An Anytime Algorithm for Evaluating Unconstrained Influence Diagrams

Manuel Luque
Departmento Inteligencia Artificial, UNED
28040 Madrid, Spain
mluque@dia.uned.es

Thomas D. Nielsen and Finn V. Jensen
Department of Computer Science
Aalborg University, Aalborg, Denmark
{tdn,fvj}@cs.aau.dk

Abstract

Unconstrained influence diagrams (UIDs) extend the language of influence diagrams to cope with decision problems in which the order of the decisions is unspecified. Thus, when solving a UID 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 computes a strategy and 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. Experiments indicate that the proposed algorithm performs significantly better under time constraints than dynamic programming.

1 Introduction

An influence diagram (ID) is a framework for representing and solving Bayesian decision problems with a linear temporal ordering of decisions (Howard and Matheson, 1984). However, in many domains the process of 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, possibly using artificial nodes and states. Examples of such decision problems include troubleshooting and medical diagnosis.

Unconstrained influence diagrams (UIDs) were introduced to represent and solve decision problems of this type (Jensen and Vomlelova, 2002); as a special case this also includes decision problems with a linear temporal ordering of the decisions. 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 (Shachter, 1986; Shenoy, 1992; Jensen et al., 1994; Madsen and Jensen, 1999).

Unfortunately, many real world 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 can be intractably large. As an example, the evaluation of the Ictneo system (Bielza et al., 1999) requires a table with $1,66 \times 10^{14}$ entries and produces a policy with $4,24 \times 10^{7}$ configurations for the first decision.

In this paper we present an anytime algorithm for solving UIDs. The algorithm provides a solution whenever it is stopped, and given sufficient time it will eventually provide a correct solution.

In comparison, the standard evaluation algorithm for UIDs (Jensen and Vomlelova, 2002) is a backward induction algorithm employing dynamic programming like most algorithms for IDs. It starts computing an optimal policy for the last decision and moves backwards in time until it reaches the first decision. If the process is stopped prematurely, the algorithm may provide a policy, however, the prescription for the first decision is completely un-informed. Furthermore, as described above, all effort so far may be spent 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 you consider a situation with a decision maker (DM) impatiently awaiting advice on what to do, he most probably wants to get an informed advice on the first decision rather than receiving detailed prescriptions for the last decisions.

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.

2 Unconstrained Influence Diagrams

UIDs were proposed in (Jensen and Vomlelova, 2002) to represent decision problems in which the order of the decisions is not linear, and for which the DM is interested in the best ordering as well as an optimal choice for each decision.

2.1 The Representation Language

We start considering a very simple example: the diabetes diagnosis problem, introduced in (Demirer and Shenoy, 2001). A physician is trying to decide on a policy for treating patients. After an initial examination of their symptoms (S), the physician has to diagnose whether the patient is suffering from diabetes (D). Diabetes has two symptoms, glucose in urine and glucose in blood. Before deciding on whether or not to treat the patient for diabetes (Tr), the physician can decide to perform a urine test (UT) and/or a blood test (BT), which will produce the test results U and B, respectively. After the physician has observed the tests results (if any) she has to decide whether to treat the patient for diabetes. Observe that the order in which the tests are performed is not specified and that the result of a test is only available if the physician decides to perform the corresponding test.

To represent this problem by an influence diagram we have to represent the unspecified ordering of the tests as a linear ordering of decisions. This can be done by introducing two decision variables to model the first test and the second test, respectively. Unfortunately, the structure of the decision problem is not apparent from the model and for large decision problems this technique will 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 for the diabetes diagnoses problem is shown in Figure 1 (explained below).

More formally, an *unconstrained influence diagram* (UID) is a DAG over three sets of nodes: a set of 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* (circles) and *non-observable* (double-circles), and we require that utility nodes have no children. We will use the terms 'node' and 'variable' interchangeably if this does not cause any confusion.

