\text{past}(V)$.

Instead of building the decision tree in full and solving it using the "average-out and fold-back" algorithm,[16] we propose to build the tree from the root toward the leaves. A heuristic function h should provide an estimate of the maximum expected utility (MEU) obtainable at every node in the decision tree. Thus, at any point in time we have a *partial decision tree* in which the heuristic can be used to estimate the MEU at the leaf nodes. These estimates can in turn be propagated upward in the tree giving an estimate of the MEU of the nodes in the explored part of the tree, and in particular, an estimate of the optimal policies for the decision nodes in this part.

A collection of optimal policies for a subset of the decision nodes is called a *partial strategy* Δ', and a partial strategy based on a heuristic function is called a *partial heuristic strategy* $\hat{\Delta}'$. Clearly, the better the heuristic function is at estimating the MEU of the triggered nodes in the partial decision tree the closer $\hat{\Delta}'$ will be at Δ'. The *uniform extension* of a partial strategy S is a full strategy obtained by assigning random policies to the decision nodes in the unexplored part of the tree.

3.2 Performing the Search

The search/construction of the coalesced decision tree starts with the tree consisting of a single root node. From this the method iteratively expands a node consistent with the UID specification.

When a node is expanded, its outgoing links are added to the decision tree as well as any successor node not already in the tree; the node to be expanded is always selected among the triggered nodes/leaves. When the new nodes are triggered, the expanded node is removed from the triggered set. When a node is added to the decision tree, a heuristic estimate of the MEU for that node is calculated. The values are propagated upwards, thereby possibly updating the current partial heuristic strategy.

The choice of which node to expand is non-deterministic. We have experimented with four selection schemes and we eventually selected the one providing the best results: to expand a node of the lowest depth, i.e., perform a breadth-first search.

In summary, a triggered node X at the lowest depth is selected for expansion, and a heuristic function is used to estimate the MEU of all triggered nodes, i.e., the leaves in the partial decision tree. Thus, at any time during the search we have a partial decision tree for which a heuristic based strategy can be computed.

Algorithm 1 presents the pseudo-code for the basic search procedure. The algorithm maintains a priority list, called *triggered*, where nodes are ordered by increasing depth in the decision tree. We denote with *reachable* the set of nodes in the decision tree formed by the root node and its descendants in the partial solution. The set of *reachable* nodes can change as a result of the corresponding updating procedures. The list *solved* contains all the nodes whose exact MEU value has been computed. The calculation of the heuristics in Line 7 and the process of updating upwards in the tree in Line 9 are explained in Section 3.3.

Algorithm 1 Search of the optimal strategy

```
1: Initialize triggered to the source node
2: while $triggered \cap reachable \neq \emptyset$ do
3:     $X$ := Extract the first node from triggered that is in reachable
4:     if $X \notin solved$ then
5:         Create successors of $X$ (if necessary) and add links from $X$
6:         for all newly created successors $W$ do
7:             CalculateHeuristics($W$) (see Algorithm 2)
8:             Add $W$ to triggered
9:         UpdateUpwards($X$) (see Algorithm 3)
```

3.3 Selecting a Heuristic Function

The choice of heuristic function not only determines the policies being computed, but it may in fact also be used to prune irrelevant parts of the tree thereby reducing complexity. We denote with MEU(X) the MEU of the node X in the decision tree, calculated when the sub-tree defined by X has been explored; we will omit the argument X from MEU(X)

when it is clear from the context. A special class of heuristic functions are the so-called admissible heuristic functions.

3.3.1 An Admissible Heuristic

A heuristic function h is *admissible* if $h(X) \geq \text{MEU}(X)$, for any node X in the decision tree. An admissible heuristic can be exploited during the search: Consider a decision node with two children X and Y. If the sub-tree defined by X has been explored and $h(Y) \leq \text{MEU}(X)$, then we need not explore the sub-tree rooted at Y.

Obviously, we would like the heuristic function h to define a tight upper bound on the expected utility, and relative to the computational complexity of solving the decision tree we would also like for h to be easy to compute.

A possible admissible heuristic function could be

$$h_U(X) = \max_{l \in \mathcal{L}} \psi(\text{path}(X, l)), \tag{1}$$

where $\mathcal{L}$ is the set of leaf nodes in the sub-tree rooted at X and $\psi(\text{path}(X, l))$ is the sum of the utilities associated with l and the path from X to l. Heuristic h_U can be efficiently calculated by max-marginalizing out the variables appearing in the domains of the utility potentials.

Unfortunately, preliminary experiments have shown that the estimation given by h_U can be very far from the MEU. For certain UIDs the estimated optimal policy for the first decision failed to stabilize over time, and in fact a random policy would on average have provided a similar solution in terms of expected utility. Since we have not been able to define an alternative computationally efficient admissible heuristic, we have instead been looking for a nonadmissible heuristic.

3.3.2 A Nonadmissible Heuristic

The heuristic h_U yields a very loose bound on the expected utility. However, since it provides an upper bound, we can combine it with a lower bound to derive a good approximation of the expected utility.

