Variable Independence in Markov Decision Problems

Irina Rish and **Rina Dechter**
Information and Computer Science
University of California, Irvine

Abstract

In decision-theoretic planning, the problem of planning under uncertainty is formulated as a multidimensional, or *factored* MDP. Traditional dynamic programming techniques are inefficient for solving factored MDPs whose state and action spaces are exponential in the number of the state and action variables, correspondingly. We focus on exploiting problems' structure imposed by *variable independence* that implies decomposability of transitional probabilities, rewards, and policies, and is captured by the *interaction graph* of an MDP, obtained from its *influence diagram*. Using the framework of *bucket elimination*[9], we formulate a variable elimination algorithm *elim-meu-id* for computing maximum expected utility, given an influence diagram, and apply it to MDPs. Traditional dynamic programming techniques for solving finite- and infinite-horizon MDPs, such as *backward induction*, *value iteration*, and *policy iteration*, can be also viewed as bucket elimination algorithms applied to a particular ordering of the state and decision variables. The time and space complexity of elimination algorithms is $O(exp(w_o^*))$, where w_o^* the *induced width* of the interaction graph along the ordering o of its nodes. Unifying framework of bucket elimination makes complexity analysis and variable ordering heuristics developed in constraint-based and probabilistic reasoning applicable to decision-theoretic planning. As we show, selecting "good" orderings improves the efficiency of traditional MDP algorithms.

Introduction

Decision-theoretic planning is concerned with problems of planning under uncertainty, that can be formulated in the well-developed framework of the Markov Decision Processes. However, a straightforward application of MDP theory to planning problems in artificial intelligence is computationally inefficient since the size of the state space is frequently immense. World's states and actions are usually represented by vectors of the state and action variables, thus yielding multidimensional, or *factored*, MDPs. The time and space complexity of the traditional dynamic programming techniques for solving MDPs is polynomial in size of the state and action spaces, and therefore exponential in the number of the state and action variables. Recent work in decision-theoretic planning has been focused on coping with this "curse of dimensionality" [3]. It was observed by Dean [6] and others that in order to overcome the exponential complexity, one has to exploit the structure in the state and action space. In particular, *variable* and *value independence* [5] should be taken in account.

In this paper, we focus on exploiting problems' structure imposed by *variable independence* that implies decomposability of transitional probabilities, rewards, and policies, and is captured by the *interaction graph* of an MDP, obtained from its *two-stage influence diagram*[7]. The nodes of an interaction graph correspond to the state and action variables, and the arcs encode probabilistic dependencies and utility (reward) function. We assume that probability, reward and policy functions are decomposable into components defined on cliques of the interaction graph. We show that this kind of problem's structure is not exploited by traditional dynamic programming techniques for solving MDPs, such as *backward induction* for finite-horizon problems, *value iteration policy iteration* for infinite-horizon case.

Using the framework of *bucket elimination*[9], we formulate a variable elimination algorithm *elim-meu-id* for computing maximum expected utility, given an influence diagram, and apply it to MDPs. Traditional dynamic programming techniques for solving finite- and infinite-horizon MDPs, such as *backward induction*, *value iteration*, and *policy iteration*, can be also viewed as bucket elimination algorithms applied to a particular ordering of the state and decision variables. Given an ordering of variables, an elimination algorithm processes each variable in this ordering and transforms the problem into an equivalent one by eliminating the variable. Such algorithms allow flexible utilization of variable ordering which can have a significant effect on their performance characteristics. While the traditional dynamic programming approach for MDPs almost exclusively followed the temporal ordering of the problem, a general elimination algorithm such as elim-meu-id can improve performance by us-

ing a more flexible ordering. The algorithm derives a good ordering from a graph that captures various dependencies in the problem in the same way it has been done for constraint networks and belief networks, and its complexity is bounded exponentially by a graph parameter called *induced width*, or w^*.

In the next section, we give necessary definitions. Section 3 defines a factored MDP and its interaction graph. An example of factored MDP is presented in the section 4. Motivation for flexible orderings is explained in the Section 5; the algorithm elim-meu-id and its properties are presented in Section 6. The next section summarizes the paper and mentions the future work.