The quantitative information associated

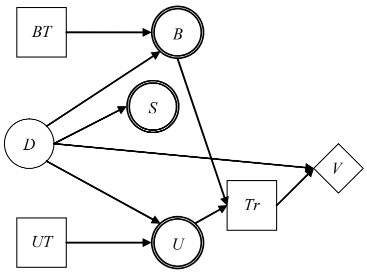

Figure 1: UID for the diabetes diagnosis problem.

with a UID consists of probability distributions and utility functions. For each chance node C we have a probability distribution $P(C \mid \text{pa}(C))$ for C given its parents $\text{pa}(C)$, and for each utility node U we have a utility function ψ_U; ψ_U maps each configuration of the parents of U to a real number. We assume that the utility functions combine additively into a joint utility function ψ.

The semantics of the links are similar to the semantics from IDs, and the traditional no-forgetting assumption is also assumed. 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. For example, in Figure 1 B is observed after deciding on BT, and S is observed before the first decision, since it has no antecedent decision variables.

The structural specification of a UID yields a partial temporal order. If a partial order is extended to a lineal order we get an influence diagram. Such an extended order is called an *admissible order*.

2.2 Solving a UID

Solving a UID means establishing a set of *step-policies* and a set of *decision-policies*. Together, the step-policies and the decision-policies form an optimal strategy. To organize the computations, we work with a secondary computational structure, called an S-DAG, which is a DAG representing the admissible orderings of the nodes in the UID (see Figure 2). A *GS-DAG* is a minimal S-DAG containing all admissible orderings relevant for computing an optimal strategy.

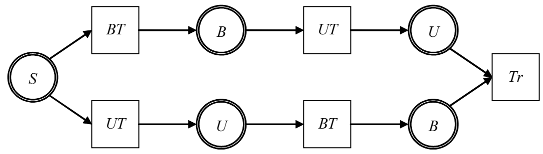

Figure 2: An S-DAG for the UID model of Figure 1.

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

To define the expected utility (EU) of a strategy S, we unfold it to a *strategy tree*: following the policies of S we construct a tree in which all root-leaf paths represent admissible orderings. The expected utility for a strategy tree is defined as for decision trees, and it is by definition the expected utility of the strategy.

Jensen and Vomlelova (Jensen and Vomlelova, 2002) describe an algorithm for finding a strategy of maximum expected utility (MEU). The algorithm utilizes the S-DAG for the UID, and basically solves the UID/S-DAG through dynamic programming similarly to solving influence diagrams (i.e., eliminating the variables in reverse temporal order).

3 An Anytime Algorithm

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.

With respect to UIDs, the standard algorithm 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. 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.

We propose an algorithm performing a forward search in a decision tree (Raiffa and Schlaifer, 1961) representation of the UID. The tree is built from the root toward the leaves, and it keeps a list of triggered nodes (the current leaves in the tree constructed so far) as candidates for expansion.[1] A triggered node X is *expanded* by adding its children to the tree and calculating the expected utility of the path from the root to X using a *heuristic function* for estimating the maximum expected utility (MEU) obtainable at the children of X.

[1]The terminology is borrowed from AO* search algorithms (Nilsson, 1980), from which the proposed algorithm has been inspired.

3.1 A Search Based Solution Algorithm

An S-DAG (and a GS-DAG) can be converted into a decision tree (possibly using a dummy source node), which in turn can be used as a computational structure for solving the corresponding decision problem (disregarding complexity issues). A decision tree is a rooted tree in which the leaves are utility nodes and the nonleaf nodes are either decision nodes or chance nodes. 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 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. These probabilities can be found by converting the UID into a Bayesian network: value nodes are removed, and decision nodes are replaced by chance nodes having no parents and with an arbitrary probability distribution.[2] Finally, with each value node V in the decision tree, we associate the utility $\psi(\text{past}(V))$ of the scenario $\text{past}(V)$. These utilities can be read directly from the UID model.

The decision tree represents each scenario in the decision problem explicitly; hence the size of the tree can grow exponentially in the

[2]The time for computing the probabilities is small compared to the time required for evaluating the UID, and we shall therefore not consider this issue further.

number of variables. The size can, however, be reduced by collapsing identical subtrees, a procedure also know as *coalescence* (Olmsted, 1983). The opportunities for exploiting coalescence can be automatically detected in the S-DAG of the UID.