As a lower bound, we use the expected utility of the uniform extension of the current partial heuristic strategy. This heuristic can be efficiently calculated by sum-marginalizing out the variables in the utility and probability potentials.

If we denote the lower bound heuristic by h_L, then we have that if all the variables in the future of node X, denoted by future(X), are chance variables, then $h_L(X) = \text{MEU}(X)$. Furthermore, when the number of decision nodes in future(X) increases the difference $\text{MEU}(X) - h_L(X)$ will also increase. The opposite holds for the heuristic $h_U(X)$.

In order to derive a heuristic closer to $\text{MEU}(X)$, we define the non-admissible heuristic h as a weighted linear combination of h_L and h_U:

$$h(X) = w_L(X)h_L(X) + w_U(X)h_U(X), \tag{2}$$

where $w_L(X) = \alpha(X) \cdot k_X \cdot c(X)$ and $w_U(X) = \alpha(X) \cdot d(X)$. Here $c(X)$ and $d(X)$ are the number of chance and decision nodes in future(X), respectively, and $\alpha(X)$ is a normalizing factor ensuring $w_L(X) + w_U(X) = 1$. By varying the parameter k_X between 0 and $+\infty$, you can achieve any desired mixture of conservatism and optimism as defined by the two heuristics.

3.4 Heuristic algorithms

We have two solution algorithms: breadth first search with admissible heuristic (BF-A) and with non-admissible heuristic (BF-N). We assume that both algorithms use the same data structures for storing the heuristic values of each node. So, in both algorithms each node X has associated three heuristic values: $h_L(X)$, $h_U(X)$ and $h(X)$. Although an implementation of BF-A would only need to store two values, we assume it also stores the three heuristic values because that allows using the same pseudo-code for BF-A and BF-N. Algorithm 2 describes the pseudo-code for computing the heuristics for a node X. At the end of its computations, Algorithm 2 also adds X to the list *solved* when it detects that no further exploration is necessary at the sub-tree rooted at X.

Algorithm 2 CalculateHeuristics

```
Require: A node $X$ of the search tree and an algorithm $\gamma$ ("BF-A" or "BF-N")
Ensure: Calculates the values of $h_L$, $h_U$ and $h$ for $X$, and considers adding $X$ to solved
 1: $c :=$ number of chance nodes in future($X$)
 2: $d :=$ number of decisions in future($X$)
 3: Calculate $h_U$ as in Equation 1
 4: if $\gamma =$"BF-A" then
 5:     $h := h_U$
 6:     if $c = 0$ then
 7:         $h_L := h_U$
 8:     else
 9:         $h_L := 0$
10: else
11:     Calculate $h_L$ and $h$ as in Section 3.3.2
12: if $c = 0$ or ($d = 0$ and $\gamma =$"BF-N") then
13:     Add $X$ to solved
```

Algorithm 3 describes the procedure for propagating any change in the heuristic values of a node X upwards in the tree. It updates the heuristic values for X based on the heuristic values of its children and using the standard algorithm for decision trees.[16] Thus, if X is a chance node, then it calculates a weighted sum of the heuristics for the children of X using the probabilities as weights. If X is a decision, then the algorithm maximizes over the heuristics for the nodes in ch(X).

If the value of some heuristic in X has changed, then Algorithm 3 attempts to prune any children of X that cannot improve the current best solution. In this way, a child W of a node X can be pruned if $h_L(X) > h_U(W)$, i.e., the call *isPrunable*(X, W) in Line 8 returns true.

Algorithm 3 UpdateUpwards

```
Require: A node $X$ of the search tree
Ensure: Updates heuristics for $X$, prunes children if possible, and propagates changes upwards
 1: for all heuristic $f \in \{h_L, h_U, h\}$ do
 2:    if $X$ is a chance node then
 3:       $f(X) = \sum_{W \in \mathrm{ch}(X)} P(W|X) \cdot f(W)$
 4:    else
 5:       $f(X) = \max_{W \in \mathrm{ch}(X)} f(W)$
 6: if some heuristic in $X$ has changed then
 7:    for all $W \in \mathrm{ch}(X)$ do
 8:       if $isPrunable(X, W)$ then
 9:          Remove the link $X \rightarrow W$
10:    if $\mathrm{ch}(X) \subset solved$ then
11:       Add $X$ to $solved$
12:    for all $W \in parents(X)$ do
13:       UpdateUpwards($W$)
```

3.5 Example

Consider the UID of Fig. 4a, where the domains of the variables X and Y are $\{a, \neg a\}$, and the domains of D and E are $\{yes, no\}$. Let $P(X = a|D = yes) = P(X = \neg a|D = no) = 0.6$, and $P(Y = a\,|E = yes) = P(Y = \neg a\,|E = no) = 0.9$. Let the utility function of U be:

$$\psi_U(d, x, e, y) = \begin{cases} 10.0\,, & \text{if } x = y \wedge (d = yes \vee e = yes) \\ 0.0\,, & \text{otherwise.} \end{cases}$$

Let the cost function attached to D, $\psi_D(d)$, be 1 if $d = yes$, and -1 otherwise. Let the cost function attached to E be $\psi_E(e) = 0$.

