An Anytime Approximation for Optimizing Policies Under Uncertainty

Rina Dechter
Department of Computer and Information Science
University of California, Irvine
Irvine, California, USA 92717
dechter@@ics.uci.edu

Abstract

The paper presents a scheme for approximation for the task of maximizing the expected utility over a set of policies, that is, a set of possible ways of reacting to observations about an uncertain state of the world. The scheme which is based on the mini-bucket idea for approximating variable elimination algorithms, is parameterized, allowing a flexible control between efficiency and accuracy. Furthermore, since the scheme outputs a bound on its accuracy, it allows an anytime scheme that can terminate once a desired level of accuracy is achieved. The presented scheme should be viewed as a guiding framework for approximation that can be improved in a variety of ways.

1 Introduction

Influence diagram (IDs) [7] are a popular framework for decision analysis. They subsume finite horizon factored observable and partially observable, Markov decision processes (MDPs, POMDPs) used to model planning problems under uncertainty [1].

The first part of the paper provides an overview of a bucket elimination algorithm presented in [4] for computing a sequence of policies which maximize the expected utility for an influence diagram. The algorithm is similar to variable elimination algorithms proposed by Shachter and others [12; 13; 11; 15; 10; 14; 16; 6], and in particular, it is analogous to the join-tree clustering algorithm for evaluating influence diagrams [6]. The new exposition using the bucket data-structure unifies the algorithm with a variety of inference algorithms in constraint satisfaction, propositional satisfiability, dynamic programming and probabilistic inference. Such algorithms can be expressed succinctly, are relatively easy to implement and the unification highlights ideas for improved performance, for incorporating variable elimination with search, for trading space for time and for approximation, all of which are applicable within the framework of bucket-elimination [2; 3].

Indeed, in this paper we extend the principle of mini-bucket approximation [5] that is applicable to any variable elimination algorithm, to the meu task. Specifically, we will derive and analyze the mini-bucket approximation for the meu task in influence diagrams and discuss its potential.

2 Background

2.1 Belief networks

Belief networks provide a formalism for reasoning about partial beliefs under conditions of uncertainty. It is defined by a directed acyclic graph over nodes representing random variables of interest.

A *directed graph* is a pair, $G = \{V, E\}$, where $V = \{X_1, ..., X_n\}$ is a set of elements and $E = \{(X_i, X_j)|X_i, X_j \in V, i \neq j\}$ is the set of edges. If $(X_i, X_j) \in E$, we say that X_i points to X_j. For each variable X_i, the set of parent nodes of X_i, denoted pa_{X_i} or pa_i, comprises the variables pointing to X_i in G. The family of X_i, F_i, includes X_i and its parent variables. A directed graph is *acyclic* if it has no directed cycles. In an *undirected graph*, the directions of the arcs are ignored: (X_i, X_j) and (X_j, X_i) are identical. The *moral graph* of a directed graph is the undirected graph obtained by connecting the parent nodes of each variable and eliminating direction.

Let $X = \{X_1, ..., X_n\}$ be a set of random variables over multivalued domains. A *belief network* is a pair (G, P) where G is a directed acyclic graph and $P = \{P(X_i|pa_i)\}$, denote conditional probability tables (CPTs). The belief network represents a probability distribution over X having the product form $P(x_1, ..., x_n) = \Pi_{i=1}^n P(x_i|x_{pa_i})$ where an assignment $(X_1 = x_1, ..., X_n = x_n)$ is abbreviated to $x = (x_1, ..., x_n)$ and where x_S denotes the restriction of a tuple x to the subset of variables S. An evidence set e is an instantiated subset of variables. We use upper case letter for variables and nodes in a graph and lower case letters for values in variable's domains.

2.2 Influence diagrams

An *Influence diagram* extends belief networks by adding also *decision variables* and reward functional components. Formally, an influence diagram is defined by $ID = (X, D, P, R)$, where $X = \{X_1, ..., X_n\}$ is a set of chance variables on multi-valued domains (the belief network part) and $D = \{D_1, ..., D_m\}$ is a set of decision nodes (or actions). The chance variables are further divided into *observable* meaning they will be observed during execution, or *unobservable*. The discrete domains of decision variables denote its possible set of action. An action in the decision node D_i is denoted by d_i. Every chance node X_i is associated with a conditional probability table (CPT), $P_i = \{P(X_i|pa_i)\}$, $pa_i \subseteq X \cup D - \{X_i\}$. Each decision variable D_i has a parent set $pa_{D_i} \subseteq X \cup D$ denoting the variables, whose values will be known and may directly affect the decision. The reward functions $R = \{r_1, ..., r_j\}$ are defined over subsets of variables $Q = \{Q_1, ..., Q_j\}$, $Q_i \subseteq X \cup D$, called *scopes*, and the utility function is defined by $u(x) = \sum_j r_j(x_{Q_j})$.[1].

The graph of an ID contains nodes for chance variables (circled) decision variables (boxed) and for reward components (diamond). For each chance or decision node there is an arc directed from each of its parent variables towards it, and there is an arc directed from each variable in the scope of a reward component towards its reward node.

Let $D_1, ..., D_m$ be the decision variables in an influence diagram. A decision rule for a decision node D_i is a mapping $\delta_i : \Omega_{pa_{D_i}} \rightarrow \Omega_{D_i}$ where for $S \subseteq X \cup D$, Ω_S is the cross product of the individual domains of variables in S. A policy is a list of decision rules $\Delta = (\delta_1, ..., \delta_m)$ consisting of one rule for each decision variable. To *evaluate an influence diagram* is to find an *optimal policy* that maximizes the expected utility (*meu*) and to compute the optimal expected utility. Assume that x is an assignment over both chance variables and decision variables $x = (x_1, ..., x_n, d_1, ..., d_m)$, The meu task is to compute

$$E = \max_{\Delta=(\delta_1,...,\delta_m)} \sum_{x_1,...,x_n} \Pi_{x_i} P(x_i, e|x^{\Delta}_{pa_i}) u(x^{\Delta}), \quad (1)$$

where x^{Δ} denotes an assignment $x = (x_1, ..., x_n, d_1, ...d_m)$ where each d_i is determined by $\delta_i \in \Delta$ as a functions of $(x_1, ..., x_n)$.

