Max-norm Projections for Factored MDPs

Carlos Guestrin
Computer Science Dept.
Stanford University
Stanford, CA 94305-9010
guestrin@cs.stanford.edu

Daphne Koller
Computer Science Dept.
Stanford University
Stanford, CA 94305-9010
koller@cs.stanford.edu

Ronald Parr
Computer Science Dept.
Duke University
LSRC / Box 90129,
Durham, NC 27708
parr@cs.duke.edu

Abstract

Markov Decision Processes (MDPs) provide a coherent mathematical framework for planning under uncertainty. However, exact MDP solution algorithms require the manipulation of a *value function*, which specifies a value for each state in the system. Most real-world MDPs are too large for such a representation to be feasible, preventing the use of exact MDP algorithms. Various approximate solution algorithms have been proposed, many of which use a linear combination of basis functions to provide a compact approximation to the value function. Almost all of these algorithms use an approximation based on the (weighted) $\mathcal{L}_2$-norm (Euclidean distance); this approach prevents the application of standard convergence results for MDP algorithms, all of which use max-norm. This paper makes two contributions. First, it presents the first approximate MDP solution algorithms — both value and policy iteration — that use max-norm projection, thereby directly optimizing the quantity required to obtain the best error bounds. Second, it shows how these algorithms can be applied efficiently in the context of *factored MDPs*, where the transition model is specified using a dynamic Bayesian network and actions may be taken sequentially or in parallel.

1 Introduction

Over the last few years, *Markov Decision Processes (MDPs)* have been used as the basic semantics for optimal planning for decision theoretic agents in stochastic environments. In the MDP framework, the system is modeled via a set of states which evolve stochastically. The key problem with this representation is that, in virtually any real-life domain, the state space is quite large. However, many large MDPs have significant internal structure, and can be modeled compactly if the structure is exploited in the representation.

*F*actored MDPs [Boutilier *et al.* 1999] are one approach to representing large structured MDPs compactly. In this framework, a state is implicitly described by an assignment to some set of *state variables*. A *dynamic Bayesian network (DBN)* [Dean and Kanazawa 1989] can then allow a compact representation of the transition model, by exploiting the fact that the transition of a variable often depends only on a small number of other variables. Furthermore, the momentary rewards can often also be decomposed as a sum of rewards related to individual variables or small clusters of variables.

Even when a large MDP can be represented compactly, e.g., in a factored way, solving it exactly is still intractable: Exact MDP solution algorithms require the manipulation of a value function, whose representation is linear in the number of states, which is exponential in the number of state variables. One approach is to approximate the solution using an approximate value function with a compact representation. A common choice is the use of *linear* value functions as an approximation — value functions that are a linear combination of basis functions.

This paper makes a twofold contribution. First, we provide a new approach for approximately solving MDPs using a linear value function. Previous approaches to linear function approximation typically have utilized a least squares ($\mathcal{L}_2$-norm) approximation to the value function. Least squares approximations are generally incompatible with most convergence analyses for MDPs, which are based on max-norm. We provide the first MDP solution algorithms — both value iteration and policy iteration — that use a linear max-norm projection to approximate the value function, thereby directly optimizing the quantity required to obtain the best error bounds.

Second, we show how to exploit the structure of the problem in order to apply this technique to factored MDPs. Our work builds on the ideas of Koller and Parr [1999; 2000], by using *factored (linear) value functions*, where each basis function is restricted to some small subset of the domain variables. We show that, for a factored MDP and factored value functions, various key operations can be implemented in closed form without enumerating the entire state space. Thus, our max-norm algorithms can be implemented efficiently, even though the size of the state space grows exponentially in the number of variables.

We describe how our max-norm algorithm can be applied in the context of various action models, including multiple effects with serial actions, where each action can affect a different part of the system and one must, therefore, choose which part of the system to act on at every time step; and the parallel actions model, where several actions are taken simultaneously. The parallel action case is applicable to a system composed of several subsystems that are controlled simultaneously. We prove that, if these subsystems interact weakly, then our algorithm can generate a good approximation of the value function.

2 Markov Decision Processes

A *Markov Decision Process (MDP)* is defined as a 4-tuple (S, A, R, P) where: S is a finite set of $|S| = N$ states; A is a set of actions; R is a *reward function* $R : S \times A \mapsto \mathbb{R}$, such that $R(s, a)$ represents the reward obtained by the agent in state s after taking action a; and P is a *Markovian transition model* where $P(s' \mid s, a)$ represents the probability of going from state s to state s' with action a.

A stationary policy π for an MDP is a mapping $\pi : S \mapsto A$, where $\pi(s)$ is the action the agent takes at state s. It is associated with a *value function* $\mathcal{V}_\pi \in \mathbb{R}^{|S|}$, where $\mathcal{V}_\pi(s)$ is the discounted cumulative value that the agent gets if it starts at state s. We will be assuming that the MDP has an infinite horizon and that future rewards are discounted exponentially with a discount factor $\gamma \in [0, 1)$. The value function for a fixed policy must be a fixed point of a set of equations that define the value of a state in terms of the value of its possible successor states. More formally, we define:

Definition 2.1 *The* DP operator, $\mathcal{T}_\pi$, *for a fixed stationary policy* π *is:*

$$\mathcal{T}_\pi \mathcal{V}(s) = R(s, \pi(s)) + \gamma \Sigma_{s'} P(s'|s, \pi(s)) \mathcal{V}(s').$$

$\mathcal{V}_\pi$ *is the fixed point of* $\mathcal{T}_\pi$*:* $\mathcal{V}_\pi = \mathcal{T}_\pi \mathcal{V}_\pi$. ∎

The optimal value function $\mathcal{V}^*$ is also defined by a set of equations. In this case, the value of a state must be the maximal value achievable by any action at that state. More precisely, we define:

Definition 2.2 *The* Bellman operator, $\mathcal{T}^*$, *is:*

$$\mathcal{T}^* \mathcal{V}(s) = \max_a [R(s, a) + \gamma \Sigma_{s'} P(s' \mid s, a) \mathcal{V}(s')].$$

$\mathcal{V}^*$ *is the fixed point of* $\mathcal{T}^*$*:* $\mathcal{V}^* = \mathcal{T}^* \mathcal{V}^*$. ∎

For any value function $\mathcal{V}$, we can define the policy obtained by acting greedily relative to $\mathcal{V}$. In other words, at each state, we take the action that maximizes the one-step utility, assuming that $\mathcal{V}$ represents our long-term utility achieved at the next state. More precisely, we define

$$Greedy(\mathcal{V})(s) = \arg\max_a [R(s, a) + \gamma \sum_{s'} P(s' \mid s, a) \mathcal{V}(s')].$$

The greedy policy relative to the optimal value function $\mathcal{V}^*$ is the optimal policy $\pi^* = Greedy(\mathcal{V}^*)$. There are several algorithms to compute the optimal policy, we will focus on the two most used: value iteration and policy iteration.

Value iteration relies on the fact that the Bellman operator is a *contraction* — it is guaranteed to reduce the *max-norm* ($\mathcal{L}_\infty$) distance between any pair of value functions by a factor of at least γ. This property guarantees that the Bellman operator has a unique fixed point $\mathcal{V}^*$ [Puterman 1994]. Value iteration exploits this property, approaching the fixed point through successive applications of the Bellman operator: $\mathcal{V}^{(t+1)} = \mathcal{T}^* \mathcal{V}^{(t)}$. After some number of iterations, the greedy policy $Greedy(\mathcal{V}^{(t)})$ will be the optimal policy.