The GSDAG of the UID of Fig. 4a is shown in Fig. 4b. Assume that we execute the algorithm BF-N, with $k_V = 2.0$ for every node V. The algorithm starts by exploring the AO graph by following the GSDAG. When the two children of *Source* in the GSDAG (D and E) have been expanded, we have the AO graph of Fig. 5. Root variable *OD* represents a branch point to decide which decision to make first: D or E. Each leaf node in the AO

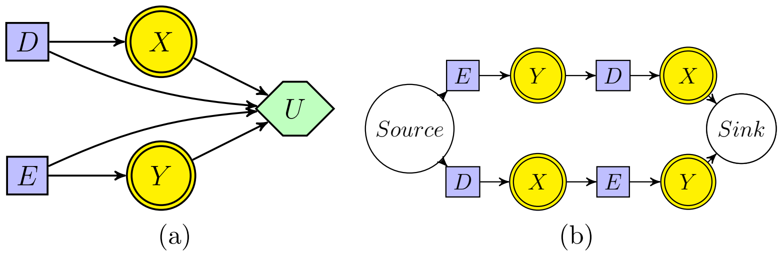


Figure 4: Small UID example (a) and its GS-DAG (b).

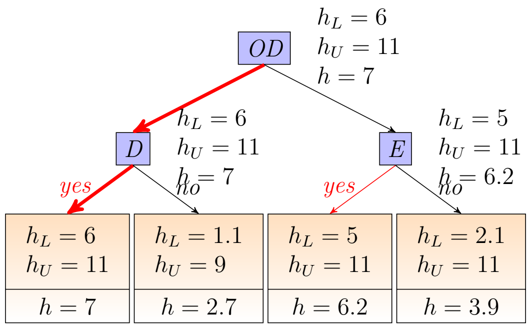


Figure 5: AO graph after expanding the two children from the root: the nodes D and E.

graph is represented by an orange-shaded rectangle containing the lower and the upper bounds, h_L and h_U as well as the heuristic h calculated as in Equation 2. Each inner node is associated with its heuristic values, h_L, h_U, and h, calculated from the corresponding values associated with its children. Links starting at chance or decision nodes are labeled with the corresponding states; links from chance nodes are additionally labeled with the associated probability (see Fig. 6). Links colored in red indicate the partial solutions to the problem; thicker red links represent the partial optimal strategy at that moment of the evaluation.

When the algorithm stops, we obtain the AO graph of Fig. 6; when the heuristic values, h_L, h_U and h, of a node X are equal, then we just denoted these three heuristics as h. Pruned arcs are indicated as dashed lines. The algorithm did not have to expand all the nodes in order to obtain an optimal strategy because it took advantage of the pruning mechanisms. For example, consider the node labeled with X in Fig. 6 given by the path $OD = D$, $D = no$; its upper bound value is lower than the lower bound of its parent. In this case, the sub-tree rooted at X can be pruned as exploration of it would not improve

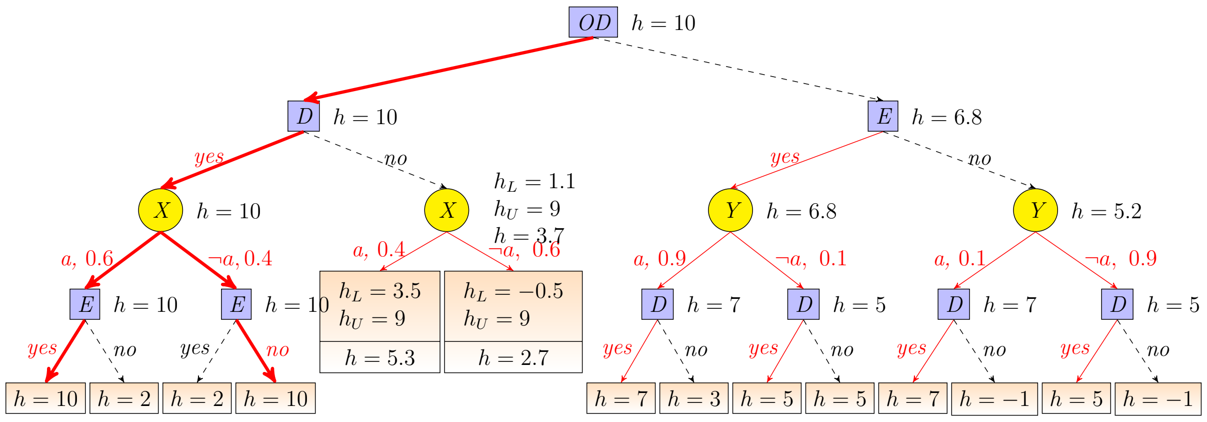


Figure 6: Final AO graph with the solution to the UID example.

the heuristic values of its parent node D. We also note that the algorithm expanded 4 levels of nodes, and did thus avoid expanding the last level of the complete decision tree.

