Efficient Solutions to Factored MDPs with Imprecise Transition Probabilities

Karina Valdivia Delgado
University of Sao Paulo
Sao Paulo, Brazil
kvd@ime.usp.br

Scott Sanner
NICTA & ANU
Canberra, Australia
ssanner@nicta.com.au

Leliane Nunes de Barros
University of Sao Paulo
Sao Paulo, Brazil
leliane@ime.usp.br

Fabio G. Cozman
Escola Politecnica, USP,
Sao Paulo, Brazil
fgcozman@usp.br

Abstract

When modeling real-world decision-theoretic planning problems in the Markov decision process (MDP) framework, it is often impossible to obtain a completely accurate estimate of transition probabilities. For example, natural uncertainty arises in the transition specification due to elicitation of MDP transition models from an expert or data, or non-stationary transition distributions arising from insufficient state knowledge. In the interest of obtaining the most robust policy under transition uncertainty, the Markov Decision Process with Imprecise Transition Probabilities (MDP-IPs) has been introduced to model such scenarios. Unfortunately, while solutions to the MDP-IP are well-known, they require nonlinear optimization and are extremely time-consuming in practice. To address this deficiency, we propose efficient dynamic programming methods to exploit the structure of factored MDP-IPs. Noting that the key computational bottleneck in the solution of MDP-IPs is the need to repeatedly solve nonlinear constrained optimization problems, we show how to target approximation techniques to drastically reduce the computational overhead of the nonlinear solver while producing bounded, approximately optimal solutions. Our results show up to two orders of magnitude speedup in comparison to traditional "flat" dynamic programming approaches and up to an order of magnitude speedup over the extension of factored MDP approximate value iteration techniques to MDP-IPs.

Introduction

Markov decision processes (MDP) (Puterman 1994) have become the *de facto* standard model for decision-theoretic planning problems and a great deal of research in recent years has aimed to exploit structure in order to compactly represent and efficiently solve factored MDPs (Boutilier, Hanks, and Dean 1999; Hoey et al. 1999; St-Aubin, Hoey, and Boutilier 2000; Guestrin et al. 2003). However, in many real-world problems, it is simply impossible to obtain a precise representation of the transition probabilities in an MDP. This may occur for many reasons, including (a) imprecise or conflicting elicitations from experts, (b) insufficient data from which to estimate precise transition models, or (c) non-stationary transition probabilities due to insufficient state information.

For example, in an MDP for traffic light control, it is difficult to estimate the turn probabilities for each traffic lane that has the option of going straight or turning. These lane-turning probabilities may change during the day or throughout the year, as a function of traffic at other intersections, and based on holidays and special events; in general it is impossible to accurately model all of these complex dependencies. In this case it would be ideal to have a traffic control policy optimized over a range of turn probabilities in order to be robust to inherent non-stationarity in the turn probabilities.

To accommodate optimal models of sequential decision-making in the presence of strict uncertainty over the transition model, the MDP with imprecise transition probabilities (MDP-IP) was introduced (Satia and Lave Jr. 1970; White III and El-Deib 1994). While the MDP-IP poses a robust framework for the real-world application of decision-theoretic planning, its solution requires nonlinear optimization, which is extremely time-consuming in practice.

To address this computational deficiency, we extend the factored MDP model to MDP-IPs and propose efficient, scalable algorithms for solving these factored MDP-IPs. This leads to the following novel contributions in this work:

- We propose to replace the usual dynamic Bayes net (DBN) (Dean and Kanazawa 1990) used in factored MDPs, with dynamic credal nets (DCNs) (Cozman 2000) to support compact factored structure in the imprecise transition model of factored MDP-IPs.
- We extend the decision-diagram based SPUDD and APRICODD algorithms for MDPs (Hoey et al. 1999; St-Aubin, Hoey, and Boutilier 2000) to MDP-IP algorithms capable of exploiting DCN structure. For this, we introduce the parameterized ADD (PADD) with polynomial expressions at its leaves and explain how to extend ADD operations to PADDs.
- As shown in the experiments, the generalization of SPUDD and APRICODD to MDP-IPs using PADDs is just the first step in obtaining efficient solutions. Observing that the key computational bottleneck in the solution of MDP-IPs is the need to repeatedly solve nonlinear constrained optimization problems, we show how to target our approximations to drastically reduce the computational overhead of the nonlinear solver while producing bounded, approximately optimal solutions.