Policy iteration iterates over policies. Each iteration consists of two phases. *Value determination* computes, for a policy $\pi^{(t)}$, the value function $\mathcal{V}_{\pi^{(t)}}$, by finding the fixed point of: $\mathcal{T}_{\pi^{(t)}} \mathcal{V}_{\pi^{(t)}} = \mathcal{V}_{\pi^{(t)}}$. *Policy improvement* defines the next policy as $\pi^{(t+1)}(s) = Greedy(\mathcal{V}_{\pi^{(t)}})$. It can be shown that this process converges to the optimal policy.

3 Solving MDPs with Max-norm Projections

In many domains, the state space is very large, and we need to perform our computations using approximate value functions. A very popular choice is to approximate a value function using *linear regression*. Here, we define our space of allowable value functions $\mathcal{V} \in \mathcal{H} \subseteq \mathbb{R}^{|S|}$ via a set of *basis functions* $H = \{h_1, \ldots, h_k\}$. A *linear value function* over H is a function $\mathcal{V}$ that can be written as $V = \sum_{j=1}^k w_j h_j$ for some coefficients $\mathbf{w} = (w_1, \ldots, w_k)$. We define $\mathcal{H}$ to be the linear subspace of $\mathbb{R}^{|S|}$ spanned by the basis functions H. It is useful to define an $|S| \times k$ matrix A whose columns are the k basis functions, viewed as vectors. Our approximate value function is then represented by $A\mathbf{w}$.

Linear value functions. The idea of using linear value functions has been explored previously [Tsitsiklis and Van Roy 1996; Koller and Parr 1999; 2000]. The basic idea is as follows: in the solution algorithms, whether value iteration or policy iteration, we use only value functions within $\mathcal{H}$. Whenever the algorithm takes a step that results in a value function $\mathcal{V}$ that is outside this space, we *project* the result back into the space by finding the value function within the space which is close to $\mathcal{V}$. More precisely:

Definition 3.1 *A* projection operator Π *is a mapping* $\Pi : \mathbb{R}^{|S|} \rightarrow \mathcal{H}$. Π *is said to be a* projection w.r.t. a norm $\|\cdot\|$ *if:* $\Pi\mathcal{V} = A\mathbf{w}^*$ *such that* $\mathbf{w}^* \in \arg\min_{\mathbf{w}} \|A\mathbf{w} - \mathcal{V}\|$. ∎

Unfortunately, these existing algorithms all suffer from a problem that we might call "norm incompatibility." When computing the projection, they all utilize the standard projection operator with respect to $\mathcal{L}_2$ norm or a *weighted* $\mathcal{L}_2$ norm. On the other hand, most of the convergence and error analyses for MDP algorithms utilize max-norm. This incompatibility has made it difficult to provide error guarantees when the projection operator is combined with the Bellman operator.

In this section, we propose a new approach that addresses the issue of norm compatibility. Our key idea is the use of a projection operator in $\mathcal{L}_\infty$ norm. This problem has been studied in the optimization literature as the problem of finding the Chebyshev solution to an overdetermined linear system of equations [Cheney 1982]. The problem is defined as finding $\mathbf{w}^*$ such that:

$$\mathbf{w}^* \in \arg\min_{\mathbf{w}} \|C\mathbf{w} - \mathbf{b}\|_\infty . \quad (1)$$

We will use an algorithm due to Stiefel [1960], that solves this problem by linear programming:

$$\begin{array}{ll} \text{Variables:} & w_1, \ldots, w_k, \phi \,; \\ \text{Minimize:} & \phi \,; \\ \text{Subject to:} & \phi \geq \sum_{j=1}^k c_{ij} w_j - b_i \quad \text{and} \\ & \phi \geq b_i - \sum_{j=1}^k c_{ij} w_j, \quad i = 1 \ldots |S| . \end{array} \quad (2)$$

For the solution $(\mathbf{w}^*, \phi^*)$ of this linear program, $\mathbf{w}^*$ is the solution of Eq. (1) and ϕ is the $\mathcal{L}_\infty$ projection error.

Value iteration. The basic idea of approximate value iteration is quite simple. We define an $\mathcal{L}_\infty$ projection operator Π_∞ that takes a value function $\mathcal{V}$ and finds $\mathbf{w}$ that minimizes $\|A\mathbf{w} - \mathcal{V}\|_\infty$. This is an instance of Eq. (1) and can be solved using Eq. (2). The algorithm alternates applications of the Bellman operator $\mathcal{T}^*$ and the projection step Π_∞:

$$\begin{aligned}\overline{\mathcal{V}}^{(t+1)} &= \mathcal{T}^* A\mathbf{w}^{(t)} \\ A\mathbf{w}^{(t+1)} &= \Pi_\infty \overline{\mathcal{V}}^{(t+1)}.\end{aligned}$$

We can analyze this process, bounding the overall error using the single-step projection errors $\phi^{(t)} = \left\|\Pi_\infty \overline{\mathcal{V}}^{(t)} - \overline{\mathcal{V}}^{(t)}\right\|_\infty$. Although these quantities may grow unboundedly in the worst case, our $\mathcal{L}_\infty$ projection minimizes them directly, thereby obtaining the best possible bounds. We omit details for lack of space.

Policy iteration As we discussed, policy iteration is composed of two steps: value determination and policy improvement. Our algorithm performs the policy improvement step exactly. In the value determination step, the value function is approximated through a linear combination of basis functions. Consider the value determination for a policy $\pi^{(t)}$. Define $R_{\pi^{(t)}}(s) = R(s, \pi^{(t)}(s))$, and $P_{\pi^{(t)}}(s' \mid s) = P(s' \mid s, a = \pi^{(t)}(s))$. We can now rewrite the value determination step in terms of matrices and vectors. If we view $\mathcal{V}_{\pi^{(t)}}$ and $R_{\pi^{(t)}}$ as $|S|$-vectors, and $P_{\pi^{(t)}}$ as an $|S| \times |S|$ matrix, we have the equations: $\mathcal{V}_{\pi^{(t)}} = R_{\pi^{(t)}} + \gamma P_{\pi^{(t)}} \mathcal{V}_{\pi^{(t)}}$. This is a system of linear equations with one equation for each state. Our goal is to provide an approximate solution, within $\mathcal{H}$. More precisely, we want to find:

$$\mathbf{w}^{(t)} = \arg\min_{\mathbf{w}} \left\|A\mathbf{w} - \left(R_{\pi^{(t)}} + \gamma P_{\pi^{(t)}} A\mathbf{w}\right)\right\|_\infty .$$

This minimization is another instance of an $\mathcal{L}_\infty$ projection (Eq. (1)), and can be solved using the linear program of Eq. (2). Thus, our approximate policy iteration iteratively alternates two steps:

$$\begin{aligned}\mathbf{w}^{(t)} &= \arg\min_{\mathbf{w}} \left\|A\mathbf{w} - \left(R_{\pi^{(t)}} + \gamma P_{\pi^{(t)}} A\mathbf{w}\right)\right\|_\infty ; \\ \pi^{(t+1)} &= \mathit{Greedy}(A\mathbf{w}^{(t)}).\end{aligned}$$

For the analysis, we define a notion of projection error for the policy iteration case, i.e., the error resulting from the approximate value determination step: $\beta^{(t)} = \left\|A\mathbf{w}^{(t)} - \left(R_{\pi^{(t)}} + \gamma P_{\pi^{(t)}} A\mathbf{w}^{(t)}\right)\right\|_\infty$. We define $\overline{\beta}^{(t)} = \beta^{(t)} + \gamma\overline{\beta}^{(t-1)}$; $\overline{\beta}^{(0)} = 0$.