3.6 Updating k

In order to choose a good value for k_X in BF-N algorithm, we propose to update it automatically as the tree is expanded. The procedure of updating upwards in the tree (Algorithm 3) is based on the expectation that the heuristic is more precise the closer we get to the leaves. Thus, when $h(X)$ has been updated, treating $\widehat{\mathrm{EU}}(X) = h(X)$ as an accurate estimate of the expected utility for X we calculate a new value for k_X:

$$k_X := \frac{(\widehat{\mathrm{EU}}(X) - h_U^0(X))d(X)}{(h_L^0(X) - \widehat{\mathrm{EU}}(X))c(X)}.$$

where $h_L^0(X)$ and $h_U^0(X)$ are the initial values of h_L and h_U in X when this was created. Note that k_X will always be non-negative. Only the nodes in *triggered* $\cap$ *ch*(X) will use k_X when they are selected for expansion; thus, when X has no triggered children then updating k in X would have no effect.

4 Framework for analyzing performance

We describe here a framework for analyzing the performance of anytime algorithms when providing informed advice on the first decisions. We use as baseline the uniform extension of the strategy, and put the anytime strategy in relation to the difference in value between the uniform extension and the optimal strategy. We advocate the use of the uniform extension as baseline because it is a strategy that the decision maker always has available.

We study the performance for the first n decisions, where $n \in \mathbb{N}$. We have chosen a value of n greater than 1 because the initial decision in the UIDs generated is always to choose the first decision among a set of unordered decisions, but we also wanted to study the selection of the decision option for the decision initially chosen. Moreover, it also allows studying the scenario in which several decisions have to be made quickly. Note that a partial strategy can at any time be extended to a full strategy $\widehat{\Delta}(t)$ by assigning random policies to the decision nodes in the unexplored part of the tree.

4.1 Evaluation measures

The necessity of having a full strategy covering all decisions can be relaxed in some situations. The decision maker may need the help of an anytime algorithm to obtain a prescription for only the first n decisions in the decision tree[2]. The remaining part of the decision problem may be evaluated by using dynamic programming. This anytime strategy, only requiring prescriptions for the first n decisions, is denoted $\widehat{\Delta}^n(t)$.

Expected utility of the anytime strategy The goodness of an anytime algorithm can be measured by considering the expected utility (EU) of the strategy found. The results for $\widehat{\Delta}^n(t)$, for varying n, will generally be different, but two interesting properties relate them. If MEU is the maximum expected utility of the UID and m is the maximum

[2]We talk about number of decisions in the decision tree instead of referring to the UID. This is because the decision nodes in the decision tree not only represent decision nodes in the UID but also branching points.

number of decisions in a path from the root to a leaf in the decision tree, then, for every instant time t, we have that $MEU = EU(\widehat{\Delta}^0(t))$ and $EU(\widehat{\Delta}^{i-1}(t)) \geq EU(\widehat{\Delta}^i(t))$, for each $i \leq m$. We will for convenience denote $EU(\widehat{\Delta}^i(t))$ by $EU^i(t)$.

Expected number of correct decisions We can measure the goodness of the decisions made at any level of the decision tree by comparing them with those prescribed under the optimal strategy. Given that all the variables on the same level of the decision tree are of the same kind, we define a *layer* as a level of the decision tree formed by the decisions on the same level. Let n be a natural number between 1 and the maximum number of decisions appearing on a path from the root to a leaf in the decision tree. Let t be an instant time and let $L^n(t)$ be the set of pairs (X, x), where X is a decision node at the n-th layer at time t and x is the decision option selected at X at time t. We say the pair (X, x) *is correct* at time t if the selected decision option x is optimal in the scenario defined by past(X). Let $C^n(t)$ be the subset of pairs of $L^n(t)$ that are correct at t. The *expected proportion of correct decisions at the n-th layer at time t*, denoted by $\mathit{CorrectAtLayer}^n(t)$, is defined as:

$$\mathit{CorrectAtLayer}^n(t) = \frac{|C^n(t)|}{|L^n(t)|} . \tag{3}$$

The *expected number of correct decisions in the first n layers*, denoted by $\mathit{CorrectFirstLayers}^n(t)$, is defined as:

$$\mathit{CorrectFirstLayers}^n(t) = \sum_{i=1}^{n} \mathit{CorrectAtLayer}^i(t) .$$

$\mathit{CorrectFirstLayers}^n(t)$ measures the expected number of correct decisions by following a single path through the first n layers of decisions.

4.2 Normalization of the measures

We want to study whether our anytime algorithm provides a policy for the first decision(s) that is better than the one proposed by the DP algorithm, which is assumed to be random for the first decisions. However, summarizing the experimental results requires combining the results of all the UIDs evaluated. This combination requires a normalization of the evaluations of all UIDs as well as a normalization of the time used by the algorithm. For the latter part of the normalization, this will be achieved by using DP as reference point.