Example 1 *Figure 1 describes the influence diagram of the oil wildcatter problem (adapted from [8]). The diagram shows that the* test decision (T) *is to be made based on no information, and the* drill (D) *decision is to be made based on the decision to* test (T) *and the* test results (R). *The test-results are dependent on* test *and* seismic-structure (S), *which depends on an unobservable variable* oil underground (O). *The decision regarding* oil-sales-policy (OSP) *is made once the* market information (MI) *and the* oil-produced (OP) *are available. There are four reward components: cost of test, $r(T)$, drill cost, $r(D, OP)$, sales cost, $r(OP, OSP)$ and oil sales, $r(OP, OSP, MI)$. The meu task is to find the three decision rules δ_T, δ_D and δ_{OSP}: $\delta_T :\rightarrow \Omega_T$, $\delta_D : \Omega_R \rightarrow \Omega_D$, $\delta_{OSP} : \Omega_{MI,OP} \rightarrow \Omega_{OSP}$ such that:*

$$E = \max_{\delta_T,\delta_D,\delta_{OSP}} \sum_{t,r,op,mi,s,o} P(r|t, s)P(op|d, o)P(mi)P(s|o)P(o)\cdot$$

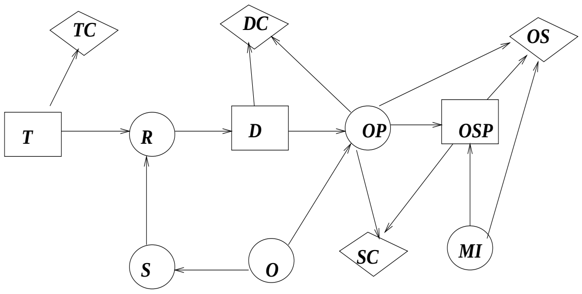

Figure 1: An influence diagram

$$[r(t) + r(d, op) + r(op, osp) + r(op, osp, mi)]$$

IDs are required to satisfy several constraints. There must be a directed path that contains all the decision nodes and there must be no forgetting in the sense that a decision node and its parents be parents to all subsequent decision nodes. The rationale behind the no-forgetting constraint is that information available now should be available later if the decision-maker does not forget. In this paper, however we do not force these requirements. For a discussion of the implications of removing these restrictions see [8].

Definition 1 (elimination functions) *Given a function h defined over subset of variables S, where $X \in S$, the functions $(\sum_X h)$ is defined over $U = S - \{X\}$ as follows. For every $U = u$, $(\sum_X h)(u) = \sum_x h(u, x)$. Given a set of functions $h_1, ..., h_j$ defined over the subsets $S_1, ..., S_j$, the product function $(\Pi_j h_j)$ and $\sum_J h_j$ are defined over $U = \cup_j S_j$. For every $U = u$, $(\Pi_j h_j)(u) = \Pi_j h_j(u_{S_j})$, and $(\sum_j h_j)(u) = \sum_j h_j(u_{S_j})$.*

3 An elimination algorithm for MEU

In [4] we presented the bucket-elimination algorithm Elim-meu-id for processing influence diagrams. For completeness sake we briefly overview the algorithm (see Figure 2).

The input to the algorithm is the set of probability components and utility components in the influence diagram. The operations of summation and maximization that define the computation (EQ. (1)) requires computation along legal orderings only. We use the ordering criteria provided in [6]: assuming a given ordering for the decision variables, unobservable variables are last in the ordering, and chance variables between D_i and D_{i+1} are those that become observable following decision D_i. Given a legal variable ordering the bucket-elimination algorithm places each probabilistic function into the bucket of its latest variable. Reward components are placed according to the special procedure in Figure 3 (for explanation see [4]). Thus, each bucket contains utility components θ_i and probability components, λ_i. The algorithm process the buckets from last to first. The procedure in a bucket computes a new probabilistic component (λ) and a new utility component (θ).

[1] The original definition of ID had only one reward node. We allow multiple rewards as discussed in [15]

The algorithm generates the λ_i of a bucket by multiplying all its probability components and summing (if the variable is a chance node) over X_i. For each $\theta_j \in bucket_i$ we compute θ_i^j as the average over X_i values normalized by the bucket's compiled λ. The computation is simplified when $\lambda = 1$, namely, when there is no evidence in the bucket, nor new probabilistic functions.

For a decision variable we compute a λ component by maximization and simplify when no probabilistic components appear in the decision bucket. We showed in[4] that processing a decision variable, does *not*, in general, allow exploiting a decomposition in the reward components.

The effect of an observation on processing either a decision bucket or a chance bucket is the assignment of the value to each component and then placing each into a lower bucket. Therefore, as usual, observed variables simplify computation by avoiding creating new dependencies among variables.

We will next demonstrate the algorithm on the wildcatter example.