As our results will demonstrate, using the above contributions we can obtain up to two orders of magnitude speedup in comparison to traditional "flat" dynamic programming approaches and up to an order of magnitude speedup over the generalization of state-of-the-art approximate factored MDP solvers to accommodate MDP-IPs.

Markov Decision Processes

Formally, a *Markov Decision Process (MDP)* is defined by the tuple $\mathcal{M} = \langle S, A, R, P, \gamma \rangle$ where: S is a finite set of fully observable states, A is a finite set of actions, $R : S \times A \rightarrow \mathbb{R}$ is a fixed reward function associated with every state and action, P defines the transition probabilities, where $P(s'|s, a)$ is the conditional probability of reaching state s' when starting in state $s \in S$ and taking action $a \in A$. γ is a discount factor s.t. $0 \leq \gamma < 1$ where rewards t time steps in the future are discounted by γ^t. A stationary policy $\pi : S \rightarrow A$ indicates the action $a = \pi(s)$ to take in each state s. The value of a policy π is defined as the sum of expected discounted rewards over an infinite horizon starting in state s and following π:

$$V_\pi(s) = E_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t | s = s_0 \right] \quad (1)$$

r_t is the reward obtained at time t. Our objective is to find an optimal policy π^* that yields the maximal value in each state, i.e., $\forall s, \pi' \; V_{\pi^*}(s) \geq V_{\pi'}(s)$.

A well known algorithm to solve an MDP is *value iteration* (Puterman 1994). It constructs a series of t-stage-to-go value functions V^t. Starting with arbitrary V^0, value iteration performs value updates for all states s, computing V^t based on V^{t-1}. The Q-value for state s and action a is:

$$Q^t(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V^{t-1}(s') \quad (2)$$

and the best value attainable at decision stage t and state s is

$$V^t(s) = \max_{a \in A} Q^t(s, a). \quad (3)$$

We define the greedy policy π_V w.r.t. some V as follows:

$$\pi_V(s) = \arg\max_{a \in A} \left(R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V(s') \right) \quad (4)$$

At the infinite horizon, the value function provably converges

$$\lim_{t \rightarrow \infty} \max_s |V^t(s) - V^{t-1}(s)| = 0 \quad (5)$$

leading to a stationary, deterministic optimal policy $\pi^* = \pi_{V^\infty}$ (Puterman 1994). For practical MDP solutions, we are often only concerned with ϵ-optimality. If we terminate the MDP when the following condition is met:

$$\max_s |V^t(s) - V^{t-1}(s)| < \frac{\epsilon(1 - \gamma)}{2\gamma} \quad (6)$$

then we guarantee that the greedy policy π_{V^t} loses no more than ϵ in value over an infinite horizon in comparison to π^* (Puterman 1994).

MDPs with Imprecise Transitions

As described in our introductory traffic example, it is often necessary to work with imprecise probabilities in order to represent incomplete, ambiguous or conflicting expert beliefs about transition probabilities. An *MDP with imprecise transition probabilities (MDP-IP)* is specifically designed for this setting. Note that the term MDP-IP was proposed by White III and Eldeib (White III and El-Deib 1994), while Satia and Lave Jr. (Satia and Lave Jr. 1970) adopt instead the term *MDP with Uncertain Transition Probabilities*.

To specify an MDP-IP, one must specify all elements of an MDP except the transition probabilities; then one must specify a set of probabilities for each transition between states. We refer to these sets as *transition credal sets*. Formally, an MDP-IP is defined by $\mathcal{M}_{IP} = (S, A, R, K, \gamma)$. This definition is identical to the MDP $\mathcal{M}$, except that *the* transition distribution P is replaced with a *set* of distributions K. We will represent K implicitly as the set of transition probabilities consistent with a set of side linear constraints.

There are several optimization criteria that can be used to define the value of a policy in an MDP-IP. In the context of the discounted infinite horizon setting that we focus on in this work, there is always a deterministic stationary policy that is *maximin* (Satia and Lave Jr. 1970); moreover, given the assumption that A is finite and the credal set K is closed, this policy induces an optimal value function that is the unique fixed-point solution of:

$$V^*(s) = \max_{a \in A} \min_{P \in K} \left\{ R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V^*(s') \right\}. \quad (7)$$

The idea of the above MDP-IP solution is to be as robust as possible in the face of the transition uncertainty given by K. In this case, the agent's goal is to maximize the reward gained, assuming that Nature tries to minimize the agent's reward. Nature's policy is not necessarily stationary, it may change over time. In the traffic scenario, we note the traffic patterns may differ on holidays versus normal weekdays even though the controller may not be explicitly aware of the holiday in its state description.