Each normalized measure is represented in a two-axis graph. The X-axis represents the time of the evaluation, interpreted as the fraction of the time spent by DP. The Y-axis represents the percentage of improvement (in the range from 0 to 1) provided by the anytime strategy compared to a uniform strategy. By normalizing the results, random policies are given the value 0 while an optimal strategy is given the value 1.

We assume that the evaluation of a UID r with an algorithm γ begins at time 0 and is stopped at time $\tau_{\gamma,r}$. We also assume that DP is always stopped when its evaluation finishes. Let $f_{\gamma,r}(t)$ be a function over time, corresponding to the result of evaluating UID r with algorithm γ. The *normalization* of $f_{\gamma,r}(t)$, denoted by $f'_{\gamma,r}(t)$, is defined by:

$$f'_{\gamma,r}(t) := \frac{f_{\gamma,r}(t \cdot \tau_{DP,r}) - f_{DP,r}(0)}{f_{DP,r}(\tau_{DP,r}) - f_{DP,r}(0)} \,. \tag{4}$$

Note that $f'_{DP,r}(0) = 0$ and $f'_{DP,r}(1) = 1$. When we have a set of UIDs $\mathbf{I}$, a summary function $f''_{\gamma}(t)$ is calculated by taking the mean of $f'_{\gamma,r}(t)$ over all the UIDs $r \in \mathbf{I}$.

For example, for *CorrectFirstLayers*$^3(t)$, if we assume that all the decisions are binary, a result of 0 in the normalized score is achieved when the expected number of correct decisions in the first three layers is 3/2. This is the result we should expect from having random policies.

5 Experiments

We have performed a series of experiments for analyzing the performance of three anytime algorithms: the dynamic programming-based (DP) and a breadth first search with both an admissible heuristic (BF-A) and a non-admissible heuristic (BF-N). The first problem we faced is that we do not have real-world examples of UIDs and the probabilistic graphical models repositories available on the Internet do not contain UIDs. For this reason, we have randomly generated UIDs. We have solved the UIDs generated with exact algorithms to obtain exact solutions and to compare them with the solutions returned by the proposed algorithms.

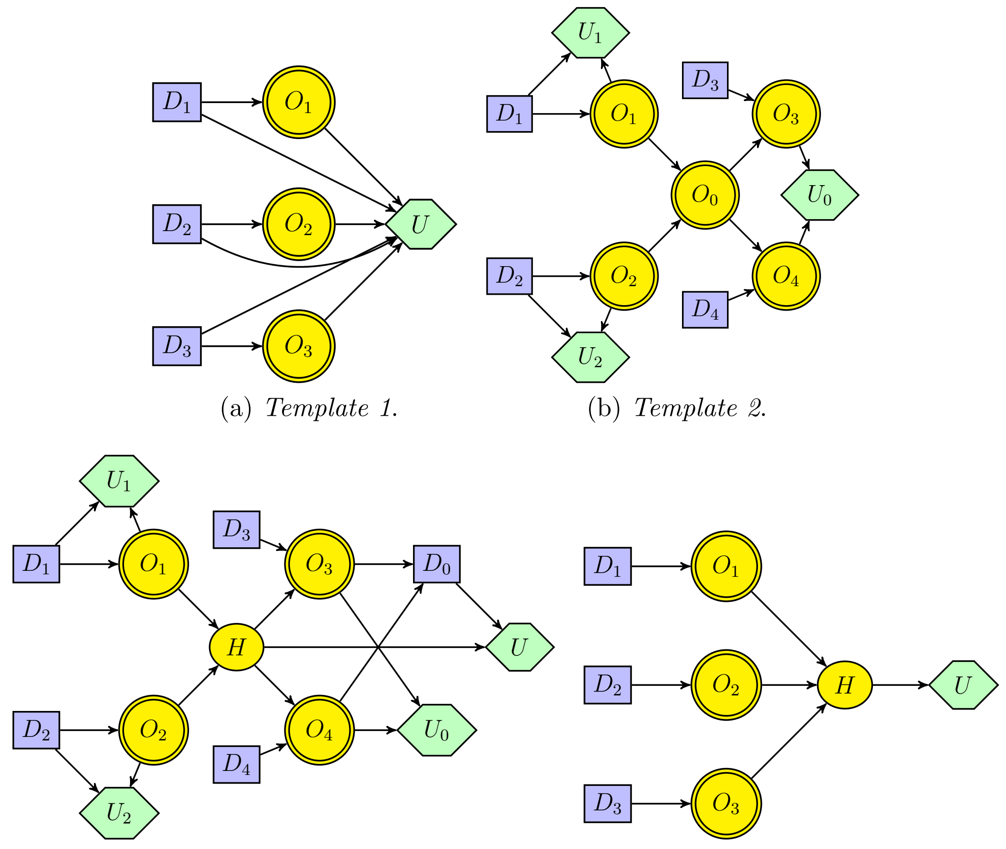
5.1 Generation of UIDs

Rather than generating completely random UIDs,[19] we follow a different approach and generated models that share characteristics with structures we would expect to find in real-world domains.