Example 2 *Consider the influence diagram of Figure 1 where* $u = r(t) + r(d, op) + r(op, mi, osp)$ *the ordering of processing is* $o = T, R, D, OP, MI, OSP, S, O$. *The chance variables O and S are unobservable and therefore placed last in the ordering. The bucket's partitioning and the schematic computation of this decision problem is given in Figure 4 and is explained next.*

Initially, bucket(O) contains $P(op|d, o), P(s|o), P(o)$. *Since this is a chance bucket having no reward components we generate*

$$\lambda_O(s, op, d) = \sum_o P(op|d, o)P(s|o)P(o)$$

placed in bucket(S). The bucket of S is processed next as a chance bucket. We compute

$$\lambda_S(r, op, d, t) = \sum_s P(r|t, s)\lambda_O(s, op, d)$$

placed in bucket(OP). The bucket of the decision variable OSP is processed next. It contains all the reward components $r(op, mi, osp)$, $r(op, osp)$, $r(t)$ *and* $r(d, op)$. *Since the decision bucket contains no probabilistic component,* $r(t)$ *is moved to the bucket of D and* $r(d, op)$ *is moved to the bucket of OP. We then create a utility component*

$$\theta_{OSP}(op, mi) = \max_{osp}[r(op, mi, osp) + r(op, osp)]$$

Which is the decision rule for OSP, and place it in bucket(MI). The bucket of MI is processed next as a chance bucket, generating a constant $\lambda = 1$ *and*

$$\theta_{MI}(op) = \sum_{mi} P(mi)\theta_{OSP}(op, mi),$$

placed in the bucket of OP. Processing the bucket of OP yields:

$$\lambda_{OP}(r, t, d) = \sum_{op} \lambda_S(r, t, op, d)$$

Algorithm elim-meu-id
Input: Influence diagram (X, D, P, R), a legal ordering of the variables, o; observations e.
Output: Decision rules $\delta_1, ..., \delta_k$ that maximizes the expected utility.
1.**Initialize:** Partition components into buckets where $bucket_i$ contains all probabilistic components whose highest variable is X_i. Reward components are partitioned using procedure *partition-rewards*. In each bucket, $\lambda_1, ..., \lambda_j$, denote probabilistic components and $\theta_1, ..., \theta_l$ utility components. Let $S_1, ..., S_j$ be the scopes of probability components, and $Q_1, ..., Q_l$ be the scopes of the utility components.
2. **Backward:** For $p \leftarrow n$ downto 1, do
Process $bucket_p$:
for all functions $\lambda_1, ..., \lambda_j, \theta_1, ..., \theta_l$ in $bucket_p$, do

- **If** (observed variable) $bucket_p$ contains $X_p = x_p$, assign $X_p = x_p$ to each λ_i, θ_i, and put each resulting function in appropriate bucket.
- **else**,
  - **If** X_p **is a chance variable** compute $\lambda_p = \sum_{X_p} \prod_i \lambda_i$ and
    for each $\theta_i \in bucket_p$ compute,
    $\theta_p^i = \frac{1}{\lambda_p} \sum_{X_p} \theta_i \Pi_{i=1}^j \lambda_i$;
  - **If** X_p **is a decision variable**, then
    If bucket has only θ's , move each free from X_p θ, to its appropriate lower bucket, and for the rest compute
    $\theta_p = \max_{X_p} \sum_j \theta_j$
    else, (the general case), compute
    $\lambda_p = \max_{X_p} \prod_{i=1}^j \lambda_i \sum_{j=1}^l \theta_j$
  - Add λ_p to the bucket of the largest-index variable in their scopes. Place θ_p in the closest chance bucket of a variable in its scope or in the closest decision bucket.

3. **Return :** decision rules computed in decision buckets.

Figure 2: Algorithm *elim-meu-id*

Procedure
partition-rewards($P_1, ..., P_n, r_1, ...r_j$, o)
For i = n to 1
If X_i is a chance variable, put all rewards mentioning X_i in $bucket_i$.
else (decision variable) put all remaining rewards in current $bucket_i$
end.
Return ordered buckets

Figure 3: partition into buckets

$bucket(O): P(o|op, D)P(s|o), P(o)$
$bucke(S): P(r|s, t) \; || \; \lambda_O(s, op, d)$
$bucket(OSP)$: $r(t), r(op, osp), r(op, mi, osp), r(d, op)$
$bucket(MI)$: $P(mi) \; || \; \theta_{OSP}(op, mi)$
$bucket(OP)$: $|| \; \lambda_S(r, t, op, d), r(op, d), \theta_{MI}(op)$
$bucket(D)$: $|| \; \lambda_{OP}(r, t, d), \theta_{OP}(d, r, t)r(t)$
$bucket(R)$: $|| \; \lambda_D(r, t)$
$bucket(T)$: $|| \; \lambda_R(t)$

Figure 4: A schematic execution of elim-meu

and

$$\theta_{OP}(r, t, d) = \frac{1}{\lambda_{OP}} \sum_{op} \lambda_S(r, t, op, d)[r(op, d) + \theta_{MI}(op)]$$

both placed in the bucket of decision variable D. Here we observe the case of a decision bucket that contains both probabilistic and utility component. The bucket has two reward components (one original r(t) and one generated recently.) It computes a λ component:

$$\lambda_D(r, t) = \max_D \lambda_{OP}(r, t, d)[r(t) + \theta_{OP}(r, t, d)]$$

which provides the decision rule for D and is placed in the bucket of R. The bucket of R has only a probability component yielding: $\lambda_R(t) = \sum_r \lambda_D(r, t)$ *placed in* $bucket(T)$*, and* $\lambda_T = \max_t \lambda_R(t)$ *is derived and provides the decision rule for T. Also,* $\max_t \lambda_R(t)$ *is the optimal expected utility.*

The sequence of solution policies is given by the argmax functions that can be created in decision buckets. Once decision T is made, the value of R will be observed, and then decision D can be made based on T and the observed R.

In summary,

Theorem 1 *[4] Algorithm elim-meu-id computes the meu of an influence diagram as well as a sequence of optimizing decision rules.* □

3.1 Complexity

As is usually the case with bucket elimination algorithms, their performance can be bounded as a function of the induced width of some graph that reflect the algorithm's execution.