Lemma 3.2 *There exists a constant $\beta_P < \infty$ such that $\beta_P \geq \beta^{(t)}$ for all iterations t of the algorithm.*

We can now bound the error in the value function resulting from our algorithm:

Theorem 3.3 *In the approximate policy iteration algorithm, the distance between our approximate value function at iteration t and the optimal value function is bounded by:*

$$\begin{aligned}\left\|A\mathbf{w}^{(t)} - \mathcal{V}^*\right\|_\infty &\leq \gamma^t \left\|A\mathbf{w}^{(0)} - \mathcal{V}^*\right\|_\infty + \frac{2\gamma\overline{\beta}^{(t)}}{(1-\gamma)^2}; \\ \textit{where: } \overline{\beta}^{(t)} &\leq \frac{\beta_P(1-\gamma^t)}{1-\gamma}. \end{aligned} \tag{3}$$

Note that this analysis is precisely compatible with our choice of $\mathbf{w}^{(t)}$ as the one that minimizes $\left\|A\mathbf{w}^{(t)} - \left(R_{\pi^{(t)}} + \gamma P_{\pi^{(t)}} A\mathbf{w}^{(t)}\right)\right\|_\infty$, as this choice is precisely designed to minimize the $\beta^{(t)}$ that appears as the bound in our analysis.

4 Solving Factored MDPs

4.1 Factored MDPs

Our presentation of factored MDPs follows that of [Koller and Parr 2000]. In a factored MDP, the set of states is described via a set of random variables $\mathbf{X} = \{X_1, \ldots, X_n\}$, where each X_i takes on values in some finite domain $\mathrm{Dom}(X_i)$. A state $\mathbf{x}$ defines a value $x_i \in \mathrm{Dom}(X_i)$ for each variable X_i. We define a state transition model τ using a *dynamic Bayesian network (DBN)* [Dean and Kanazawa 1989]. Let X_i denote the variable X_i at the current time and X'_i the variable at the next step. The *transition graph* of a DBN is a two-layer directed acyclic graph G_τ whose nodes are $\{X_1, \ldots, X_n, X'_1, \ldots, X'_n\}$. We denote the parents of X'_i in the graph by $\mathit{Parents}_\tau(X'_i)$. For simplicity of exposition, we assume that $\mathit{Parents}_\tau(X'_i) \subseteq \mathbf{X}$; i.e., all arcs in the DBN are between variables in consecutive time slices. (This assumption can be relaxed, but our algorithm becomes somewhat more complex.) Each node X'_i is associated with a *conditional probability distribution (CPD)* $P_\tau(X'_i \mid \mathit{Parents}_\tau(X'_i))$. The transition probability $P_\tau(\mathbf{x}' \mid \mathbf{x})$ is then defined to be $\prod_i P_\tau(x'_i \mid \mathbf{u}_i)$, where $\mathbf{u}_i$ is the value in $\mathbf{x}$ of the variables in $\mathit{Parents}_\tau(X'_i)$.

Consider, for example, the problem of optimizing the behavior of a system administrator maintaining a network of n computers. Each machine is connected to some subset of other machines. In one simple network, we might connect the machines in a ring, with machine i connected to machines $i+1$ and $i-1$. (In this example, we assume addition and subtraction are performed modulo n). Each machine is associated with a binary random variable F_i, representing whether it has failed. The parents of F'_i are F_i, F_{i-1}, F_{i+1}. The CPD of F'_i is such that if $F_i = \mathit{true}$, then $X'_i = \mathit{true}$ with high probability; i.e., failures tend to persist. If $F_i = \mathit{false}$, then F'_i is a noisy or of its two other parents; i.e., a failure in either of its neighbors can independently cause machine i to fail.

We can define the transition dynamics of an MDP by defining a separate DBN model $\tau_a = \langle G_a, P_a \rangle$ for each action a. However, it is often useful to introduce a *default transition model* and describe actions in terms of their modifications to the default model [Boutilier *et al.* 1999]. Typically, actions will alter the transition probabilities of only a small subset of the variables. Therefore, as in [Koller and Parr 2000], we use the notion of a *default transition model* $\tau_d = \langle G_d, P_d \rangle$. For each action a, we define $\mathit{Effects}[a] \subseteq$

$\mathbf{X}'$ to be the variables in the next state whose local probability model is different from τ_d, i.e., those variables X_i' such that $P_a(X_i' \mid \mathit{Parents}_a(X_i')) \neq P_d(X_i' \mid \mathit{Parents}_d(X_i'))$.

In our system administrator example, we might an action a_i for rebooting machine i, and a default action d for doing nothing. The transition model described above corresponds to the "do nothing" action, which is also the default transition model. The transition model for a_i is different from d only in the transition model for the variable F_i', which is now $F_i' = \mathit{true}$ with high probability, regardless of the status of the neighboring machines.

Finally, we need to provide a compact representation of the reward function. We assume that the reward function is factored additively into a set of localized reward functions, each of which only depends on a small set of variables.

Definition 4.1 *A function f is* restricted *to a domain $\mathbf{C} \subseteq \mathbf{X}$ if $f : \mathrm{Dom}(\mathbf{C}) \mapsto \mathbb{R}$. If f is restricted to $\mathbf{Y}$ and $\mathbf{Y} \subset \mathbf{Z}$, we will use $f(\mathbf{z})$ as shorthand for $f(\mathbf{y})$ where $\mathbf{y}$ is the part of the instantiation $\mathbf{z}$ that corresponds to variables in $\mathbf{Y}$.*

Let $R_1, \ldots, R_r$ be a set of functions, where each R_i is restricted to variable cluster $\mathbf{W}_i \subset \{X_1, \ldots, X_n\}$. The reward function for state $\mathbf{x}$ is defined to be $\sum_{i=1}^{r} R_i(\mathbf{x}) \in \mathbb{R}$. In our example, we might have a reward function associated with each machine i, which depends on F_i.

Factorization allows us to represent large MDPs very compactly. However, we must still address the problem of solving the resulting MDP. Our solution algorithms rely on the ability to represent value functions and policies, the representation of which requires the same number of parameters as the size of the space. One might be tempted to believe that factored transition dynamics and rewards would result in a factored value function, which can thereby be represented compactly. Unfortunately, even in factored MDPs, the value function rarely has any internal structure [Koller and Parr 1999].

Koller and Parr [1999] suggest that there are many domains where our value function might be "close" to structured, i.e., well-approximated using a linear combination of functions each of which refers only to a small number of variables. More precisely, they define a value function to be a *factored (linear) value function* if it is a linear value function over the basis $h_1, \ldots, h_k$, where each h_i is restricted to some subset of variables $\mathbf{C}_i$. In our example, we might have basis functions whose domains are pairs of neighboring machines, e.g., one basis function which is an indicator function for each of the four combinations of values for the pair of failure variables.

As shown by Koller and Parr [1999; 2000], factored value functions provide the key to doing efficient computations over the exponential-sized state sets that we have in factored MDPs. The key insight is that restricted domain functions (including our basis functions) allow certain basic operations to be implemented very efficiently. In the remainder of this section, we will show that this key insight also applies in the context of our algorithm.

4.2 Factored Max-norm Projection

The key computational step in both of our algorithms is the solution of Eq. (1) using the linear program in Eq. (2). In our setting, the vectors $\mathbf{b}$ and $C\mathbf{w}$ are vectors in $\mathbb{R}^{|S|}$. The LP tries to minimize the largest distance between b_i and $(C\mathbf{w})_i$ over all of the exponentially many states s_i. However, both $\mathbf{b}$ and $C\mathbf{w}$ are factored, as discussed in the previous section. Hence, our goal is to solve Eq. (1) without explicitly considering each of the exponentially many states. We tackle this problem in two steps.