The generation of UIDs was divided into two phases. First, we obtained the structure of UIDs (arcs and nodes, including their type). Second, we generated the probability and utility potentials.

Each UID structure is created by instantiating a parameterized *template* described by a set of parameters. Templates 1 and 4 (see Fig. 7a and Fig. 7d) are instantiated by parameter n, the number of decision nodes. Template 2 (Fig. 7b) is instantiated by two parameters: n_1, the number of ancestor decision nodes of O_0; and n_2, the number of decision nodes that are not ancestors of O_0. Template 3 (Fig. 7c) is instantiated by two parameters: n_1, the numbers of ancestor decision nodes of H; and n_2, the number of decision nodes different from D_0 that are not ancestors of H.

The structure of the templates represent patterns that are likely to be found in real problems with a partial order of decisions. For example, the UID example of *Template 3* (Fig. 7c) corresponds to a medical decision problem, where the unobservable variable H represents a disease, D_1 and D_2 indicate two vaccinations for H, and D_3 and D_4 represents medical tests. Variables O_1 and O_2 represent the body's reaction to each vaccination; variables O_3 and O_4 represents the tests results. Decision D_0 represents the decision about the treatment.



(a) *Template 1.* (b) *Template 2.*

(c) *Template 3.* (d) *Template 4.*

Figure 7: Template examples.

The values assigned to the template parameters influence the complexity of the evaluation. One measure of the complexity of the evaluation of a UID is the number of paths in the GS-DAG. In general, the number of paths in the GS-DAG for UIDs generated according to templates 1 and 4 is $n!$, while in the case of Templates 2 and 3 it is $n_1! \cdot n_2!$, with n, n_1 and n_2 being the instantiation parameters.

After creating the structure of a UID, we randomly generated a probability table for each chance node (both observables and non-observables) and a utility table for each utility node.

Table 1: Summary of $EU^i(t)$ and $\mathit{CorrectFirstLayers}^i(t)$ (short-hand CFL) for the BF-A algorithm (left columns) and for the BF-N algorithm (right columns).

	BF-A				**BF-N**			
	25 %	**50 %**	**75 %**	**100 %**	**25 %**	**50 %**	**75 %**	**100 %**
$EU^1(t)$	-3	-4	-3	5	54	61	56	59
$EU^2(t)$	43	65	75	72	64	75	89	94
$EU^3(t)$	36	50	59	60	61	72	83	87
$\mathit{CFL}^1(t)$	1	-2	-2	3	48	52	48	53
$\mathit{CFL}^2(t)$	13	18	21	24	46	54	56	61
$\mathit{CFL}^3(t)$	9	12	15	17	34	41	44	49

5.2 Experimental results

We have created UIDs by varying the instantiating template parameters. In Template 1, n has been varied from 3 to 6. In Templates 2 and 3, we have set n_2 equal to n_1 and have varied n_1 from 2 to 4. In Template 4, we have varied n from 3 to 5. For each UID graph, we have randomly created 50 different realizations by randomly generating the numbers in the probability and utility potentials. This amounts to a total of 650 UIDs.

The algorithms were implemented in.[1] The experiments were conducted on a PC with Intel Core 2 CPU @ 2.4 GHz with 2 GB of memory and using Java 6.0.

The results of the experiments are summarized in Table 1, $EU^i(t)$ and $\mathit{CorrectFirstLayers}^i(t)$ correspond to the two measures described in Section 4.1. In particular, $\mathit{CorrectFirstLayers}^1(t)$ denotes the frequency of selecting the best initial decision (i.e., a branching point decision). Each cell represents the percentage of improvement of the anytime algorithm compared to a uniform strategy.

For example, $\mathit{CorrectFirstLayers}^3(t)$ accumulates the number of correct decisions at the first 3 decision levels in the tree. Consider now the table by applying the inverse normalization function for, say, the cell defined by column 25% and row $\mathit{CorrectFirstLayers}^3(t)$ in Table 1 and assume for simplicity that all decisions are binary, including the branching points.[3] The percentage value, 34, in the cell is the result of the normalization

[3]The assumption that all decisions are binary is made to simplify the explanation and does, in general, not reflect the UIDs that have been generated for the experiments.

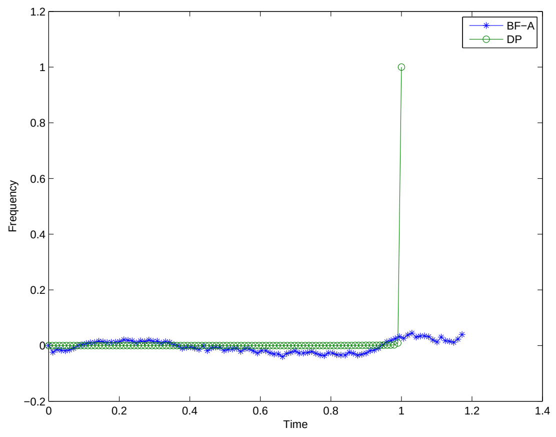
$3/2 + (3 - 3/2) \times 34/100 = 2.01$ (see Equation 4). Thus, on average, 2.01 of the first three decisions on any single path in the tree will be selected correctly by BF-N using only 25 percent of the time spent by DP.