An *ordered graph* is a pair (G, d) where G is an undirected graph and $d = X_1, ..., X_n$ is an ordering of the nodes. The *width of a node* in an ordered graph is the number of the node's neighbors that precede it in the ordering. The *width of an ordering d*, denoted $w(d)$, is the maximum width over all nodes. The *induced width of an ordered graph*, $w^*(d)$, is the width of the induced ordered graph obtained as follows: nodes are processed from last to first; when node X is processed, all its preceding neighbors are connected. The *induced width of a graph*, $w*$, is the minimal induced width over all its orderings.

The relevant graph for elim-meu-id, is the *augmented graph* obtained from the ID graph as follows. All the parents of chance variables are connected (moralizing the graph), all the parents of reward components are connected, and all the arrows are dropped. Value nodes and their incident arcs are deleted. The reader can check that the width of the augmented graph for the oil example along the order of processing is 3 while the induced-width is 4.

The complexity of elim-meu-id is $O(n \cdot exp(w^*))$, where w^* is the induced-width of the augmented ordered graph. (see [4] for more details.)

4 Mini-bucket for MEU

Since the complexity of elim-meu-id is exponential in the induced-width of some graph and since the induced-width is at least as large as that of the underlying moral graph of the probabilistic subnetwork, frequently space complexity will not allow executing the full bucket elimination algorithm and we need an approximation scheme.

The idea of the mini-bucket approach is to localize the computation done in each bucket [5]. Since elim-meu-id process chance variables and decision variables differently we will derive the mini-bucket simplification for each of the two cases. We will demonstrate the idea using the car example.

4.1 The car buyer example

Consider a car buyer that needs to buy one of two used cars. The buyer of the car can carry out various tests with various costs, and then, depending on the test results, decide which car to buy. Figure 5 gives the influence diagram representing the car buyer example adapted from [9]. T denotes the choice of test to be performed, $T \in \{t_0, t_1, t_2\}$ (t_0 means no test, t_1 means test car 1, and t_2 means test car 2). D stands for the decision of which car to buy, $D \in \{buy\ 1,\ buy 2\}$. C_i represents the quality of car i, $i \in \{1, 2\}$ and $C_i \in \{q_1, q_2\}$ (denoting good and bad qualities. t_i represents the outcome of the test on car i, $t_i \in \{pass,\ fail, null\}$. The null option corresponds to the case when a test on car i was not performed. The cost of testing is given as a function of T, by $r(T)$, the reward in buying car 1 is defined by $r(C_1, D = buy\ 1)$ and the reward for buying car 2 is $r(C_2, D = buy\ 2)$. The rewards $r(C_2, D = buy\ 1) = 0$, and $r(C_1, D = buy\ 2) = 0$. The total reward or utility is given by $r(T) + r(C_1, D) + r(C_2, D)$.

The task is to determine T and D that maximize the expected utility. Namely,

$$E = \max_{T,D} \sum_{t_2, t_1, C_2, C_1} P(t_2|C_2, T)P(C_2)P(t_1|C_1, T) \cdot \quad (2)$$
$$P(C_1)[r(T) + r(C_2, D) + r(C_1, D)].$$

The bucket-elimination algorithm partitions the probabilistic and reward components into ordered buckets as described in the algorithm. Using the ordering $o = T, t_2, t_2, D, C_2, C_1$, the initial partitioning for the car example is given in Figure 6.

Figure 7 shows the recorded functions in the buckets after processing in reverse order of o.

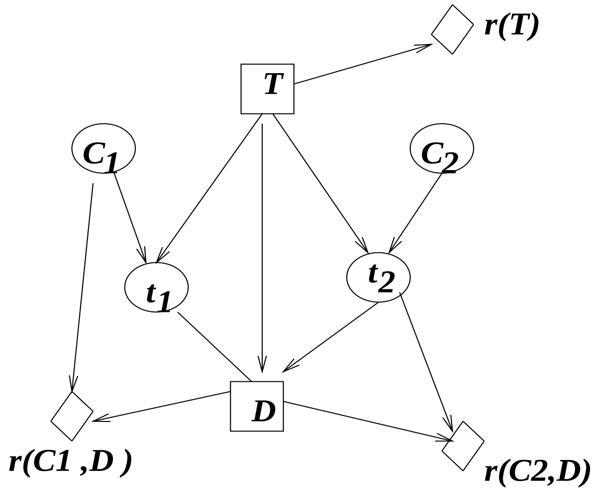

Figure 5: Influence diagram for car buyer example

$bucket(C_1)$: $P(C_1), P(t_1|C_1,T), r(C_1,D)$
$bucket(C_2)$: $P(C_2), P(t_2|C_2,T), r(C_2,D)$
$bucket(D)$:
$bucket(t_1)$:
$bucket(t_2)$:
$bucket(T)$: $r(T)$

Figure 6: Initial partitioning into buckets

The optimizing policies for δ_T is the simple decision function computed in $bucket(T)$, the argmax of EQ. (3):

$$E = \max_T \lambda_{t_1}(T)\lambda_{t_2}(T)[r(T) + \theta_{t_2}(T)]. \quad (3)$$

and $\theta_D(t_1, t_2, T)$ that is recorded in $bucket(t_1)$ or its associated maximizing decision D.