Definitions and Preliminaries

Given an undirected graph G, and an ordering o of its nodes, the number of nodes connected to node Q that precede it in the ordering is called the *width* of Q relative to o. The width w_o of an ordering o is the maximum width of nodes along the ordering, and the width w of a graph is the minimum width over all its orderings. The graph generated by recursively connecting the parents of G, in the reverse order of o is called the *induced graph* of G, and its width, $w^*(o)$, is called the *induced width* of G [11, 10].

It is known that the induced width of a graph embedded in a k-tree is bounded by k [1]. A k-tree is defined recursively as follows. A clique of size k (complete graph with k vertices) is a k-tree. Given a k-tree defined on $Q_1, ..., Q_{i-1}$, a k-tree on $Q_1, ..., Q_i$ can be generated by selecting a clique of size k and connecting Q_i to every node in that clique.

A *belief network (BN)* [13] is a directed acyclic graph, where each node corresponds to a random variable, and arcs denote probabilistic (conditional) dependence between nodes. Given a directed arc (x, y), x is called *parent* of y, and y is called *child* of x. Each node x in the network is associated with the probability function $P(x|pa(x))$, where $pa(x)$ is the parent set of x. The network defines a joint distribution of n variables given by $P(x_1, ..., x_n) = \prod_{i=1}^{n} P(x_i|pa(x_i))$. A *moral* graph is obtained from a belief network by connecting all nodes having a common child, and ignoring the directionality of edges.

Influence diagrams extend belief networks by introducing *decision* nodes that correspond to decisions made by an agent, and *value* nodes that represent utilities, or values. The nodes of belief networks are called *chance* nodes. ID is a directed acyclic graph, where the incoming arcs to chance and value nodes represent functional dependencies (probabilistic for chance nodes, and deterministic for value nodes), while the incoming arcs to decision nodes reflect *informational* dependencies: in order to make a decision d an agent should know the values of its parents, $pa(d)$.

Factored Markov Decision Problems

Decision-theoretic planning tasks are formulated in the framework of Markov Decision Processes, that involve the set of possible world's *states*, the set of *actions*, available to an agent, *transitional probabilities* of moving from a state to state after an action has been taken, and *rewards* associated with an action in a particular state. Actions are performed at *decision epochs*, or *time stages*. Formally, a multidimensional, or *factored*, MDP is defined by:

- A set of decision epochs, $T = \{1, ..., N\}$; N is finite for *finite horizon* MDPs, and $N = \infty$ for *infinite horizon* MDPs;
- a set of n state variables $X = \{X_1, ..., X_n\}$; each variable X_i has domain D_i. The state space is defined then as $\Omega_X = \times_{i=1}^{n} D_i$; $x^t = (x_1^t, ..., x_n^t)$ denotes the state at time t.
- A set of m action variables, $A = \{A_1, ..., A_m\}$; each variable A_i has domain D_{A_j}. The action space is defined as $\Omega_A = \times_{j=1}^{m} D_{A_j}$; $a^t = (a_1^t, ..., a_m^t)$ denotes the action at time t.
- A set of probability functions $P(x_i^t|pa(x_i^t))$, where $pa(x_i^t) \subseteq \{a_1^{t-1}, ..., a_m^{t-1}\} \cup \{x_1^{t-1}, ..., x_n^{t-1}\} \cup \{x_1^t, ..., x_n^t\}$.
- Initial rewards $r(x, a)$ for taking an action a in the state x.
- terminal reward $r^N(x)$ for getting to the state x at time N (only for finite-horizon MDPs) .