Instead of building the decision tree in full and solving it using the "average-out and fold-back" algorithm (Raiffa and Schlaifer, 1961), we propose to build the tree from the root toward the leaves. A heuristic function h provides an estimate of the 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, which gives an estimate of the MEU of the nodes in the explored part of the tree, and, in particular, an estimate of the optimal policy 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 the heuristic function is called a *partial heuristic strategy* $\hat{\Delta}'$. Clearly, the closer the heuristic function is at estimating the MEU of the triggered nodes in the partial decision tree, the closer the EU of $\hat{\Delta}'$ will be at the EU of Δ'.

A partial strategy can always be extended to a full (not necessarily optimal) strategy by assigning random policies to the decision nodes in the unexplored part of the tree. When we have a set of policies S, we define the *uniform extension* of S as a strategy Δ such that every policy in S is in Δ and the rest of the policies in Δ are uniform distributions.

3.2 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. A special class of heuristic functions are the so-called admissible heuristic functions.

Definition 1. A heuristic function h is said to be *admissible* if $h(N) \geq \text{MEU}(N)$ for any node N in the decision tree.

An admissible heuristic can be exploited during the search: Consider a decision node whose children X and Y are the roots in two subtrees. If the subtree defined by Y has been explored and $h(X) \leq \text{MEU}(Y)$, then we need not explore the subtree rooted at X.

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.

3.2.1 An Admissible Heuristic

A possible choice of heuristic function could be (Vomlelova, 2003)

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

where $\mathcal{L}$ is the set of leaf nodes in the subtree 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.

It is trivial to see that h_U is admissible. Moreover, h_U has the advantage of being computationally efficient, since it can be evaluated by max-marginalizing out the variables appearing in the domains of the utility potentials. The number of required max-marginalizations is at most $|\mathbf{V}_C \cup \mathbf{V}_D|$. In contrast to the dynamic programming approach, the complexity of computing this heuristic does not depend on the number of possible paths in the GS-DAG as max-operations commute.

Unfortunately, preliminary experiments have shown that h_U yields a very loose bound on the expected utility. 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 provide 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.

3.2.2 A Nonadmissible Heuristic

The estimation given by the admissible heuristic h_U can be extremely far from the MEU. However, since it provides an upper bound on the expected utility, we can use it in combination with a lower bound to derive a good approximation to the expected utility.

As a lower bound h_L, we use the expected utility of the uniform extension of the current partial heuristic strategy; decision nodes in the unexplored part of the decision tree are treated as chance nodes with a uniform distribution. Relative to the computational complexity of solving the UID, this heuristic can be calculated efficiently by sum-marginalizing out the variables in the utility and probability potentials. The number of required marginalizations is at most $|\mathbf{V}_C \cup \mathbf{V}_D|$ and does not depend on the number of paths in the GS-DAG as sum-marginalizations commute (this means that we are not required to follow an admissible elimination order consistent with the UID).

If all the variables in the future of node X are chance variables, i.e., if $\text{future}(X) \subseteq \mathbf{V}_C$, then $h_L(X) = \text{MEU}(X)$. Furthermore, as the number of decision nodes in future(X) increases the larger the difference $\text{MEU}(X) - h_L(X)$ will be. The opposite holds for the heuristic $h_U(X)$.

In order to derive a heuristic close to the actual expected utility, we define the non-admissible heuristic h as a weighted linear combination of h_L and h_U:

$$h(X) = w_L(X)h_L(X) + w_U(X)h_U(X),$$

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

One potential difficulty with this heuristic is how to choose a good value for k_X. To alleviate this problem, we propose to update k_X automatically as the tree is expanded. The intuition underlying the updating method is that we would in general expect the heuristic to be more precise the closer we get to the leaves: After a node X has been expanded we first estimate the expected utility of its children (using h and the current value for k_X). These estimates are then propagated upward in the tree: If X is a chance node, then the value propagated to X is $\widehat{\text{EU}}(X) = \sum_{Y \in \text{ch}(X)} P(Y|\text{past}(X))h(Y)$ and if X is a decision node then the value is $\widehat{\text{EU}}(X) = \max_{Y \in \text{ch}(X)} h(Y)$. By treating $\widehat{\text{EU}}(X)$ as an accurate estimate of the expected utility for X, we calculate a new value for k_X by setting $\widehat{\text{EU}}(X) = h(X)$:

$$k_X := \frac{\widehat{\text{EU}}(X) - \alpha h_U(X)d(X)}{\alpha c(X)h_L(X)}.$$

Note that k_X will always be non-negative, and that the update is not guaranteed to get us closer to the true expected utility (we might e.g. have started off with the "correct" value for k_X).

3.2.3 Performing the Search

The search/construction of the coalesced decision tree starts with the tree consisting of a single root node together with its children (such a tree stump is always uniquely identifiable). From this tree structure 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 a node is added to the decision tree, a heuristic estimate of the MEU for that node is calculated. The values are then propagated upwards, possibly updating the current partial heuristic strategy.

The choice of which node to expand is non-deterministic. We have experimented with three selection schemes: (*i*) expand the node X with highest probability $P(\text{past}(X))$ of occurring (decision nodes are given an even probability distribution), (*ii*) expand the node X with highest weight $w(X) = P(\text{past}(X)) \cdot h(X)$, where h is the heuristic function estimating the expected utility of node X, and (*iii*) expand the node of lowest depth, i.e., perform a breadth first search. Preliminary experiments suggest that the latter provides the best results, and this is therefore the selection scheme used in the tests documented in Section 4.

4 Experiments

We have performed a series of experiments for assessing the performance of the proposed algorithm. For comparison we used dynamic programming (Jensen and Vomlelova, 2002), and to test the algorithms we generated a collection of random UIDs.

4.1 Generation of UIDs

It is easy to come up with artificial UID structures that, from a specification point of view, cannot be considered proper models of real-world decision problems. As an example, consider a UID with a decision node having only barren nodes (Shachter, 1986) in its future. Thus, rather than generating completely random UIDs (Vomlelova, 2003) we have instead tried to guide the UID generation by making perturbations of pre-specified UID templates.

Specifically, we manually constructed four UID templates from which we sampled 13 UID structures with varying number of decision nodes, chance nodes, and observable chance nodes. For each structure we randomly generated 50 realizations (probability and utility tables), producing a total of 650 models. Space restrictions prevent us from including additional details, but all models (including the templates) and a description of the sampling algorithm can be found at `www.ia.uned.es/~mluque/UID`.

4.2 Evaluation Metrics

The proposed anytime algorithm is intended for situations, where a DM is required to take one or more initial decisions but does not have time to wait for dynamic programming to finish. On the other hand, after the specified decisions have been taken we assume that there is sufficient time for dynamic programming to return an optimal strategy for the remaining decisions. Here we also assume that simply solving the UID offline and storing the policies as look-up tables is prohibitive due to space requirements.

The performance of the algorithm is evaluated according to the following two characteristics. *i*) The frequency with which the anytime algorithm returns the correct decision options (relative to the optimal strategy) for all decision nodes down to the ith level in the decision tree. *ii*) The expected utility of following the strategy prescribed by the anytime algorithm for the first i levels of decisions, followed by the optimal strategy for the remaining decisions. Both of these two measures depend on the amount of computation time used by the anytime algorithm, and to compare the results for different models, time is thus specified relative to the time required for dynamic programming to finish.

4.3 Experimental Results

The algorithms were implemented in Java 6.0 with the Elvira software package.[3] The experiments were performed on an Intel Core 2 computer (2.4 GHz) with 2 GB of memory.

First of all, it is important to emphasize

[3]The Elvira program was developed as a collaborative project of several Spanish universities (Elvira Consortium, 2002). The program and its source code can be downloaded from `www.ia.uned.es/~elvira`.

that all reported values are normalized with the uniform strategy as baseline value, i.e., the uniform strategy and the optimal strategy attains the values 0 and 1, respectively.