There are various algorithms for solving *flat* (i.e., enumerated state) MDP-IPs based on dynamic programming (Satia and Lave Jr. 1970; White III and El-Deib 1994). In this work, we build on a flat value iteration solution to MDP-IPs (Satia and Lave Jr. 1970):

$$V^t(s) = \max_{a \in A} \min_{P \in K} \left\{ R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) V^{t-1}(s') \right\} \quad (8)$$

Value iteration for MDP-IPs is the same as that given in (2) and (3) for MDPs except that now for every state s, we optimize our action choice $a \in A$ w.r.t. the *worst-case* distribution $P \in K$ that minimizes the future expected value. Thus we ensure that the resulting value function and policy are robust to the worst outcome that Nature could choose.

We note that Nature's *true* transition function P may be non-stationary; Nature can choose a *different* $P \in K$ for *every* action a and *every* state s and *every* time t. Convergence properties of value iteration in (8) still hold when Nature's transitions are non-stationary but bounded by $P \in K$ since (8) optimizes w.r.t. the worst-case (Satia and Lave Jr. 1970).

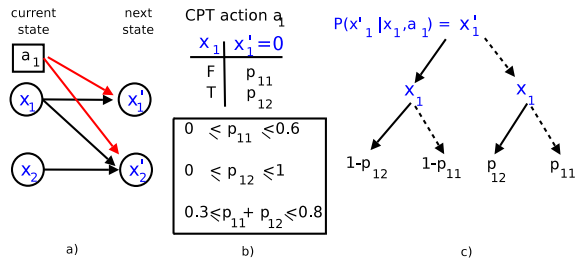

Figure 1: a) Dynamic Credal Network for action $a_1 \in A$. b) Conditional probability table for the state variable X_1' and the constraints related to the probabilities. c) The Parameterized ADD representation for $P(x_1'|x_1, a_1)$ that we call $CPT_{a_1}^{x_1'}$.

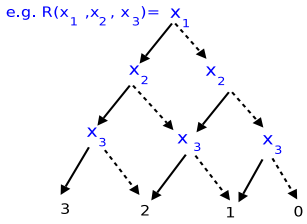

Figure 2: An example reward function $R(x_1, x_2, x_3) = \sum_{i=1}^{3} x_i$ represented as an ADD. A solid line indicates the true (1) branch of a variable test and a dashed line indicates the false (0) branch.

Factored MDP-IPs

We define a *factored MDP-IP* as an MDP-IP where states $\vec{x} \in \{0,1\}^n$ are compactly specified as a joint assignment to a vector of n binary state variables $(X_1, \ldots, X_n)$. While our extensions are not necessarily restricted to binary state variables, we make this restriction here for simplicity of notation. As our first major contribution, we extend the factored MDP representation (Boutilier, Hanks, and Dean 1999) to compactly represent MDP-IPs.

Compact Representation

Factored model In MDP-IPs, the transition probabilities are specified using linear constraints, meaning that the transition probability may be *any* distribution consistent with these constraints. We refer to the set of all legal distributions as a *credal set* (K). The challenge then is to specify such transition credal sets in a factored manner that is itself compact. For this purpose, we can use *dynamic credal networks (DCNs)* (Cozman 2000; 2005).

An example DCN is shown in Figure 1a. A DCN has the same two-layer structure as a dynamic Bayesian network (DBN) (Dean and Kanazawa 1990). For each variable X_i' in a DCN, we have a *conditional probability table (CPT)* with imprecise probabilities. If we examine the CPTs in Figure 1b, we note that entries are specified by parameters p_{ij} (i for variable X_i', j for the jth parameter in the CPT for X_i'). Furthermore, we note that we have side linear constraints on these p_{ij} shown in the box below the CPT. Thus, given $\vec{p} = (\ldots, p_{ij}, \ldots)$ consistent with the side linear constraints, we obtain a legal transition distribution where $pa_a(X_i')$ are X_i''s parents in the DCN for action $a \in A$:

$$P(\vec{x}'|\vec{x}, \vec{p}, a) = \prod_{i=1}^{n} P(x_i'|pa(X_i'), a, \vec{p}) \tag{9}$$