Finite-horizon Markov Decision Problem [14] is to find a *policy* $\pi = (d^1, ..., d^{N-1})$, which is a sequence of *decision rules* $d^t : \Omega_X \rightarrow \Omega_A$, such that a certain optimality criterion is maximized. In the infinite horizon case, we are usually looking for a *stationary* (time-independent) policy $\pi\Omega_X \rightarrow \Omega_A$. Widely used optimality criterion is the *expected total discounted reward* [14] with *discount factor* λ, $0 \leq \lambda < 1$. If $\lambda = 1$, the criterion is called the *expected total reward*. For N-stage finite-horizon MDPs , it is defined as

$$V_{\lambda,N}^{\pi}(x) \equiv E_x^{\pi}\left(\sum_{t=1}^{N-1} \lambda^{t-1} r(x^t, a^t) + \lambda^{N-1} r(x^N)\right),$$

where $E_s^{\pi}(\cdot)$ is the expected value with respect to policy π conditioned on $x^1 = x$; $x^t and a^t$ denote, respectively, the state and the action at time t. For an infinite-horizon MDP, the expected total discounted reward of policy π is defined as

$$V_{\lambda}^{\pi}(x) \equiv \lim_{N \rightarrow \infty} E_x^{\pi} \sum_{t=1}^{N} \lambda^{t-1} r(x^t, a^t).$$

Both expected total discounted reward and expected total reward can be used for finite-horizon problems, while in the infinite horizon case only the first one is always well-defined and guarantees the convergence of traditional MDP algorithms, such as value iteration and policy iteration.

We intend to exploit the MDP structure based on *decomposability* of transitional probabilities, rewards,

and policies. Transitional probability function is *multiplicative decomposable*:

$$P(x^t|x^{t-1}, a^t) = \prod_{i=1}^{n} P(x_i^t|pa(x_i^t)).$$

The reward function is assumed to be *additively decomposable* on subsets of the state and action variables, $X_i' \subseteq X$, $A_i' \subseteq A$, $i = 1, ..., k$, as follows:

$$r(X) = \sum_{i=1}^{k} r_i(X_i', A_i').$$

Finally, we consider *decomposable decision rules* (and, as a result, *decomposable policies*) $d(X) = (d_1(Y_1), ..., d_m(Y_m))$, $Y_i \subseteq X$, that generalize the concept of decision rules (policies) in multidimensional space. This kind of decomposability of functions can be captured by an *interaction graph* of a problem [4], where nodes correspond to the variables, and the arcs connect the variables participating in same function components (*interacting* variables).

We represent an N-stage MDP (in general, non-stationary) by an *N-stage interaction graph (NG)* which is obtained as follows: 1. construct an influence diagram defined on the state and action variables at $t = 1, ..., N$, $\{X_i^t | i = 1, ..., n\}$, $\{A_j^t | j = 1, ..., m\}$, without specifying value nodes; instead, connect the variables participating in the same reward components X_i' at each time slice; informational arcs denote the dependencies between the state variables and decision rule components, i.e. there is a directed arc from X_i^t to A_j^t iff $X_i \in Y_j$, where Y_j is the argument set of j-th component of decision rule; 2. moralize the graph (for each node having incoming directional arcs, connect the node's parents and make those arcs undirected). The arcs have now a uniform meaning: two variables are connected by an arc iff they participate in same component of a probability, reward, or decision rule function. A stationary MDP, finite or infinite horizon, can be represented by a *two-stage interaction graph (2G)*.

Unit commitment: a factored MDP

The unit commitment problem is a scheduling problem from the power industry. A power plant has n power generating units, that need to be scheduled over a time period of N hours, so that certain constraint are satisfied and total cost of running the units is minimized. The *load demand* constraint requires that the power load demand L^t at each hour t does not exceed the power supply of committed units. *Minimum running time (min-up-time)* and *minimum shut-down time (min-down-time)* restrictions imply that a unit cannot be turned off (on) unless it stayed on (off) for certain minimal amount of hours. The cost includes two components, the *transitional cost* of starting up units, and the *production cost* of running a unit on a certain level of power generation. The objective is to find a most economic feasible schedule of units' start and stop times.