We will next apply the mini-bucket idea. From EQ. (2) we get:

$$E = \max_T \sum_{t_2}\sum_{t_1} \max_D \sum_{C_2} P(t_2|C_2,T)P(C_2)\cdot \quad (4)$$

$$\sum_{C_1} P(t_1|C_1,T)P(C_1)[r(T) + r(C_1,D) + r(C_2,D)]$$

Focusing on the expression involving C_1 (that corresponds to processing $bucket(C_1)$) leads to computing the λ and θ as in EQ. (5) and (6):

$$\lambda_{C_1}(t_1,T) = \sum_{C_1} P(t_1|C_1,T)P(C_1) \quad (5)$$

$bucket(C_1)$: $P(C_1), P(t_1|C_1,T), r(C_1,D)$
$bucket(C_2)$: $P(C_2), P(t_2|C_2,T), r(C_2,D)$
$bucket(D)$: $||\ \theta_{C_1}(t_1,T,D), \theta_{C_2}(t_2,T,D)$
$bucket(t_1)$: $||\ \lambda_{C_1}(t_1,T),\quad \theta_D(t_1,t_2,T)$
$bucket(t_2)$: $||\ \lambda_{C_2}(t_2,T),\quad \theta_{t_1}(t_2,T)$
$bucket(T)$: $r(T)||\lambda_{t_1}(T),\quad \lambda_{t_2}(T),\quad \theta_{t_2}(T)$.

Figure 7: Schematic bucket evaluation of car example

and

$$\theta_{C_1}(t_1,T,D) = \frac{1}{\lambda_{C_1}(t_1,T)}\cdot\sum_{C_1} P(t_1|C_1,T)P(C_1)r(C_1,D) \quad (6)$$

Instead, we will now derive an approximation for these expressions.

$$E_{C_1} = \sum_{C_1} P(t_1|C_1,T)P(C_1)[r(T)+r(C_1,D)+r(C_2,D)] =$$
$$= \sum_{C_1} P(t_1|C_1,T)P(C_1)[r(T) + r(C_2,D)]+$$
$$\sum_{C_1} P(t_1|C_1,T)P(C_1)r(C_1,D).$$

By applying the summation over C_1 in two stages, migrating summation inside the product we get

$$E_{C_1} \leq [r(T) + r(C_2,D)][\sum_{C_1} P(t_1|C_1,T)][\sum_{C_1} P(C_1)]+$$
$$[\sum_{C_1} P(t_1|C_1,T)][\sum_{C_1} P(C_1)r(C_1,D)] =$$
$$[\sum_{C_1} P(t_1|C_1,T)][\sum_{C_1} P(C_1)]\cdot[r(T) + r(C_2,D)+$$
$$\frac{1}{\sum_{C_1} P(C_1)}[\sum_{C_1} P(C_1)r(C_1,D)]]$$

The last expression corresponds to partitioning the bucket of C_1 into two mini-buckets; one containing $P(t_1|C_1,T)$ and the other containing $P(C_1)$ and $r(C_1,D)$. Each mini-bucket is processed as a chance bucket in elim-meu-id (see in Figure 8, the initial partitioning into mini-buckets). We assign indices to mini-buckets in a buckets in the order from left to right. Processing the first mini-bucket of C_1 yields $\lambda^1_{C_1}(t_1,T) = \sum_{C_1} P(t_1|C_1,T)$ (superscripts denote the mini-bucket creating the function), and processing the second mini-bucket yields both a λ function $\lambda^2_{C_1} = \sum_{C_1} P(C_1) = 1$, and a θ function $\theta^2_{C_1}(D) = \sum_{C_1} P(C_1)r(C_1,D)$, both placed in a corresponding bucket below. Processing the bucket of C_2 is similar.

Likewise, we can derive the mini-bucket processing of decision variable. In our example, by the time $bucket(D)$ is processed it contains two θ functions created earlier. Since the bucket has no probabilistic component, it is processed as a regular full decision bucket and since it has functions on over D only. Namely we compute $\theta_D = max_D(\theta^2_{C_1}(D)+\theta^2_{C_2}(D))$. The final status of the buckets after processing is given in Figure 8.

We see that the mini-bucket partitioning yields a decision D, that is independent of the testing of the cars: $D = argmax_D(\theta_{C_1}(D)+\theta_{C_2}(D))$ (superscripts are omitted whenever no confusion may arise). Indeed, since the functions $\lambda_i(t_i)$ for $i = 1,2$ evaluates to the constant (1), the decision $T = t_0$ is selected when evaluating bucket T (we assume negative reward functions). So the approximation yields a very simple decision rule. No testing (the decision for T) and then the appropriate maximization of expected cost for deciding which car to buy. We next present the general derivation.

$bucket(C_1)$: $\{P(t_1|C_1,T)\}, \{P(C_1), r(C_1,D)\}$
$bucket(C_2)$: $\{P(t_2|C_2,T)\}, \{P(C_2), r(C_2,D)\}$
$bucket(D)$: $||\ \theta^2_{C_1}(D), \quad \theta^2_{C_2}(D)$
$bucket(t_1)$: $||\ \lambda^1_{C_1}(t_1,T)$
$bucket(t_2)$: $||\ \lambda^1_{C_2}(t_2,T)$
$bucket(T)$: $r(T)\ ||\ \lambda_{t_1}(T), \lambda_{t_2}(T),\ \theta_D$
$T = armax_T \lambda_{t_1}(T)\lambda_{t_2}(T)[r(T)+\theta_D]$