Maximizing over the state space. First, assume that $\mathbf{b}$ and $C\mathbf{w}$ are given, and that our goal is simply to compute $\max_i((C\mathbf{w})_i - b_i)$. This computation is a special case of a more general task: maximizing a factored linear function over the set of states. In other words, we have a factored linear function F, and we want to compute $\max_{\mathbf{x}} F(\mathbf{x})$. Our goal is to find the state $\mathbf{x}$ over which the value of this expression is maximized. To avoid enumerating all the states, we must use the fact that F has a factored representation as $\sum_j f_j$ where each f_j is a restricted domain function.

We can maximize such a function F using a construction called a *cost network* [Dechter 1999], whose structure is very similar to a Bayesian network. We review this construction here, because it is a key component in our solution to the linear program Eq. (2).

Let $F = \sum_{j=1}^{m} f_j$, and let f_j be a restricted domain function with domain $\mathbf{Z}_j$. Our goal is to compute

$$\max_{x_1,\ldots,x_n} \sum_j f_j(\mathbf{Z}_j[\mathbf{x}])$$

where $\mathbf{Z}_j[\mathbf{x}]$ is the instantiation of the variables in $\mathbf{Z}_j$ in the assignment $\mathbf{x}$. The key idea is that, rather than summing up all the functions and the doing the maximization, we maximize over the variables one at a time. When maximizing over x_l, only summands that involve x_l participate in the maximization. For example, consider:

$$\max_{x_1,x_2,x_3,x_4} f_1(x_1,x_2) + f_2(x_1,x_3) + f_3(x_2,x_4) + f_4(x_3,x_4)$$

We can first compute the maximum over x_4; the functions f_2 and f_3 are irrelevant, so we can push them out. We get

$$\max_{x_1,x_2,x_3} f_1(x_1,x_2) + f_2(x_1,x_3) + \max_{x_4}[f_1(x_2,x_4) + f_4(x_3,x_4)]$$

The result of the internal maximization depends on the values of x_2, x_3; i.e., we can introduce a new function $e_1(X_2, X_3)$ whose value at the point x_2, x_3 is the value of the internal $\max$ expression. Our problem now reduces to computing

$$\max_{x_1,x_2,x_3} f_1(x_1,x_2) + f_2(x_1,x_3) + e_1(x_2,x_3)$$

which has one fewer variable. We continue eliminating variables one at a time, until we have eliminated all of them; the result at that time is a number, which is the desired maximum over $x_1, \ldots, x_4$.

In general, the variable elimination algorithm is as follows. It maintains a set $\mathcal{F}$ of functions, which initially contains $\{f_1, \ldots, f_m\}$. The algorithm then repeats the following steps:

1. Select an uneliminated variable X_l;
2. Extract from $\mathcal{F}$ all functions $e_1, \ldots, e_L$ whose domain contains X_l.
3. Define a new function $e = \max_{x_l} \sum_j e_j$ and introduce it into $\mathcal{F}$. The domain of e is $\cup_{j=1}^{L} \mathrm{Dom}[e_j] - \{X_l\}$.

The computational cost of this algorithm is linear in the number of new "function values" introduced in the elimination process. More precisely, consider the computation of a new function e whose domain is $\mathbf{Z}$. To compute this function, we need to compute $|\mathrm{Dom}[\mathbf{Z}]|$ different values. The cost of the algorithm is linear in the overall number of these values, introduced throughout the algorithm. As shown in [Dechter 1999], this cost is exponential in the induced width of the undirected graph defined over the variables $X_1, \ldots, X_n$, with an edge between X_l and X_m if they appear together in one of our original functions f_j.

Factored LP. Now, consider our original problem of maximizing $\|C\mathbf{w} - \mathbf{b}\|_\infty$, where both $C\mathbf{w}$ and $\mathbf{b}$ are factored. As in Eq. (2), we want to construct a linear program which performs this optimization. However, we want a compact LP, that avoids an explicit enumeration of the constraints for the exponentially many states. The first key insight is that we can replace the entire set of constraints — $(C\mathbf{w})_i - b_i \leq \phi$ for all states i — by the equivalent constraint $\max_i((C\mathbf{w})_i - b_i) \leq \phi$. The second key insight is that this new constraint can be implemented using a construction that follows the structure of the cost network. An identical construction applies to the complementary constraints: $b_i - (C\mathbf{w})_i \leq \phi$.

More precisely, $C\mathbf{w} - \mathbf{b}$, viewed as a function of the state variables, has the form $\sum_j f_j^{\mathbf{w}}$, as above. Here, each $f_j^{\mathbf{w}}$ is either of the form $w_j g_j$, where g_j is one of the columns in C, or simply g_j, where g_j is part of the factorization of $\mathbf{b}$. By assumption, $f_j^{\mathbf{w}}$ has a restricted domain $\mathbf{Z}_j$. We can now mirror the construction of the cost network in order to specify the linear constraints that $\max_{\mathbf{x}}(\sum_j f_j^{\mathbf{w}}(\mathbf{x}))$ must satisfy.

Consider any function e used within $\mathcal{F}$ (including the original f_i's), and let $\mathbf{Z}$ be its domain. For any assignment $\mathbf{z}$ to $\mathbf{Z}$, we introduce a variable into the linear program whose value represents $u_{\mathbf{z}}^{e}$. For the initial functions $f_i^{\mathbf{w}}$, we include the constraint that $u_{\mathbf{z}}^{f_i} = f_i^{\mathbf{w}}(\mathbf{z})$. As $f_i^{\mathbf{w}}$ is linear in $\mathbf{w}$, this constraint is linear in the LP variables. Now, consider a new function e introduced into $\mathcal{F}$ by eliminating a variable X_l. Let $e_1, \ldots, e_L$ be the functions extracted from $\mathcal{F}$, and let $\mathbf{Z}$ be the domain of the resulting e. We introduce a set of constraints:

$$u_{\mathbf{z}}^{e} \geq \sum_{j=1}^{L} u_{(\mathbf{z}, x_l)[\mathbf{Z}_j]}^{e_j} \qquad \forall x_l$$

where $\mathbf{Z}_j$ is the domain of e_j and $(\mathbf{z}, x_l)[\mathbf{Z}_j]$ denotes the value of the instantiation $(\mathbf{z}, x_l)$ restricted to $\mathbf{Z}_j$. Let e_n be the last function generated in the elimination, and recall that its domain is empty. Hence, we have only a single variable u^{e_n}. We introduce the additional constraint $\phi \geq u^{e_n}$.

It is easy to show that minimizing ϕ "drives down" the value of each variable $u_{\mathbf{z}}^{e}$, so that

$$u_{\mathbf{z}}^{e} = \max_{x_i} \sum_{j=1}^{\ell} u_{(\mathbf{z}, x_i)[\mathbf{Z}_j]}^{e_j}.$$

We can then prove, by induction, that u^{e_n} must be equal to $\max_{\mathbf{x}} \sum_j f_j^{\mathbf{w}}(\mathbf{x})$. Our constraints on ϕ ensure that it is greater than this value, which is the maximum of $\sum_j f_j^{\mathbf{w}}(\mathbf{x})$ over the entire state space. The LP, subject to those constraints, will minimize ϕ, guaranteeing that we find the vector $\mathbf{w}$ that achieves the lowest value for this expression.

Returning to our original formulation, we have that $\sum_j f_j^{\mathbf{w}}$ is $C\mathbf{w} - \mathbf{b}$ in one set of constraints and $\mathbf{b} - C\mathbf{w}$ in the other. Hence our new set of constraints is equivalent to the original set: $C\mathbf{w} - \mathbf{b} \leq \phi$ and $\mathbf{b} - C\mathbf{w} \leq \phi$. Minimizing ϕ finds the $\mathbf{w}$ that minimizes the $\mathcal{L}_\infty$ norm, as required.