Each unit i can be described by two variables, the state of the unit, $X_i \in \{ON, OFF\}$, and duration of staying in the current state (in hours), D_i. The actions applicable to each unit are $A_i = \{TURN-ON, TURN-OFF, DO-NOTHING\}$. We assume that units are mutually independent, and their failures (repairs) are described by a Poisson process with expected failure rate λ (repair rate μ). $TURN-ON$ and $TURN-OFF$ actions are deterministic (failure of a unit in the moment of turning it on is assumed to have probability zero). Load demand is modeled by Gaussian distribution with known mean and standard deviation. Those assumptions allow to define transitional probabilities for each action. Minimum up- and down-time constraints rule out impossible transitions by setting corresponding transitional probabilities to 0, e.g.: $P(X_i^{t+1} = ON | D_i^t, X_i^t = OFF, A_i^t = TURN-ON) = 0$ if $D_i^t < min_down_time$. The reward function

$$r(s, a) = -(Op_Cost(s) + Tr_Cost(s, a)) + LD(s)$$

has three components: $Op_Cost(s)$,the cost of operating units in a state s (total fuel cost); $Tr_Cost(s, a)$, the cost of transition from the state s^t at hour t to another state after taking an action a; and the component $LD(s)$, representing the load demand constraint. If we make an assumption that we can import (buy) or export (sell) electric power; then, if $s = (x_1, d_1, ..., x_n, d_n, L)$,

$$LD(s) \equiv LD(x, L) = c\{\sum_{i=1}^{n} P_i \cdot x_i - L\},$$

where c is the cost per electric power unit (MW), and P_i is the power generation level of the i-th unit. The interaction graph of a unit commitment problem is given in Figure 1.

The traditional formulation of the unit commitment task is to find a schedule, or *plan*, $a^1, ..., a^{N-1}$, maximizing total reward which is the sum of one-step rewards $r(x^t, a^t)$, and adopts deterministic model, i.e. neglects unit failures and assumes that load demand forecast is known. We formulate the problem as a finite-horizon MDP, but can restrict ourselves to policies that have state-independent decision rules, i.e. can be represented as a sequence of actions $(a^1, ..., a^{N-1})$, or schedule, maximizing expected total reward. Another task would be to find an optimal policy in its conventional meaning, as a function on states, but this task is obviously much harder then the previous one.

Structure-Preserving Operators and Variable Orderings

As we mentioned above, we focus on problems having decomposable transitional probability and reward function. We investigate how this structure can be exploited by traditional dynamic programming algorithms for solving MDPs. Those techniques compute

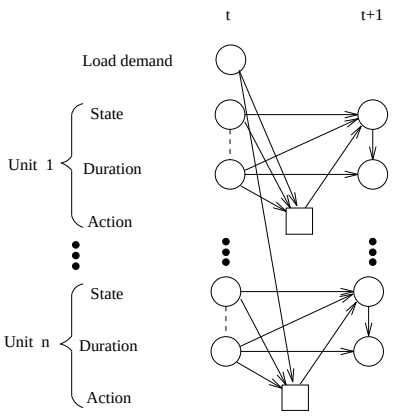

Figure 1: Interaction graph of a unit commitment problem

the total reward by sequential application of certain functional operators to a *value function* defined on the state spaces [14]. *Maximum return operator* L is defined as

$$Lv \equiv \max_{d \in D}\{r^d + \lambda P^d v\} = \max_{d \in D} L_d v,$$

where $v \in V$, V the space of bounded real-valued functions on the state space, $d : \Omega_X \rightarrow \Omega_A$ is a decision rule (a stationary policy), P^d is transitional probability matrix with components $P(x'|x, d(x))$, and r^d is the vector with components $r(x, d(x))$, and L_d is the *one period return operator* for a decision rule d.