Fixing the time instant in Table 1, we can observe the tendency of $\mathit{CorrectFirstLayers}^i(t)$ and $EU^i(t)$ when i increases. As BF-A and BF-N perform a breadth first search, we would expect that the lower the i the better the values in the table because the algorithm may not have enough time for computing the policies for the higher levels in the tree.

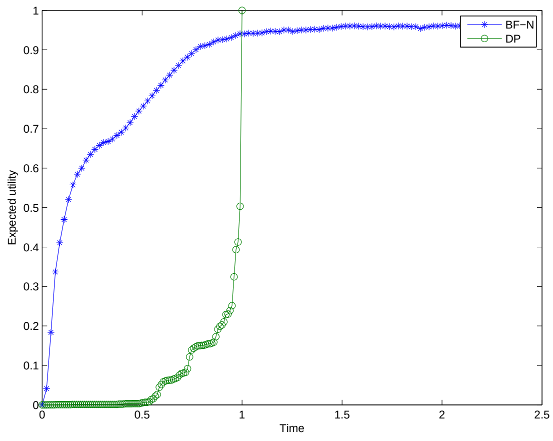
However, an increase in the value from $\mathit{CorrectFirstLayers}^1(t)$ to $\mathit{CorrectFirstLayers}^2(t)$ contradicts the tendency. The first layer in the UIDs generated corresponds to a branching point for choosing the best decision among typically more than two unordered decisions, while the second layer corresponds to choosing the decision option of a dichotomous decision. Thus, choosing the best decision at a branching point is therefore a more difficult problem than choosing the decision option.

We can extract three main conclusions from Table 1. First, the results of BF-A are not much better than using random policies, as many of the values in the table are close to 0 or even negative. Second, BF-N outperforms BF-A in all the recorded measures. Third, the algorithm BF-N improves over time w.r.t. all the recorded measures and it always gives much better results than using a random strategy.

Fig. 8 exemplifies the behavior of the algorithms. In Fig. 8a we can see how BF-A does not give any clue about the policies for the first two decisions during the entire time spent by the DP evaluation and therefore the partial strategy is very close to the uniform strategy. Fig. 8b shows how DP fails to give informed advice on the policies for the first two decisions until 50 percent of the time has passed. In contrast, the strategy returned by BF-N for the first two layers improves over the time and, although DP computes the optimal strategy faster than BF-N, when DP stops the approximate strategy is near-to-optimal.



(a) Comparison of $\mathit{CorrectFirstLayers}^1(t)$ between DP and BF-A.



(b) Comparison of $EU^2(t)$ between DP and BF-N.

Figure 8: Examples of the algorithms performance over time.

6 Related work

Many authors have investigated the problem of finding a trade-off between the computational effort of evaluating a decision model and the quality of the solution (see, e.g.,[4,5]). However, previous works do not address the problem of finding a qualified anytime policy for the first decision of the decision problem.

Garcia-Sanchez and Druzdzel[2] proposed a Monte-Carlo algorithm that, by using the same samples to rank the decision options, could reduce the variance in estimation of the expected utility. Their concerns about the ranking of the decision options lead us to define the measure $\mathit{CorrectAtLayer}^1(t)$ and subsequently to generalize it for any layer in the decision tree. However, Garcia-Sanchez and Druzdzel[2] did not consider the anytime aspect of the problem and time was therefore not considered a factor when the performance of their algorithm was analyzed.

Several authors have used search of the AO graph for solving an ID.[12,15,20] Instead of using an alternate sequence of AND and OR layers as Qi and Poole,[15] we follow the same approach as Yuan et al.[20] and have a level in the tree for each variable in the UID. This allows us to more efficiently make upwards updates of the heuristic estimates in the tree.

Contrary to Qi and Poole[15] as well as Yuan et al.[20] we use a non-admissible heuristic for guiding the search, while at the same time maintaining an upper bound on the expected utility in order be able to prune the tree. Finally, our method performs a breadth-first search, in contrast to Marinescu[12] and Yuan et al.,[20] in order to explore all the branches in the first levels of the tree at an early stage.

7 Conclusions

We have proposed an algorithm that addresses a problem that we believe has previous been overlooked by proposals of anytime algorithms for solving IDs and UIDs: provide informed anytime advice for the first decisions in the decision problem, thus accommodating the immediate needs of the decision maker. We have proposed a heuristic guided search-based algorithm as well as a framework for analyzing the performance of anytime algorithms for decision models like IDs and UIDs.