4.3 Factored Solution Algorithms

The factored max-norm projection algorithm described previously is the key to applying our max-norm solution algorithms in the context of factored MDPs.

Value iteration. Let us begin by considering the value iteration algorithm. As described above, the algorithm repeatedly applies two steps. It first applies the Bellman operator to obtain $\overline{\mathcal{V}}^{(t+1)} = \mathcal{T}^* A\mathbf{w}^{(t)}$. Let $\pi^{(t)}$ be the stationary policy $\pi^{(t)}(s) = \mathit{Greedy}(A\mathbf{w}^{(t)})$. Note that $\pi^{(t)}$ corresponds to the Bellman operator, i.e., $\mathcal{T}_{\pi^{(t)}} = \mathcal{T}^*$. Thus, we can compute $\mathcal{T}^* A\mathbf{w}^{(t)}$ by computing $\pi^{(t)}$ and then performing a *backprojection* operation to compute $\overline{\mathcal{V}}^{(t+1)}$. Assume, for the moment, that $\mathcal{T}_{\pi^{(t)}}$ is a factored transition model; we discuss the computation of the greedy policy and the resulting transition model below. As discussed by Koller and Parr [1999], the backprojection operation can be performed efficiently if the transition model and the value function are both factored appropriately. Furthermore, the resulting function $\overline{\mathcal{V}}^{(t+1)}$ is also factored, although the factors involve larger domains.

To recap their construction briefly, let f be a restricted domain function with domain $\mathbf{Y}$; our goal is to compute $P_\tau f$. We define the *back-projection of* $\mathbf{Y}$ *through* τ as the set of parents of $\mathbf{Y}'$ in the transition graph G_τ — $\Gamma_\tau(\mathbf{Y}') = \cup_{Y' \in \mathbf{Y}'} \mathit{Parents}_\tau(Y')$. It is easy to show that: $(P_\tau f)(\mathbf{x}) = \sum_{\mathbf{y}'} P_\tau(\mathbf{y}' \mid \mathbf{z}) f(\mathbf{y}')$, where $\mathbf{z}$ is the value of $\Gamma_\tau(\mathbf{Y})$ in $\mathbf{x}$. Thus, we see that $(P_\tau f)$ is a function whose domain is restricted to $\Gamma_\tau(\mathbf{Y})$. Note that the cost of the computation depends linearly on $|\mathrm{Dom}(\Gamma_\tau(\mathbf{Y}))|$, which depends on $\mathbf{Y}$ (the domain of f) and on the complexity of the process dynamics.

The second step in value iteration is to compute the projection $A\mathbf{w}^{(t+1)} = \Pi_\infty \overline{\mathcal{V}}^{(t+1)}$, i.e., find $\mathbf{w}^{(t+1)}$ that minimizes $\left\| A\mathbf{w}^{(t+1)} - \overline{\mathcal{V}}^{(t+1)} \right\|_\infty$. As both $\overline{\mathcal{V}}^{(t+1)}$ and $A\mathbf{w}^{(t+1)}$ are factored, we can perform this computation using the factored LP discussed in the previous section.

Policy iteration. Policy iteration also iterates through two steps. The policy improvement step simply computes the

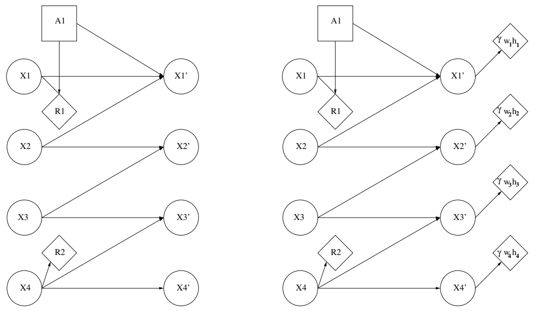

Figure 1: Simple MDP represented as an influence diagram: (a) single decision node case; (b) augmented version, the discounted basis functions, $\gamma w_i h_i$, are added as utility nodes.

greedy policy relative to $\pi^{(t)}$. We discuss this step below. The approximate value determination step computes

$$\mathbf{w}^{(t)} = \arg\min_{\mathbf{w}} \|A\mathbf{w} - (R_{\pi^{(t)}} + \gamma P_{\pi^{(t)}} A\mathbf{w})\|_\infty$$

The difference between this operation and the one used in value iteration is that $\mathbf{w}$ appears in both terms, i.e., the second term is not a constant vector. However, by rearranging the expression, we get

$$\arg\min_{\mathbf{w}} \left\| (A - \gamma P_{\pi^{(t)}} A)\, \mathbf{w}^{(t)} - R_{\pi^{(t)}} \right\|_\infty .$$

Again, assuming that $P_{\pi^{(t)}}$ is factored, we can conclude that $C = (A - \gamma P_{\pi^{(t)}} A)$ is also a matrix whose columns correspond to restricted-domain functions. Thus, we can again apply our factored LP.

Policies In our discussion above, we assumed that we have some mechanism for computing the greedy policy $Greedy(A\mathbf{w}^{(t)})$, and that this policy has a compact representation and a factored transition model. Not all possible actions models result in tractable transition models. However, we are able to show that several realistic action models produce structured policies that are compatible with our factored max-norm projection.

4.4 Action Models

Restricted effect sets The simplest action model we consider is one where all actions have the same, small effects set. In our system administrator example, this would correspond to the case where the system administrator is able to influence the behavior of the network directly through his manipulation of a small number of server machines in his office. The remaining machines are influenced only indirectly as the effects of the system administrator's actions propagate through the network. This can be represented with a simple *dynamic influence diagram* [Howard and Matheson 1984; Russell and Norvig 1994] with only one action node (in an influence diagram variables are represented as circles, rewards as diamonds and actions as squares, Figure 1(a) illustrates a simple example).

Once we have represented the problem with an influence diagram, we can go back to the algorithm. We have a linear value function $\mathcal{V} = A\mathbf{w}$, and our goal is to find its optimal policy $\pi = Greedy(\mathcal{V})$. This computation can be performed efficiently in the factored case by defining an *augmented influence diagram*. Recall that the assumption behind the greedy policy is that, if we take action a at state $\mathbf{x}$, we get the immediate reward $R(\mathbf{x}, a)$, and then get $\gamma\mathcal{V}(\mathbf{x}')$ at the next state $\mathbf{x}'$. We can then define an influence diagram based on the one defining the process dynamics; we need only add utility nodes to represent the value at the next state. For each basis function h_i, add a utility node whose parents are the variables in $\mathbf{C}'_i$ and whose utility function is the discounted weighted basis function: $\gamma w_i^{(t)} h_i$. Figure 1(b) shows the augmentation for our example, we have 4 basis $h_1, \ldots, h_4$ and $h_i = h_i(X_i)$, that is $\mathbf{C}_i = \{X_i\}$.

The result is not an influence diagram, as the nodes in the current state are not associated with a probabilistic model. Rather, it is *conditional* influence diagram; for each assignment of values to the current state variables $\mathbf{X}$, we will have a different decision problem. For any value $\mathbf{x}$ to these variables, the optimal action a in the influence diagram is the greedy action at the corresponding state s, i.e., the one that maximizes $R(\mathbf{x}, a) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, a)\mathcal{V}(\mathbf{x}')$.