For the example in Figure 1 we have: $P(X_1' = 0, X_2' = 0|X_1 = 1, X_2 = 1, a_1, \vec{p}) = p_{12}p_{24}$. Note that we use x_i to represent the value assignment of the state variable X_i. Then we can define the *factored transition credal set* $K = \{P(\vec{x}'|\vec{x}, \vec{p}, a) |\ \vec{p}$ is consistent with side linear constraints $\}$.

Decision Diagrams With *algebraic decision diagrams (ADDs)* (Bahar et al. 1993), we can compactly represent functions $\{0,1\}^n \rightarrow \mathbb{R}$. ADDs are *directed acyclic graphs (DAGs)* whose variable tests on any path from root to leaf follow a fixed total variable ordering. ADDs often provide an efficient representation of functions with context-specific independence (Boutilier et al. 1996) and shared structure. For example, the reward function $R(x_1, x_2, x_3) = \sum_{i=1}^{3} x_i$ represented in Figure 2 as an ADD exploits the redundant structure of sub-diagrams through its DAG representation.

Unary operations such as $\min$, $\max$, marginalization over variables ($\sum_{x_i \in X_i}$), as well as binary operations such as addition ($\oplus$), subtraction ($\ominus$), multiplication ($\otimes$), $\min(\cdot,\cdot)$ and $\max(\cdot,\cdot)$ can be performed efficiently on ADDs. Space limitations prevent an algorithmic description of these operations; we refer the reader to (Bahar et al. 1993) for details.

In order to represent the conditional probabilities for an MDP-IP in a compact form, we propose a novel extension of ADDs called *parameterized ADDs (PADDs)* since the leaves are parameterized expressions as shown in Figure 1c. PADDs generalize the constant leaves of ADDs to polynomials expressed in the following sum-of-products canonical form where the d_i are constants and the p_{ij} are parameters:

$$d_0 + \sum_i d_i \prod_j p_{ij} \tag{10}$$

It is straightforward to extend the unary and binary operations for ADDs to also apply to PADDs — this only requires that (a) identical leaves are assigned the same unique identifier and that (b) operations on the leaves are modified to accept and produce resulting polynomials in the form of (10). We crucially note that the binary operations of $\oplus$, $\ominus$, and $\otimes$ are *closed* for PADDs with leaves in the form of (10), that is, these operations on PADDs yield new PADDs with leaves that can also be expressed as (10). Note that PADDs are not closed under binary division, $\min(\cdot,\cdot)$, or $\max(\cdot,\cdot)$; fortunately, such operations will not be needed in our proposed solution to factored MDP-IPs.

Factored MDP-IP Value Iteration

SPUDD-IP Description We extend the *SPUDD* (Hoey et al. 1999) algorithm for exploiting DBN and ADD structure in the solution of MDPs to a novel algorithm *SPUDD-IP* for exploiting DCN and PADD structure in the solution of MDP-IPs. We begin by expressing MDP-IP value iteration in the following factored form using operations on decision diagrams:

$$V^t(\vec{x}) = \max_{a \in A} \left\{ R(\vec{x}, a) \oplus \gamma \min_{\vec{p}} \boxed{\sum_{\vec{x}'} \bigotimes_{i=1}^{n} P(x_i'|pa_a(x_i'), a, \vec{p}) V^{t-1}(\vec{x}')} \right\} \tag{11}$$