The results obtained by letting the anytime algorithm run for e.g. 50% of the time required by dynamic programming are listed in the second column in Table 1; $EU^i(t)$ and $AccFreqDec^i(t)$ correspond to the two measures described in Section 4.2. In particular, $AccFreqDec^1(t)$ denotes the frequency of selecting the best initial decision (i.e., a branching point decision). For example, if we assume that the initial choice is between two decisions, then the anytime algorithm returns the optimal decision with a frequency of 0.742 $(0.5 + 0.484 \cdot 0.5)$. Similarly, suppose that the expected utility of following a random policy for the first decision is 90 and the MEU is 100, then a value of 0.514 for $EU^1(t)$ corresponds to an expected utility of 95.14.

From the results we clearly see that the algorithm improves over time w.r.t. all the recorded characteristics. Additional results can be found at `www.ia.uned.es/~mluque/UID`.

	25 %	**50 %**	**75 %**
$\mathrm{EU}^1(\mathrm{t})$	0,442	0,514	0,538
$\mathrm{EU}^2(\mathrm{t})$	0,609	0,769	0,865
$\mathrm{EU}^3(\mathrm{t})$	0,546	0,703	0,794
$AccFreqDec^1(t)$	0,383	0,484	0,505
$AccFreqDec^2(t)$	0,396	0,503	0,563
$AccFreqDec^3(t)$	0,291	0,381	0,428

Table 1: Results for the anytime algorithm.

Acknowledgements

The first author was supported by the Department of Education of Madrid, the European Social Fund and the Spanish Ministry of Education and Science (grant TIN-2006-11152). We would like to thank Marta Vomlelova for giving us access to her UID implementation.

References

[Bielza et al.1999] C. Bielza, S. Ríos, and M. Gómez. 1999. Influence diagrams for neonatal jaundice management. In *AIMDM '99*, pages 138–142, London, UK. Springer-Verlag.

[Demirer and Shenoy2001] R. Demirer and P. P. Shenoy. 2001. Sequential valuation asymmetric decision problems. *Lecture Notes in Computer Science*, pages 252–265.

[Elvira Consortium2002] The Elvira Consortium. 2002. Elvira: An environment for creating and using probabilistic graphical models. In *PGM'02*, pages 1–11, Cuenca, Spain.

[Howard and Matheson1984] R. A. Howard and J. E. Matheson. 1984. Influence diagrams. In R. A. Howard and J. E. Matheson, editors, *Readings on the Principles and Applications of Decision Analysis*, pages 719–762.

[Jensen and Vomlelova2002] F. V. Jensen and M. Vomlelova. 2002. Unconstrained influence diagrams. In *UAI'02*, pages 234–241, San Francisco, CA. Morgan Kaufmann.

[Jensen et al.1994] F. Jensen, F. V. Jensen, and S. L. Dittmer. 1994. From influence diagrams to junction trees. In *UAI'94*, pages 367–373, San Francisco, CA. Morgan Kaufmann.

[Madsen and Jensen1999] A. Madsen and F. V. Jensen. 1999. Lazy evaluation of symmetric Bayesian decision problems. In *UAI'99*, pages 382–390, San Francisco, CA. Morgan Kaufmann.

[Nilsson1980] N. J. Nilsson. 1980. *Principles of Artificial Intelligence*. Tioga, Palo Alto, CA.

[Olmsted1983] S. M. Olmsted. 1983. *On Representing and Solving Decision Problems*. Ph.D. thesis, Dept. Engineering-Economic Systems, Stanford University, CA.

[Raiffa and Schlaifer1961] H. Raiffa and R. Schlaifer. 1961. *Applied Statistical Decision Theory*. MIT press, Cambridge.

[Shachter1986] R. D. Shachter. 1986. Evaluating influence diagrams. *Operations Research*, 34:871–882.

[Shenoy1992] P. P. Shenoy. 1992. Valuation based systems for Bayesian decision analysis. *Operations Research*, 40:463–484.

[Vomlelova2003] M. Vomlelova. 2003. Unconstrained influence diagrams - experiments and heuristics. In *WUPES'2003*, Hejnice, Czech Republic.