From the experimental results we can draw two main conclusions related to the proposed algorithm. Firstly, selecting a good heuristic function is decisive for achieving good performance; employing a simple admissible heuristic offers a response that is not much better than using a uniform strategy. Secondly, the proposed anytime algorithm implementing a non-admissible heuristic balancing a lower and upper bound of the expected utility has demonstrated a significant improvement in the anytime recommendations for the first decisions of the problem, and its prescriptions are considerably better than using a uniform strategy.

Acknowledgments

The first author has been supported by the Department of Education of the autonomous region of Madrid and the European Social Fund (ESF). We would like to thank the Machine Intelligence group at the Department of Computer Sciences at Aalborg University for fruitful discussions, and Marta Vomlelová for giving us access to her UID implementation.

References

[1] Elvira Consortium. Elvira: An environment for creating and using probabilistic graphical models. In J. A. Gámez and A. Salmerón, editors, *Proceedings of the First European Workshop on Probabilistic Graphical Models (PGM'02)*, pages 1–11, 2002.

[2] D. Garcia-Sánchez and M. J. Druzdzel. An efficient exhaustive anytime sampling algorithm for influence diagrams. In A. Salmerón and J. A. Gámez, editors, *Advances in Probabilistic Graphical Models*, pages 255–273. Springer, Berlin, Germany, 2007.

[3] M. Gómez, C. Bielza, J. A. Fernández del Pozo, and S. Ríos-Insua. A graphical decision-theoretic model for neonatal jaundice. *Medical Decision Making*, 27:250–265, 2007.

[4] M. C. Horsch and D. Poole. An anytime algorithm for decision making under uncertainty. In G. Cooper and S. Moral, editors, *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI'98)*, pages 246–255, San Francisco, CA, 1998. Morgan Kauffmann.

[5] E. Horvitz and A. Seiver. Time-critical action: Representations and application. In D. Geiger and P. Shenoy, editors, *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence (UAI'97)*, pages 250–257, San Francisco, CA, 1997. Morgan Kauffmann.

[6] R. A. Howard and J. E. Matheson. Influence diagrams. In R. A. Howard and J. E. Matheson, editors, *Readings on the Principles and Applications of Decision Analysis*, pages 719–762. Strategic Decisions Group, Menlo Park, CA, 1984.

[7] F. Jensen, F. V. Jensen, and S. L. Dittmer. From influence diagrams to junction trees. In R. L. de Mantaras and D. Poole, editors, *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence (UAI'94)*, pages 367–373, San Francisco, CA, 1994. Morgan Kauffmann.

[8] F. V. Jensen and M. Vomlelová. Unconstrained influence diagrams. In A. Darwiche and N. Friedman, editors, *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence (UAI'02)*, pages 234–241, San Francisco, CA, 2002. Morgan Kauffmann.

[9] M. Luque. *Probabilistic Graphical Models for Decision Making in Medicine.* PhD thesis, UNED, Madrid, 2009.

[10] M. Luque, F. J. Díez, and C. Disdier. A decision support system for the mediastinal staging of non-small cell lung cancer. In A. Nicholson, editor, *Proceedings of the Ninth UAI Bayesian Modelling Applications Workshop (BMAW'11)*, Barcelona, Spain, 2011.

[11] A. Madsen and F. V. Jensen. Lazy evaluation of symmetric Bayesian decision problems. In K. Laskey and H. Prade, editors, *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence (UAI'99)*, pages 382–390, San Francisco, CA, 1999. Morgan Kauffmann.

[12] R. Marinescu. A new approach to influence diagrams evaluation. In M. Bramer and R. Ellis and M. Petridis, editor, *Research and Development in Intelligent Systems XXVI*, pages 107–120. Springer London, 2010.

[13] N. J. Nilsson. *Principles of Artificial Intelligence.* Tioga, Palo Alto, CA, 1980.

[14] S. M. Olmsted. *On Representing and Solving Decision Problems.* PhD thesis, Dept. Engineering-Economic Systems, Stanford University, CA, 1983.

[15] R. Qi and D. Poole. A new method for influence diagram evaluation. *Computational Intelligence*, 11:498–528, 1995.

[16] H. Raiffa and R.O. Schlaifer. *Applied Statistical Decision Theory.* John Wiley & Sons, Cambridge, MA, 1961.

[17] R. D. Shachter. Evaluating influence diagrams. *Operations Research*, 34:871–882, 1986.

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

[19] M. Vomlelová. Unconstrained influence diagrams - experiments and heuristics. In *Sixth Workshop on Uncertainty Processing (WUPES'2003)*, Hejnice, Czech Republic, 2003.

[20] C. Yuan, X. Wu, and E. Hansen. Solving multistage influence diagrams using branch-and-bound search. In P. Grunwald and P. Spirtes, editors, *Proceedings of the Twenty-sixth Conference on Uncertainty in Artificial Intelligence (UAI'10)*, pages 691–700, Corvallis, OR, 2010. AUAI Press.