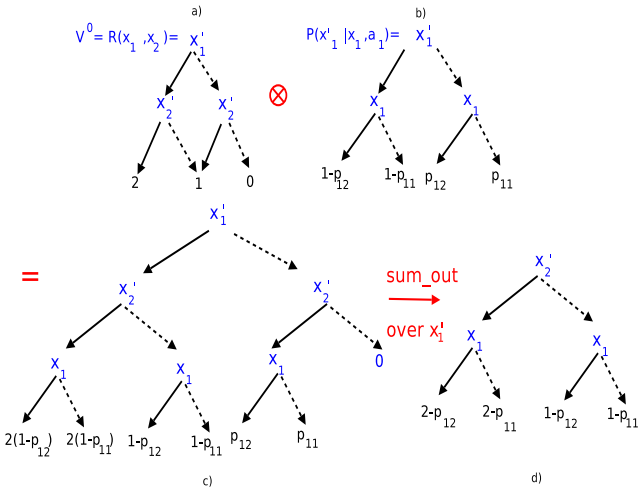

Figure 3: a) We show $V^0_{ADD} = R(x_1, x_2) = \sum_{i=1}^{2} x_i$ represented as an ADD. b) The PADD representation for $P(x'_1|x_1, a_1)$ ($CPT^{x'_1}_{a_1}$). c) The multiplication $V^0_{ADD} \otimes CPT^{x'_1}_{a_1}$ resulting in a PADD. d) The result of summing out over x'_1, which is a PADD.

Because the transition CPTs $P(x'_i|pa_a(x'_i), a, \vec{p})$ in the MDP-IP DCN contain parameters $\vec{p}$, these CPTs must be represented in decision diagram format as PADDs. On the other hand, the reward $R(\vec{x}, a)$ can be represented as an ADD since it contains only constants (for the purpose of operations, note that ADDs are special cases of PADDs). Although it may appear the form of $V^t(\vec{x})$ might be a PADD, we note that the parameters $\vec{p}$ are "minimized"-out w.r.t. the side constraints on $\vec{p}$ during the $\min_{\vec{p}} \square$ operation. This is crucial, because the $\max_{a \in A}$ can only be performed on ADDs (recall that $\max$ is not a closed operation on PADDs). Thus the resulting $V^t(\vec{x})$ computed from the $\max_{a \in A}$ has constant leaves and can be expressed as the ADD special case of PADDs.

To explain the efficient evaluation of (11) in more detail, we can exploit the variable elimination algorithm (Zhang and Poole 1994) in the marginalization over all next states $\sum_{\vec{x'}}$. For example, if x'_1 is not dependent on any other x'_i for $i \neq 1$, we can "push" the sum over x'_1 inwards to obtain:

$$V^t(\vec{x}) = \max_{a \in A} \left\{ R(\vec{x}, a) \oplus \gamma \min_{\vec{p}} \left[\sum_{x'_i (i \neq 1)} \bigotimes_{i=1 (i \neq 1)}^{n} P(x'_i|pa_a(X'_i), a, \vec{p}) \sum_{x'_1} P(x'_1|pa_a(X'_1), a, \vec{p}) V^{t-1}(\vec{x}') \right] \right\} \quad (12)$$

We show this graphically in an example in Figure 3. Here, we have the ADD representation for the first value function $V^0 = R$ (Figure 3a), which we multiply by $CPT^{x'_1}_a$ (Figure 3b) yielding the result (Figure 3c) and sum this out over x'_1 to obtain the final result (Figure 3d). Then we can continue with x'_2, multiplying this result by the $CPT^{x'_2}_a$, summing out over x'_2, and repeating for all x'_i to compute $\square$.

Representing the contents of $\square$ as $f(\vec{x}, a, \vec{p})$, we obtain

$$V^t(\vec{x}) = \max_{a \in A} \left\{ R(\vec{x}, a) \oplus \gamma \min_{\vec{p}} \boxed{f(\vec{x}, a, \vec{p})} \right\}. \quad (13)$$

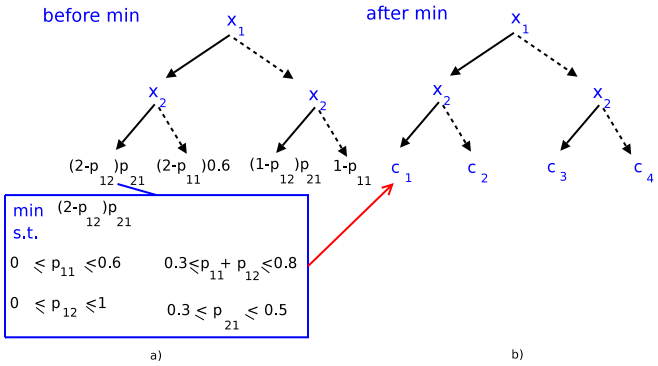

Figure 4: a) The PADD before minimization and a multilinear program for the first leaf, the solution for this leaf is the constant value c_1. b) The resulting ADD after the minimization at all leaves.