This approach seems to demand that we do a separate optimization for each state $\mathbf{x}$. The key insight is that only a subset of the variables $\mathbf{X}$ will be relevant for optimizing our action A. More precisely

Definition 4.2 *For some action node A, let $Influences[A]$ be all utility nodes in the augmented influence diagram that are descendants of A. Let $Control[A]$ be the variables in $\mathbf{X}$ that have a directed path to any node in $Influences[A]$.* ∎

Lemma 4.3 *In the augmented influence diagram, whenever there are two state $\mathbf{x}_1$ and $\mathbf{x}_2$ that agree on the values of the variables in $Control[A]$, then the optimal actions in states $\mathbf{x}_1$ and $\mathbf{x}_2$ are the same.* ∎

Thus, the policy $Greedy(\mathcal{V})$ for the action node A is a function of $Control[A] \subseteq \mathbf{X}$ rather than a function of all variables $\mathbf{X}$. Therefore it can be represented and computed more concisely. In our example, $Influences[A] = \{R_1, \gamma w_1 h_1\}$ and $Control[A] = \{X_1, X_2\}$.

Once the policy has been computed, we need to generate the resulting transition model P_π and reward function R_π. We use a simple transformation of the DBN. For a fixed policy, the node A can be represented as a deterministic node, whose parents are $Control[A]$ and CPD is the policy π. The node A is a deterministic node, so it can be eliminated by adding edges from all variables in $Control[A]$ to each variable in $Effects[A]$. In our example, the structure of the DBN is unchanged. For the reward function, we simply add edges from all variables in $Control[A]$ to all utility functions that are children of A. In terms of the functions, if $R_i(\mathbf{W}_i, A)$ is a restricted domain function of some subset $\mathbf{W}_i$ of the variables and of A, then it will become a restricted domain function of $\mathbf{W}_i \cup Control[A]$. If it does not depend on A, it will be unchanged. In the example, R_2 is unchanged, but R_1 becomes a child of both X_1 and X_2. The resulting DBN could be more complex, but is still generally factored. Thus we can apply the max-norm projection algorithm described in Section 4.2.

Multiple effects with serial actions While there are many domains where only a small number of variables can be directly influenced by actions, a more realistic model is one in which all, or nearly all, of the variables can, in principle, be influenced directly by actions. We make the restriction, however, that only a small number can be influenced by any particular action. In system administrator problem, we free system administrator from his office to move freely through the network rebooting machines as needed. The only constraint is that he can reboot at most one machine per time step.

As show in [Koller and Parr 2000], the greedy policy for this action model relative to a factored value function has the form of a *decision list*. More precisely, the policy can be written in the form $\langle \mathbf{t}_1, a_1\rangle, \langle \mathbf{t}_2, a_2\rangle, \ldots, \langle \mathbf{t}_L, a_L\rangle$, where each $\mathbf{t}_i$ is an assignment of values to some small subset $\mathbf{T}_i$ of variables, and each a_i is an action. The optimal action to take in state $\mathbf{x}$ is the action a_j corresponding to the first event $\mathbf{t}_j$ in the list with which $\mathbf{x}$ is consistent.

Koller and Parr show that the greedy policy can be represented compactly using such a decision list, and provide an efficient algorithm for computing it. Unfortunately, as they discuss, the resulting transition model is usually not factored. Thus, there does not appear to be possible to implement decision list policy with simple augmentation of the DBN, as was the case with restricted effects sets. However, we can modify our max-norm projection operation in manner that essentially implements the decision list policy without ever explicitly constructing decision list transition model.

The basic idea is to introduce two new cost networks corresponding to each branch in the decision list. We then augment each cost network in a way that allows the network's constraints on ϕ to be active only when they are consistent with the current policy. Let S_i be the set of states $\mathbf{x}$ for which $\mathbf{t}_i$ is the first event in the decision list for which $\mathbf{x}$ is consistent. For simplicity of presentation, we will discuss only the construction of the constraints $\phi \geq ((C\mathbf{w})_i - b_i)$, the construction of the other set of constraints, $\phi \geq (b_i - (C\mathbf{w})_i)$, is isomorphic. Recall that our LP construction defines a set of constraints that imply that $\phi \geq ((C\mathbf{w})_i - b_i)$ for each state i. Suppose we could afford to have a separate set of constraints for the states in each subset S_i. For each state in S_i, we know that action a_i is taken. Hence, we could apply our the factored max-norm projection constraints for each state using P_{a_i}.

The only issue is to explain how to achieve this effect with our factored projection by doing work that is proportional to the size of the decision list, not the entire state space. We need to guarantee that at each step of the decision list cost network constraints derived from P_{a_i} are applied only to states in S_i. Specifically, we must guarantee that they are applied only to states consistent with $\mathbf{t}_i$, but not to states that are consistent with some $\mathbf{t}_j$ for $j < i$. To guarantee the first condition, we simply instantiate the variables in $\mathbf{T}_i$ to take the values specified in $\mathbf{t}_i$. That is, our cost network for t_i now considers only the variables in $\{X_1, \ldots, X_n\} - \mathbf{T}_i$, and computes the maximum only over the states consistent with $\mathbf{T}_i = \mathbf{t}_i$. To guarantee the second condition, we ensure that

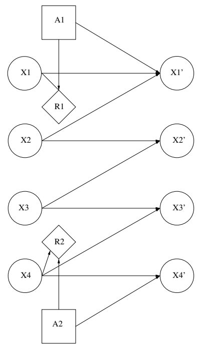

Figure 2: Simple MDP with two decision nodes.

we do not impose any constraints on states associated with previous decisions. This is achieved by adding indicators $\mathcal{I}_j$ for each previous decision $\mathbf{t}_j$, with weight $-\infty$. These will cause the constraints associated with $\mathbf{t}_i$ to be trivially satisfied by states in S_j for $j < i$. Note that each of these indicators is a restricted domain function of $\mathbf{T}_j$ and can be handled in the same fashion as all other terms in the factored LP. Thus, for a decision list of size L, our factored LP contains constraints from $2L$ cost networks.

Multiple effects with parallel actions The last action model we consider is one with parallel actions: at every time step, one must make several decisions and they will all be performed simultaneously. In our network administration example, it could be the case that the system is comprised of several subnets. At every time step, a decision needs to be made for every subnet and they are executed at the same time. The new problem introduced by the parallel actions model is that one can no longer enumerate all possible actions as the number of actions is grows exponentially with the number of decisions that need to be made at every time step. However, this type of systems can be represented compactly in an influence diagram by having several decision nodes. Figure 2 illustrates and example where two decisions are taken simultaneously.

More precisely, consider a problem with g simultaneous decisions $\mathbf{A} = \{A_1, \ldots, A_g\}$. At every time step, each A_i must make a decision from its set of possible actions, $\mathrm{Dom}(A_i)$. As we discussed, the size of the action space $\mathbf{A}$ is exponential in the number of decisions.

Unfortunately, at this point we cannot solve this problem for the general case, so we will need to make two restrictive assumptions: (a) two decision nodes cannot directly influence the same variable, more precisely, $\mathit{Effects}[A_i] \cap \mathit{Effects}[A_j] = \emptyset$; and, (b) two decision nodes cannot influence the same rewards or basis functions, more precisely, $\mathit{Influences}[A_i] \cap \mathit{Influences}[A_j] = \emptyset$. Assumption (a) is not a very restrictive assumption, as in many practical systems this kind of decomposition is present. Assumption (b) seems to be the more restrictive one, as it does not allow us to use basis functions that span the effects of two decision nodes. Fortunately, as we will show below in Theorem 4.5, in systems that are composed of weakly interacting subsystems, we can still achieve good approximations.

Each decision A_i will be associated with a policy: $\pi_i : \mathbf{X} \mapsto \mathrm{Dom}(A_i)$, we will show that, under the assumptions above, each π_i will depend only on a small subset of $\mathbf{X}$. We will denote the policy for all decision nodes as $\pi = \{\pi_1, \ldots, \pi_g\}$.