Dynamic programming algorithm for solving N-stage finite-horizon MDPs, called *backward induction*[14], computes the expected total reward as $V_N^\pi = L^{N-1} r^N$, where $L^1 v \equiv Lv, L^N v \equiv L L^{N-1} v$. Value iteration generalizes backward induction for the infinite horizon, and computes $L^N v$ until a certain *stopping criterion* is not satisfied (see [14]). Policy iteration employs a different approach: it starts with an initial policy (decision rule) d and computes its value at the step called *policy evaluation* by sequential applying L_d operator until the sequence converges to some v(in fact, *modified policy iteration*[14] provides better stopping criteria). Then, during *policy improvement*, one step of a greedy local search step in the space of all policies d, with objective function $L_d v$, is performed; this involves computing Lv. It is proven that policy iteration is guaranteed to converge to the global maximum, i.e. to find an optimal policy.

It is easy to show that the operator Lv does not preserve the decomposability of rewards and transitional probabilities, i.e. given v and P decomposable on same subsets of variables, we cannot give a decomposable representation of Lv in general case. L combines taking expectation over the state variables with computing maximum over decision rules, that "ties" the components of the value function together. On the other hand, L_d preserves this structure since it does not involve maximization over decision rules.

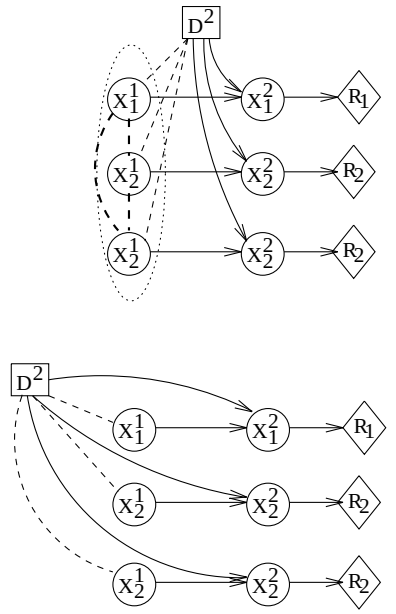

Figure 2: Graph-based ordering heuristics

The idea behind our approach presented in the next section is to allow flexible ordering of maximization and expectation operations. The ordering should be selected in accordance with the particular structure of a problem depicted in its interaction graph. Figure 2 shows an interaction graph and two possible orderings of its nodes, (X^1, D^1, X^2), and (D^1, X^1, X^2), where X^t denotes an arbitrary sequence of x_i^t, and D^t denotes action variable at time t. Note that the induced width of the first ordering is 3, while for the second ordering it is 2. An ordering heuristic, called *min-width*[8], would prefer the second ordering which corresponds to shifting maximization operation in front of L_d operators in our example. For similar example with N decision epochs, the complexity of dynamic programming on "traditional" ordering $(X^1, D^1, ..., X^{N-1}, D^{N-1}, X^N)$ (i.e. backward induction) is $O(N|D_i|^{2n}|D_{A_j}|)$, while its complexity on $(D^1, ..., D^{N-1}, X^1, ..., X^N)$ is $O(N|D_i|^2|D_{A_j}|^N)$ The computation becomes exponential in the number of time slices, but is not exponential in the state space.

Bucket Elimination for Influence Diagrams

A bucket elimination algorithm *elim-meu* for computing maximum expected utility, given a belief network containing decision variables, was presented in [9]. It implies that all decision variables are the root nodes in the belief network. We generalize elim-meu by allowing an arbitrary network structure.

Let $X = \{x_1, ..., x_n\}$ denote the set of chance nodes of an influence diagram , $D = \{d_1, ..., d_m\}$ be the set of decision (action) nodes. An additively decomposable utility (reward) function is defined as $r(X, D) =$

$\sum_{j=1}^{k} r_j(Y_j)$, where $Y \subseteq X \cup D$. Let G denote the interaction graph of the influence diagram obtained as described above; vector x (vector d) denotes an assignment to the chance (decision) nodes; x_Y (d_Z) is the restriction of the assignment x (the assignment d) on $Y \subseteq X$ (on $Z \subseteq D$). Given G, and evidence e (value assignment to variables in $X_e \subseteq X$), algorithm *elim-meu-id* (Figure 3) computes maximum expected total reward

$$V_e = max_d \sum_x \prod_{i=1}^{n} P(x_i|pa(x_i), e) \sum_{j=1}^{k} r_j(x_{Y_j})$$

and an assignment to decision nodes in form of *decomposable policy function* $d(X) = (d_1(X'_1), ..., d_m(X'_m))$, $X'_i \subseteq X$, that maximizes V_e.