Figure 8: Initial partitioning into buckets

4.2 General derivation for chance buckets

As specified by the full bucket-elimination algorithm, if X_p is a chance bucket, the full bucket elimination algorithm, computes in $bucket_p$ (that contains the components: $\lambda_1,...,\lambda_j,\theta_1,...,\theta_k$) two types of functions. $\lambda_p = \sum_{X_p}\prod_{i=1}^{j}\lambda_i$ and, for each $\theta_i \in bucket_p$, $\theta^i_p = \frac{1}{\lambda_p}\sum_{X_p}\theta_i(\prod_{i=1}^{j}\lambda_i)$. Since the functions λ_p and θ^i_p may have high dimensionality we will try to approximate them with functions of lower dimension. To achieve that, two parameters, i and m are used. Given this two parameters, we partition the probabilistic components and the utility components $\lambda_1,...,\lambda_j,\theta_1,...,\theta_k$ in bucket X_p into an (i,m)-partitioning, not allowing more than i variables or more than m components in a mini-bucket. Denoting the mini-buckets $Q\prime = \{Q_1,...,Q_r\}$. For each $Q_l \in Q\prime$, containing $\lambda_{l_1},...,\lambda_{l_{t_l}},\theta_{l_1},...,\theta_{l_{f_l}}$ we compute

$$\lambda^l = \sum_{X_p}\prod_{i=1}^{t_l}\lambda_{l_i},$$

and, for each $\theta_{l_j} \in Q_l$ we compute

$$\theta^{l_j} = \frac{1}{\lambda^l}\sum_{X_p}\theta_{l_j}\prod_{i=1}^{t_l}\lambda_{l_i}$$

Each of these components is moved to a lower bucket using the usual rule.

We next show that for any partitioning, this approximation yields an upper bound on the exact computation in a chance bucket.

Let's denote by t_p the indices of variables in the scopes of the θ functions which are currently placed in lower buckets, namely $i \in t_p$ if $\theta_i \in bucket_k$ for $k < p$, and by l_p the set of indices of θ_i functions in $bucket_p$ (namely, those are defined over X_p). The function computed in the bucket of X_p by full bucket elimination is equivalent to:

$$E_p = \sum_{X_p}\prod_{i=1}^{j}\lambda_i(\sum_{k\in t_p}\theta_k + \sum_{k\in l_p}\theta_k)$$

Since θ_i for $i \in t_p$ does not depend on X_p we get:

$$E_p = \sum_{k\in t_p}\theta_k \cdot \sum_{X_p}\prod_{i=1}^{j}\lambda_i + \sum_{k\in l_p}\sum_{X_p}\theta_k\prod_{i=1}^{j}\lambda_i$$

Given the partitioning into mini-buckets $Q\prime = Q_1,...,Q_r$, and assuming that θ_k appears in a specific mini-bucket $Q_{l_{j(k)}}$, we can upper-bound E_p by

$$E_p \le \sum_{k\in t_p}\theta_k \cdot \prod_{Q_j\in Q\prime}\sum_{X_p}\prod_{\lambda_{j_i}\in Q_j}\lambda_{j_i}$$

$$+\sum_{k\in l_p}\sum_{X_p}\theta_k(\prod_{\lambda\in Q_{l_{j(k)}}}\lambda)\cdot(\prod_{Q_l\in Q\prime, l\neq l_{j(k)}}\sum_{X_p}\prod_{\lambda\in Q_l}\lambda)$$

Namely, we bound $\sum_{X_p}\prod_{i=1}^{j}\lambda_i$ by $\prod_{Q_l\in Q\prime}\sum_{X_p}\prod_{\lambda\in Q_l}\lambda$ when exchanging summation with multiplication. Lets denote:

$$g^p = \prod_{Q_l\in Q\prime}\sum_{X_p}\prod_{\lambda\in Q_l}\lambda$$

We get:

$$E_p \le g^p\cdot[\sum_{k\in t_p}\theta_k + \sum_{k\in l_p}\frac{1}{\prod_{Q_l\in Q\prime}\sum_{X_p}\prod_{\lambda\in Q_l}\lambda}\cdot$$

$$\cdot[\sum_{X_p}\theta_k\prod_{\lambda\in Q_{l_{j(k)}}}\lambda]\cdot(\prod_{Q_l\in Q\prime, j\neq j_l}\sum_{X_p}\prod_{\lambda\in Q_l}\lambda)]$$

After canceling out multiplicands in the dominator and denominator in the second summand we get:

$$= g^p\cdot[\sum_{k\in t_p}\theta_k + \sum_{k\in l_p}\frac{1}{\sum_{X_p}\prod_{\lambda\in Q_{l_{j(k)}}}\lambda}\cdot\sum_{X_p}\theta_k\prod_{\lambda\in Q_{l_{j(k)}}}\lambda]$$

This concludes the justification for processing chance mini-buckets. It shows that for each mini-bucket of a chance variable we simply apply the same procedure that is associated with the processing of full chance buckets.

4.3 General derivation for decision bucket

A decision bucket may contain both probabilistic and utility components. In that case the full bucket elimination algorithm computes a λ function

$$\lambda_p = \max_{X_p}\prod_i\lambda_i(\sum_{k\in t_p\cup l_p}\theta_k)$$

Lets choose a partitioning where all utility components are placed in one mini-bucket. Let $Q\prime = \{Q_1,...,Q_r\}$ be the partitioning and let Q_1 be the the mini-bucket containing all the utility components.

$$\lambda_p = \max_{X_p}[\prod_{\lambda\in Q_1}(\lambda\sum_{k\in t_p\cup l_p}\theta_k)\prod_{j=2,...,r}\prod_{\lambda_i\in Q_j}\lambda_i]$$

In that case λ_p can be approximated, when migrating the maximization into each mini-bucket, by

$$\lambda_p \le \max_{X_p}[\prod_{\lambda_i\in Q_1}\lambda_i\sum_k\theta_k]\cdot\prod_{j=2,...,r}\max_{X_p}\prod_{\lambda_i\in Q_j}\lambda_i \qquad (7)$$

This means that we compute a λ component in each λ-pure mini-bucket by

$$\lambda^l_p = \max_{X_p}\prod_{\lambda_i\in Q_l}\lambda_i \qquad (8)$$

and in the mixed bucket by

$$\lambda_p^{Q1} = \max_{X_p}[\prod_{\lambda_i \in Q_1} \lambda_i \sum_k \theta_k] \tag{9}$$