Previously, we describe how to determine the greedy policy for the single decision node case and then use this policy to generate a factored transition model $P_{\pi^{(t)}}$ that can be applied directly the max-norm projection algorithm described in Section 4.2. To extend these algorithms to the case of parallel actions, we need only to show how to determine the greedy policy. Once we generate $\pi^{(t)} = \{\pi_1^{(t)}, \ldots, \pi_g^{(t)}\}$, we can compute $P_{\pi^{(t)}}$ and $R_{\pi^{(t)}}$ by applying the simple transformations precisely as in the single decision node case.

Finding the greedy policy for the parallel action case is virtually identical to the single decision case. Let $\mathcal{V} = A\mathbf{w}$ be our current value function. Our goal is to compute $\pi = \mathit{Greedy}(\mathcal{V})$, i.e., we want to compute:

$$\pi(s) = \arg\max_{\mathbf{a}} [R(s, \mathbf{a}) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, \mathbf{a}) \sum_i w_i h_i(\mathbf{x}')].$$

Where $\mathbf{a} = \{a_1, \ldots, a_g\}$ is a setting for the actions of all decision nodes at state s. Again, this computation can be performed efficiently in the factored case by defining the *augmented influence diagram* in exactly the same manner. Under the assumptions above, this maximization will be decomposed into g independent maximizations, each one having the same form as the single decision node case. Thus they can be performed efficiently. It is easy to verify that the policy for each decision node A_i is a mapping $\pi_i^{(t)} : \mathit{Control}[A_i] \mapsto \mathrm{Dom}(A_i)$. Hence, the policy for decision can also be represented compactly, and leads to a factored transition model P_π, as required for the remaining phases of the algorithm.

This analysis has another important consequence. Assume that the algorithm terminates with some final (factored) value function. The optimal greedy policy for that value function will require that decision A_i only observe the variables in $\mathit{Control}[A_i]$. Thus, when implementing the resulting controller on a practical system, each decision will only need a subset of the state variables as input.

One point that remains open is how restrictive is the fact, imposed by assumption (b), that we cannot use basis functions that span the effects of two decision nodes. In the following analysis, we show that when a system can be decomposed into subsystems, the effects of a decision node and the basis functions being restricted to a subsystem as in our assumption, and these subsystems are weakly interacting, then we can generate good approximate value functions.

First, we need to define a system and subsystems. Suppose you have a system $\mathcal{B}$ composed of $|\mathcal{B}|$ subsystems B_i, for example a network composed of many subnets. The transition probabilities of subsystem i depends on its own state, on the state of some other subsystems we represent by E_i and on the decision particular to that system A_i, i.e., $P(b_i'|b_i, e_i, a_i)$. Each subsystem will also have its own reward function r_i. Finally, we assume that basis are restricted functions of B_i and that they are composed of indicator

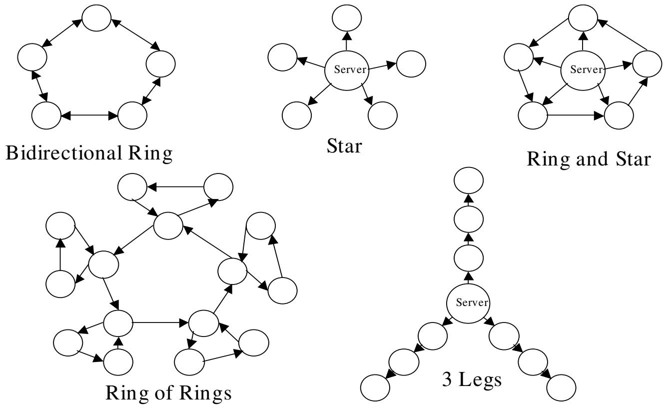

Figure 3: Network topologies tested.

functions for every value of b_i. Thus, our basis can represent any function over the variables in a subsystem, but they do not overlap. Finally, we need to define weak interaction:

Definition 4.4 *A system $\mathcal{B}$ is said to be ε-interacting if:*

$$\max_{b_i, e_i, a_i} \|P(b_i'|b_i, e_i, a_i) - P(b_i'|b_i, a_i)\|_1 \leq \varepsilon \text{ ; for all } i;$$

where $P(b_i'|b_i, a_i) = \frac{\sum_{e_i} P(b_i'|b_i, e_i, a_i)}{|E_i|}$. ∎

That is, if the $\mathcal{L}_1$ distance between the transition probabilities and the transition probabilities with the external effects averaged out is less than ε, than the external effects have small influences on the subsystem and the whole system is weakly interacting.

Using this formulation, we can prove that the value of this system can be well approximated by our approximate policy iteration algorithm. In the following Theorem, we will show that the maximum value determination error in policy iteration, β_P, as defined in Lemma 3.2, will be small when the system is weakly interacting.

Theorem 4.5 *If a system $\mathcal{B}$ is ε-interacting, the maximum value determination error in the approximate policy iteration algorithm is bounded by:*

$$\beta_P \leq \frac{\gamma R_{max}}{1 - \gamma} \varepsilon. \quad ∎ \tag{4}$$

This Theorem gives only sufficient conditions for our algorithm to yield good approximations in a weakly interacting system. Note that these are not necessary conditions and the algorithm may perform well in other cases not analyzed here.

5 Experimental Results

We implemented the factored policy and value iteration algorithms in Matlab, using CPLEX as the LP solver. The algorithm was tested on the SysAdmin problem with various network architectures, shown in Figure 3. At every time step, the SysAdmin could go to one machine and reboot it, causing it to be working in the next time step with high probability. This problem is an instance of the *multiple effects*

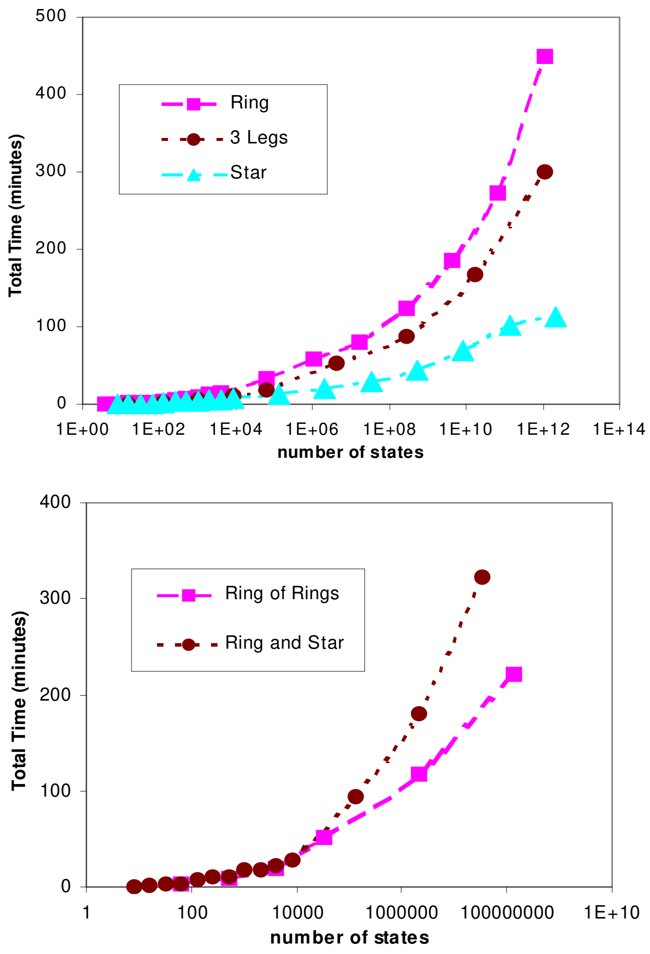

Figure 4: Running times for policy iteration on variants of the SysAdmin problem.