The basic idea of the bucket elimination algorithms is to partition the set of functions defined on the nodes $Q_1, ..., Q_l$ of the input graph into subsets, or *buckets*, in accordance with a given ordering $o = Q_1, ..., Q_l$, so that all functions having Q_i as their highest ordered argument are placed into the i-th bucket. The buckets are processed then from the last to the first; processing a bucket of a chance node means computing an expectation on the corresponding variable, while processing a bucket of an action node means maximization over all possible values of the corresponding variable. Below we show how the expression for V_e is modified when a decision or chance node is eliminated. Let $F_i = \{x_i\} \cup pa(x_i)$, $F = \cup_{i=1}^{n}\{F_i\}$, $\lambda_i(F_i) = P(x_i|pa(x_i), e)$, $\mu_j(Y_j) = r_j(Y_j)$, then V_e can be expressed as follows:

$$\max_{d_1,...,d_m} \sum_{x_1,...x_n} \prod_{F_i} \lambda_i(F_i) \sum_{Y_j} \mu_j(Y_j).$$

Let us denote by $F_{IN}^{x_n}$ ($Y_{IN}^{d_m}$) the set of all subsets F_i (Y_j) that include x_n (d_m), and by $F_{OUT}^{x_n}$ ($Y_{OUT}^{d_m}$) the set of all subsets F_i (Y_j) that do not include x_n (d_m).

Elimination of x_n:

$$\max_{d_1,...,d_m} \sum_{x_1,...x_n} \prod_{F_i \in F} \lambda_i(F_i) \sum_{Y_j} \mu_j(Y_j) = \max_{d_1,...,d_m}$$

$$\sum_{x_1,...x_{n-1}} \sum_{x_n} \prod_{F_i \in F_{OUT}^{x_n}} \lambda_i(F_i) \sum_{x_n} \prod_{F_i \in F_{IN}^{x_n}} \lambda_i(F_i)$$

$$\{ \sum_{Y_j \in Y_{OUT}^{x_n}} \mu_j(Y_j) + \sum_{Y_j \in Y_{IN}^{x_n}} \mu_j(Y_j)\} = \max_{d_1,...,d_m}$$

$$\sum_{x_1,...x_{n-1}} \prod_{F_i \in F_{OUT}^{x_n}} \lambda_i(F_i)\lambda'(U_{x_n})\{ \sum_{Y_j \in Y_{OUT}^{x_n}} \mu_j(Y_j) + \frac{\theta(W_{x_n})}{\lambda'(U_n)}\},$$

where $U_{x_n} = \cup_i\{F_i|F_i \in F_{IN}^{x_n}\} - \{x_n\}$, $W_n = \cup_i\{F_i|F_i \in F_{IN}^{x_n}\} \bigcup \cup_j\{Y_j|Y_j \in Y_{IN}^{x_n}\} - \{x_n\}$, $\lambda'(U_{x_n}) = \sum_{x_n} \prod_{F_i \in F_{IN}^{x_n}} \lambda_i(F_i)$, $\theta(W_{x_n}) = \sum_{x_n} \prod_{F_i \in F_{IN}^{x_n}} \lambda_i(F_i) \sum_{Y_j \in Y_{IN}^{x_n}} \mu_j(Y_j)$. Functions $\lambda'(U_{x_n})$ and $\mu'(W_{x_n}) = \frac{\theta(W_{x_n})}{\lambda'(U_{x_n})}$ are computed when processing the bucket of x_n, and added to the buckets of highest ordered variables in U_{x_n} and W_{x_n}, correspondingly.