If the mixed mini-bucket has too many variables, it is further partitioned as guided by the following expressions. Following some algebraic manipulation we get from EQ. (7):

$$\lambda_p \leq \prod_{j=1,...,r} \max_{X_p} \prod_{\lambda_i \in Q_j} \lambda_i \sum_{k \in t_p}[\theta_k + \frac{\max_{X_p} \prod_{\lambda_i \in Q_1} \lambda_i \sum_{k \in l_p} \theta_k}{\max_{X_p} \prod_{\lambda_i \in Q_1} \lambda_i}]$$

This suggests that, as before, the pure λ buckets are computed by EQ. (8). In the mixed mini-bucket Q_1, we first move all θ elements not defined over X_p to lower buckets and then compute a λ function

$$\lambda_p^{Q_1} = \max_{X_p} \Pi_{\lambda \in Q_1} \lambda$$

and a θ function by:

$$\theta_p^1 = \frac{\max_{X_p} \prod_{\lambda_i \in Q_1} \lambda_i \sum_{k \in l_p} \theta_k}{\max_{X_p} \prod_{\lambda_i \in Q_1} \lambda_i} \tag{10}$$

If $\sum_{k \in l_p} \theta_k$ is still too high dimensional and cannot fit into one mini-bucket, we can further partition Q_1 into mixed mini-buckets $\{Q_{1_1}, ...Q_{1_j}, ..., Q_{1t}\}$ and compute separate functions for each subset $Q_{1_j} \subseteq Q_1$. By exchanging summation and multiplication we get:

$$\theta_p^1 \leq \frac{\sum_j \max_{X_p}[\sum_{\theta_k \in Q_{1_j}} \theta_k \prod_{\lambda_i \in Q_1} \lambda_i]}{\max_{X_p} \prod_{\lambda_i \in Q_1} \lambda_i} \tag{11}$$

Therefore, it will allow computing for each $Q_{1_j} \subseteq Q_1$, the function $\theta_p^{1_j}$ by:

$$\theta_p^{1_j} = \frac{\max_{X_p}[\sum_{\theta_k \in Q_{1_j}} \theta_k] \prod_{\lambda_i \in Q_1} \lambda_i}{\max_{X_p} \prod_{\lambda_i \in Q_1} \lambda_i}. \tag{12}$$

Algorithm *approx-meu-id(i,m)* is described in Figure 9.

Theorem 2 *Algorithm Approx-meu-id(i,m) computes an upper bound on the maximum expected utility of a set of policies. Its complexity is time and space exponential in the bounding parameters i and m.* □

Example 3 *Consider again the wildcatter example. The initial partitioning into buckets is as in the full bucket elimination case. Here we show the final buckets assuming i=3.*

The processing of each bucket is explained as follows. Bucket(O) has two mini-buckets. Processing the first yields a λ component: $\lambda_O^1(op, d) = \sum_O P(o|op, d)$ which is placed in bucket(OP). Processing the second mini-bucket yields a λ component: $\lambda_O^2(S) = \sum_O P(s|o)p(o)$ placed in bucket S. Processing bucket(S) is done as a full bucket since its number of variables is bounded by 3. We get: $\lambda_S(r, t) = \sum_S P(r|s, t)\lambda_O^2(s)$, placed in bucket(r). From now on, all buckets will be processed as full buckets. The bucket of the decision variable OSP is processed next. Since the decision bucket contains no probabilistic

Algorithm approx-meu-id(i,m)
Input: Influence diagram (X, D, P, R), a legal ordering , o; observations e.
Output: Decision rules $\delta_1, ..., \delta_k$ approximating the max expected utility.
1.**Initialize:** Partition into buckets as in elim-meu-id. Call probability matrices $\lambda_1, ..., \lambda_j$ and utility matrices $\theta_1, ..., \theta_l$. Let $S_1, ..., S_j$ be the scopes of the probability components and $Q_1, ..., Q_l$ be the scopes of the reward components.
2. **Backward:** For $p \leftarrow n$ downto 1, do

- **If** (observed variable, or a predetermined decision policy) $bucket_p$ contains $X_p = x_p$, then assign $X_p = x_p$ to each λ_i, θ_i, and place result in lower bucket.
- **else, If** X_p **is a chance variable** then generate an (i, m)-partitioning of $bucket_p$, $Q\prime = Q_1, ..., Q_t$.
  For each mini-bucket Q_l, containing $\lambda_{l_1}, ..., \lambda_{l_j}, \theta_{l_1}, ..., \theta_{l_l}$ compute:
  - $\lambda^l = \sum_{X_p} \prod_{i=1}^t \lambda_{l_i}$
  - For each $\theta_k \in Q_l$ compute $\theta^{l_k} = \frac{1}{\lambda_l} \sum_{X_p} \theta_k \prod_{\lambda \in Q_l} \lambda$

  else, If X_i **is a decision variable** then Place all utility components in Q_1 together with λ components whose scopes are included in that bucket. Generate an (i,m) partitioning to the rest of the λs, yielding mini-buckets: $Q_1, ..., Q_t$. Call probability matrices in Q_l $\lambda_{l_1}, ..., \lambda_{l_j}$
  - For every pure mini-bucket Q_l compute $\lambda^l = \sum_{X_p} \prod_{i=1}^t \lambda_{l_i}$
  - **If** the mixed mini-bucket Q_1 fits into one mini-bucket of an (i,m)-partitioning compute a λ component using the full bucket decision rule:
    $\lambda_p^{Q1} = \max_{X_p} \Pi_{\lambda \in Q_1} \lambda \sum_{\theta \in Q_1} \theta$
    else, move all θ functions not defined on X_p to lower buckets, then apply an arbitrary (i,m)-partitioning for the rest of the functions called $Q_{1_1}, ..., Q_{1_n}$.
    For any mixed $Q_{1_j} \subseteq Q_1$, compute:

$$\theta_p^{1_j} = \frac{\max_{X_p} \prod_{\lambda_i \in Q_1} \lambda_i \sum_{\theta_k \in Q_{1_j}} \theta_k}{\max_{X_p} \prod_{\lambda_i \in Q_1} \lambda_i} \tag{13}$$

  Add θ and λ to the bucket of the largest-index in their argument list.