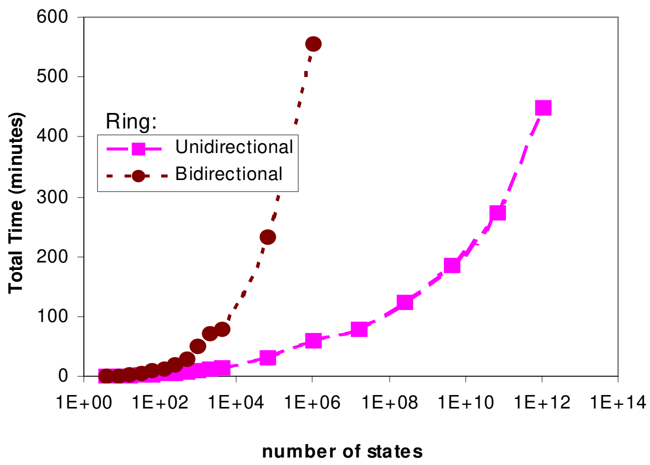

Figure 5: Unidirectional versus bidirectional rings.

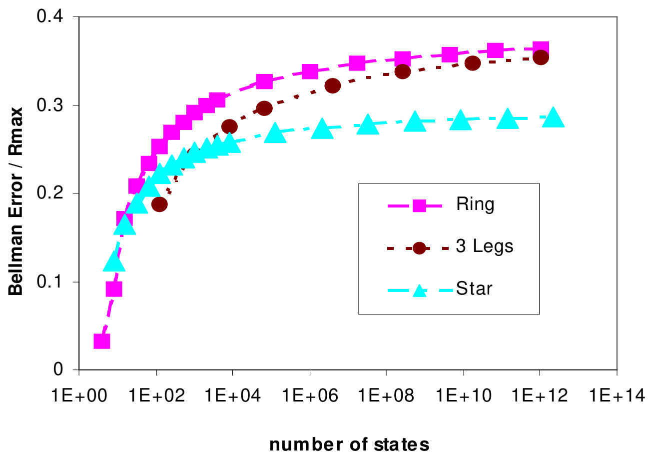

Figure 6: Bellman error after convergence.

with serial actions model described in Section 4.4. Each machine received a reward of 1 when working (except in the ring, where one machine receives a reward of 2, to introduce some asymmetry) and the discount factor is $\gamma = 0.95$. Note that the additive structure of the reward function makes it unsuitable for the tree-structured representation used by Boutilier et al. [1999]. The basis functions included independent indicators for each machine, with value 1 if the machine is working and zero otherwise, and the constant basis, that is value 1 for all states. We present only results for policy iteration. The time per iteration was about equal for policy and value iteration, but policy iteration converged in many fewer iterations. In general, for the systems we tested, policy iteration took about 4 or 5 iterations to converge.

To evaluate the complexity of the algorithm, tests were performed with increasing number of states, that is, increasing number of machines on the network. Figure 4 shows the running time for increasing problem sizes. Note that the number of states is growing exponentially, but running time is increasing only logarithmically in the number of states (about $O(|\mathbf{X}| \cdot |A|)$ for the unidirectional ring architecture).

We also note that the size of the decision list for the final policy grows approximately linearly with the number of machines.

For further evaluation of the algorithm, we compared unidirectional and bidirectional rings, Figure 5. In terms of the DBN, in the unidirectional ring case there are 2 parents per variable, against 3 in the bidirectional case. This change not only increases the sizes of the backprojections, but also the intermediate factors in the cost network computation, which span 3 variables for the unidirectional case, against 5 for the bidirectional one. Thus, as the graph illustrates, the behaviour of the bidirectional ring is still $O(|\mathbf{X}| \cdot |A|)$, but the constants are much larger.

For these large models, we can no longer compute the correct value function, so we cannot evaluate our results by computing $\|\mathcal{V} - \mathcal{V}^*\|_\infty$. However, the *Bellman error*, defined as $\mathrm{BellmanErr}(\mathcal{V}) = \|\mathcal{T}^*\mathcal{V} - \mathcal{V}\|_\infty$, can be used to provide a bound: $\|\mathcal{V}^* - \mathcal{V}\|_\infty \leq \frac{\varepsilon}{1-\gamma}$ [Williams and Baird 1993]. Thus, we use the Bellman error to evaluate our answers for larger models. Figure 6 shows that the Bellman error increases very slowly with the number of states. On small models for which we could compute the optimal policy, the value of the policies produced by our algorithm was very close to optimal.

6 Conclusions

In this paper, we presented new algorithms for approximate value and policy iteration. Unlike previous approaches, our algorithms directly minimize the $\mathcal{L}_\infty$ error, and are thereby more amenable to providing tight bounds on the error of the approximation. We have shown that the $\mathcal{L}_\infty$ error can be minimized using linear programming. We have also provided an approach for representing this LP compactly for factored MDPs with several different action models, allowing our algorithms to be applied efficiently even for MDPs with exponentially large state spaces and parallel, simultaneous actions. In the case of parallel actions, we have demonstrated conditions under which the policy decomposes into a collection of local decisions which individually involve only a small number of variables. We can provide error bound on the resulting policy based upon the strength of the interaction between the local sets of variables.

Our algorithms optimize the max-norm error when approximating the value function, giving them better theoretical performance guarantees than algorithms that optimize the $\mathcal{L}_2$-norm. Moreover, they are substantially easier to implement than $\mathcal{L}_2$-norm methods for factored models. We have shown that these algorithms also perform well in practice, even for some very large problems. There are many possible extensions to this work. In particular, we hope to extend our algorithms and analyses to cover more general classes of action models and context-specific independence.

Acknowledgments

We are very grateful to Dirk Ormoneit for many useful discussions. This work was supported by the ONR under the MURI program "Decision Making Under Uncertainty", by the ARO under the MURI program "Integrated Approach to Intelligent Systems", and by the Sloan Foundation.

References

C. Boutilier, T. Dean, and S. Hanks. Decision theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 1999.

E. W. Cheney. *Approximation Theory*. Chelsea Publishing Co., New York, NY, 2nd edition, 1982.

T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3):142–150, 1989.

R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1–2):41–85, 1999.

R. A. Howard and J. E. Matheson. Influence diagrams. In R. A. Howard and J. E. Matheson, editors, *Readings on the Principles and Applications of Decision Analysis*, pages 721–762. Strategic Decisions Group, Menlo Park, California, 1984.

F. Jensen, F. Jensen, and S. Dittmer. From influence diagrams to junction trees. In *Uncertainty in Artificial Intelligence: Proceedings of the Tenth Conference*, pages 367–373, Seattle, Washington, July 1994. Morgan Kaufmann.

D. Koller and R. Parr. Computing factored value functions for policies in structured MDPs. In *Proc. IJCAI-99*. Morgan Kaufmann, 1999.

D. Koller and R. Parr. Policy iteration for factored mdps. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI-00)*, Stanford, California, June 2000. Morgan Kaufmann.

M. Puterman. *Markov decision processes: Discrete stochastic dynamic programming*. Wiley, New York, 1994.

S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1994.

E. Stiefel. Note on jordan elimination, linear programming and tchebycheff approximation. *Numerische Mathematik*, 2:1 – 17, 1960.

J. N. Tsitsiklis and B. Van Roy. Feature-based methods for large scale dynamic programming. *Machine Learning*, 22:59–94, 1996.

R. Williams and L. Baird. Tight performance bounds on greedy policies based on imperfect value functions. Technical Report NU-CCS-93-14, Northeastern University, 1993.