Algorithm elim-meu-id
Input: G, an ordering o, evidence e.
Output: An assignment $d^1, ..., d^m$ to decision variables that maximizes the expected reward V_e.
1. Initialize: generate an ordered partition of the conditional probability matrices, and the reward components, $bucket_1$, ..., $bucket_q$, where $bucket_p$ contains matrices $\lambda(F_i)$ and reward components $\mu(Y_j)$, whose highest variable has number p. Put the evidence assignments x^e into the corresponding buckets. A bucket of a chance (action) node is called *chance* (*decision*) bucket.
Backward steps:
2.**for** $k \leftarrow q$ **downto** 1 **do**

- **If** $bucket_k$ contains evidence x_j^e, **then** substitute x_j by x_j^e in all $\lambda(F_i)$, $\mu(Y_j)$ and put the resulting matrices in appropriate buckets.
- **else, if** k is chance bucket, compute $\lambda'(U_{x_k})$ and $\mu'(W_{x_k})$;
- **else** compute $\gamma(W_{d_k})$; Put the computed functions into the buckets of highest ordered variables in U_{x_n} and W_{x_k}, or (W_{d_k}).

3. Return an optimal set of functions $(d_1, ...d_m)$ recorded in the decision buckets, and the maximum expected reward V_e.

Figure 3: Algorithm elim-meu-id

Elimination of d_m:

$$\max_{d_1,...,d_m} \sum_{x_1,...x_n} \prod_{F_i \in F} \lambda_i(F_i) \sum_{Y_j} \mu_j(Y_j) = \max_{d_1,...,d_{m-1}}$$

$$\sum_{x_1,...x_n} \prod_{F_i \in F_{OUT}^{d_m}} \lambda_i(F_i) \max_{d_m} \prod_{F_i \in F_{IN}^{d_m}} \lambda_i(F_i)$$

$$\{ \sum_{Y_j \in Y_{OUT}^{d_m}} \mu_j(Y_j) + \sum_{Y_j \in Y_{IN}^{d_m}} \mu_j(Y_j)\} =$$

$$= \max_{d_1,...,d_{m-1}} \sum_{x_1,...x_n} \prod_{F_i \in F_{OUT}^{d_m}} \lambda_i(F_i)\gamma(W_{d_m}),$$

where $W_{d_m} = \cup_i \{F_i|F_i \in F_{IN}^{d_m}\} \bigcup \cup_j\{Y_j|Y_j \in Y_{IN}^{d_m}\} - \{d_m\}$, and $\gamma(W_{d_m}) = \max_{d_m} \prod_{F_i \in F_{IN}^{d_m}} \lambda_i(F_i) \{\sum_{Y_j \in Y_{OUT}^{d_m}} \mu_j(Y_j) + \sum_{Y_j \in Y_{IN}^{d_m}} \mu_j(Y_j)\}$. The function $\gamma(W_{d_m})$ is computed in the bucket of d_m and added to the bucket of highest ordered variable in W_{d_m}.

Algorithm elim-meu-id can be applied to NG, an N-stage interaction graph of a finite-horizon MDP in order to find a policy $\pi = (d_1, ...d_m)$. It can be also incorporated in dynamic programming techniques for solving infinite-horizon problems, for example, as policy evaluation step of the policy iteration algorithm.

We can show that

Theorem 1: *Given an interaction graph NG of a finite-horizon MDP, and an ordering of its nodes*

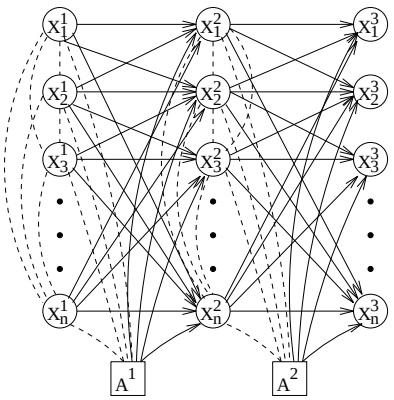

Figure 4: Complete interstate dependence

o, algorithm elim-meu-id can find an optimal plan in $O(N(n+m)exp(w_o^*))$ *time, where* N *is the number of decision epochs,* n *and* m *are the numbers of the state and action variables, respectively, and* w_o^* *is the induced width of NG along the ordering o.*

It is known that finding a minimum-induced-width ordering for an arbitrary graph is an NP-hard problem [1]. We intend to exploit the regularity of NG which is just an "unfolded" 2G in case of stationary MDPs.