3. **Forward:** Return an upper bound on the maximum expected utility computed in the first bucket. Execute the policies recorded in the buckets of the decision parents using the ordering $X_1, ...X_n$.

Figure 9: Algorithm approx-elim-meu-id

$bucket(O) : P(o|op,D)P(s|o), P(o)$
$bucke(S) : P(r|s,t) \;||\; \lambda_O(s,op,d)$
$bucket(OSP)$: $r(t), r(op,osp), r(op,mi,osp), r(d,op)$
$bucket(MI)$: $P(mi) \;||\; \theta_{OSP}(op,mi)$
$bucket(OP)$: $||\; (\lambda_S(r,t,op,d), r(op,d), \theta_{MI}(op)$
$bucket(D)$: $||\; \lambda_{OP}(r,t,d), \theta_{OP}(d,r,t)r(t)$
$bucket(R)$: $||\; \lambda_D(r,t)$
$bucket(T)$: $||\; \lambda_R(t)$

The mini-bucket processing results in:
$bucket(O) : \{P(o|op,d)\}\{P(s|o), P(o)\}$
$bucke(S) : P(r|s,t) \;||\; \lambda^2_O(s)$
$bucket(OSP)$: $r(t), r(op,osp), r(op,mi,osp), r(d,op)$
$bucket(MI)$: $P(mi) \;||\; \theta_{OSP}(op,mi))$
$bucket(OP)$: $||\; \lambda^1_O(op,D),\ r(op,d),\ \theta_{MI}(op)$
$bucket(D)$: $||\; \lambda_{OP}(d), \theta_{OP}(d), r(t)$
$bucket(R)$: $||\; \lambda_S(r,t)$
$bucket(T)$: $||\; \lambda_{M_i}, \lambda_D(t), \lambda_R(t)$

Figure 10: Comparing the complete algorithm and its approximation on the oil example

component, $r(t)$ is moved to the bucket of D and $r(d,op)$ is moved to the bucket of OP. We then create a utility component

$$\theta_{OSP}(op,mi) = \max_{osp}[r(op,mi,osp) + r(op,osp)]$$

and place it in bucket(MI), and so on. Processing the bucket T is by: $\theta_T = \max_t[\lambda_R(t) + \lambda_{Mi} + \lambda_D(t)]$ providing the decision rule for T. Also, θ_T is an upper bound on the optimal expected utility (see Figure 10). The new recorded functions are shown to the right of the bar. The subscript of each function indicate the bucket originating the function. We see that the approximation generates only functions on two variables while the exact algorithm created functions on four variables ($\lambda_S(r,t,op,d)$). We observe that the decision policy for OSP is not changed by this mini-bucket execution, however decision D is recorded as a function of T only while for the full algorithm it is a function of both T and R.

5 Discussion and Conclusion

The paper presented a bucket-elimination algorithm for maximizing the expected utility over a set of policies and analyze its complexity using graph parameters such as induced width. We present a general approximation scheme based on the mini-bucket idea for approximating the bucket-elimination algorithm for finding optimal policies. The scheme allow a flexible control of accuracy and efficiency using a parameter i that bounds the size of the functions that can be recorded by the algorithm. The approximation generates an upper and lower bounds thus allowing an aposteriori knowledge of its quality. The approach is applicable also to the more planning-specific models of finite horizon MDPs and POMDPs to be further studied.

References

[1] C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverag. *Artificial Intelligence Researc (JAIR)*, 11:1–94, 1999.

[2] R. Dechter. Bucket elimination: A unifying framework for probabilistic inference algorithms. In *Uncertainty in Artificial Intelligence (UAI'96)*, pages 211–219, 1996.

[3] R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, pages 41–85, 1999.

[4] R. Dechter. A new perspective on algorithms for optimizing policies under uncertainty. In *AI-Planning, 2000 (submitted)*, 2000.

[5] R. Dechter and I. Rish. A scheme for approximating probabilistic inference. In *Proceedings of Uncertainty in Artificial Intelligence (UAI'97)*, pages 132–141, 1997.

[6] F.V. Jensen F. Jensen and S.L. Dittmer. From influence diagrams to junction trees. In *Tenth Conference on Uncertainty in Artificial Intelligence*, pages 367–363, 1994.

[7] R. A. Howard and J. E. Matheson. *Influence diagrams.* 1984.

[8] R. Qi N. L. Zhang and D. Poole. A computational theory of decision networks. *International Journal of Approximate Reasoning*, pages 83–158, 1994.

[9] J. Pearl. *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufmann, 1988.

[10] R. Shachter and M. Peot. Decision making using probabilistic inference methods. In *Proceedings of Uncertainty in Artificial Intelligence (UAI92)*, pages 276–283, 1992.

[11] R. D. Shachter. An ordered examination of influence diagrams. *Networks*, 20:535–563, 1990.

[12] R.D. Shachter. Evaluating influence diagrams. *Operations Research*, 34, 1986.

[13] R.D. Shachter. Probabilistic inference and influence diagrams. *Operations Research*, 36, 1988.

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

[15] J.A. Tatman and R.D. Shachter. Dynamic programming and influence diagrams. *IEEE Transactions on Systems, Man, and Cybernetics*, pages 365–379, 1990.

[16] N. L. Zhang. Probabilistic inference in influence diagrams. *Computational Intelligence*, pages 475–497, 1998.