Assume that all state variables depend on each other, and on all the variables and actions at the previous time slice (see an example in Figure 4). Each two-slice segment of NG forms a clique of size $2n+m$ so NG is a chain of such cliques. Easy to see that an ordering $(X^1, A^1, ..., A^{N-1}, X^N)$, where X^t denotes an arbitrary sequence of $X_i^t, i = 1, ..., n$, has the induced width $2n+m-1$. We will call any such ordering *natural*, because it follows the sequence of decision epochs.

Clearly,

Proposition 1: *Given a 2G with* n *state variables, and* m *action variables, the induced width of the corresponding NG is bounded by* $2n+m-1$.

Consequently,

Proposition 2: *The minimum induced width ordering for an arbitrary NG can be computed in time polynomial in the number of time slices, but exponential in* n *and* m.

We would like to have a bound on induced width of NG, given a $2G$, such that $w^*(NG) = O(w^*(2G))$. Unfortunately, this is impossible for arbitrary networks. For example, it is known that $n \times N$ grid (NG) has $w^* = n$, while w^* of the corresponding 2-slice grid $(2G)$ w^* is 2. Therefore, finding tractable MDP classes having $w^*(NG) = O(w^*(2G))$ is an open research problem.

Conclusions

We considered the specific structure of MDPs imposed by variable independence that implies decomposability of transitional probabilities, rewards, and policies. This structure is captured by an interaction graph of an MDP. We propose a general bucket elimination technique for computing maximum expected utility on influence diagrams (elim-meu-id), and apply it to MDPs. While the traditional dynamic programming approach for MDPs almost exclusively followed the temporal ordering of the problem, a general elimination algorithm such as elim-meu-id can improve performance by using a more flexible ordering derived from the interaction graph structure.

References

[1] Arnborg, S., Corneil, D.G., and Proskurowski, A., Complexity of Finding Embedding in a k-tree, *Journal of SIAM, Algebraic Discrete Methods*, 8(2):177-184 (1987).

[2] R. Bellman, Dynamic Programming. Princeton University Press, 1957.

[3] R. Bellman, *Adaptive Control Processes: A Guided Tour*, Princeton University Press, Princeton, New Jersey, 1961.

[4] Bertele, U. and Brioschi, F., *Nonserial Dynamic Programming*, Academic Press, New York, 1972.

[5] C. Boutilier, T. Dean, and S. Hanks, Planning under uncertainty: structural assumptions and computational leverage, *EWSP*, 1995.

[6] T. Dean, *Decision Theoretic Planning and Markov Decision Processes*, a tutorial presented at the Summer Institute on Probability and Artificial Intelligence, Corvalis, Oregon, 1994.

[7] R. Dearden and C. Boutilier, Abstraction and approximate decision theoretic planning. Unpublished manuscript.

[8] R. Dechter, Constraint networks, *Encyclopedia of Artificial Intelligence* (2nd Ed.), John Wiley, New York, 1991 pp. 276-285.

[9] R. Dechter, Bucket Elimination: A Unifying framework for several probabilistic inference algorithms, to appear in *Proceedings of UAI-96*, Portland, 1996.

[10] R. Dechter and J. Pearl, Network-based heuristics for constraint satisfaction problems. In *Artificial Intelligence*, 34, pp. 1-38, 1987.

[11] E.C. Freuder, A sufficient condition for backtrack-free search. *Journal of the ACM*, 29, 1982, 24-32.

[12] R.A. Howard, *Dynamic Programming and Markov Processes*, MIT Press, Cambridge, Massachusetts, 1960.

[13] J. Pearl, *Probabilistic reasoning in intelligent systems: networks of plauseible inference*, Morgan Kaufmann, 1988.

[14] M.L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*, John Wiley & Sons, 1994.