Note that $\min_{\vec{p}} f(\vec{x}, a, \vec{p})$ leads to a separate nonlinear expression minimization for every $\vec{x}$ and every a subject to the side linear constraints on $\vec{p}$. This optimization problem may be represented as a simple *multilinear program*[1] if p_{ij} only appears in the DCN CPT for X'_i (this prevents multiplication of p_{ij} by itself, thereby preventing non-multilinear terms).

To demonstrate how the $\min_{\vec{p}} \square$ is performed on PADDs, we refer to Figure 4. Here, each leaf expression in $\square$ (Figure 4a) given by the PADD corresponds to the function that Nature must minimize in each state. We crucially note that the PADD aggregates states with the same minimization objective, thus saving time-consuming calls to the multilinear solver. We will observe this time savings in our experiments.

Now, we need to make a call to the multilinear solver for each leaf, passing the leaf expression as the objective to minimize subject to the side constraints of our DCN that specify the legal $\vec{p}$ – we can then replace this leaf with a constant for the optimal objective value returned by the multilinear solver (Figure 4b). We can see that after the $\min_{\vec{p}}$ operation, all PADDs are simplified to the special case of ADDs with leaf nodes that are constants.

To complete one step of factored MDP-IP value iteration, we take the ADD resulting from the $\min$ operation, multiply it by the scalar γ, add in the reward $R(\vec{x}, a)$, and finally perform a sequence of binary *ADD* $\max(\cdot, \cdot)$ operations to compute the $\max_a$, thus yielding the ADD $V^t(\vec{x})$ from the ADD for $V^{t-1}(\vec{x})$ and completing one step of value iteration from (11).

SPUDD-IP Algorithm Factored MDP-IP value iteration is formally specified in the following two procedures:

`Solve` (Algorithm 1) constructs a series of t-stage-to-go value functions V^t_{DD} that are represented as ADDs. First we create the PADD representation of all DCN CPTs in the MDP-IP and initialize the first value function to 0. The loop is repeated until a maximum number of iterations or until a Bellman error BE termination condition ($BE < tol$) is met.

[1]To qualify as multilinear, nonlinear terms are allowed if the exponent of each parameter in a term is 1. For example, $p_1 p_2 p_3$ is multilinear, but $p_1^2 p_2 p_3$ is not.

Algorithm 1: `Solve`(MDP-IP, tol, $maxIter$, δ, APP, Obj)

begin
- Create PADD CPTs $CPT_a^{x_i'}$ for MDP-IP;
- $V_{ADD}^0 = 0$;
- $Vmax = \max(R_{DD})$;
- $t = 0$;
- **while** $i < maxIter$ **do**
  - $t = t + 1$;
  - $V_{DD}^t = -\infty$;
  - **foreach** $a \in A$ **do**
    - Q_{DD}^t=`Regress(`$V_{DD}^{t-1}, a, \delta \cdot Vmax, Obj$`)`;
    - V_{DD}^t=max`(`V_{DD}^t,Q_{DD}^t`)`;
  - $Diff_{DD} = V_{DD}^t \ominus V_{DD}^{t-1}$;
  - $BE = \max(\max(Diff_{DD}), -\min(Diff_{DD}))$;
  - **if** $BE < tol$ **then**
    - break;
  - **if** APP *pruning* **then**
    - V_{DD}^t =`ApproxADD` $(V_{DD}^t, \delta \cdot Vmax)$;
  - $Vmax = \max(R_{DD}) + \gamma Vmax$;
- **return** V_{DD}^t ;

end

Algorithm 2: `Regress`($V_{DD}, a, error, Obj$)

begin
- Q_{DD} = `convertToPrimes(`V_{DD}`)`;
- **for** *all* X_i' *in* Q_{DD} **do**
  - $Q_{DD} = Q_{DD} \otimes CPT_a^{x_i'}$;
  - $Q_{DD} = \sum_{x_i' \in X_i'} Q_{DD}$;
- **if** Obj *pruning* **then**
  - Q_{DD} =`approxPADDLeaves` $(Q_{DD}, error)$;
- Q_{DD} = `callSolverForEachLeafIn` (Q_{DD});
- **return** $R_{DD} \oplus (\gamma \otimes Q_{DD})$;

end

Algorithm 3: `ApproxADD`($value_{DD}^i$,$error$)

begin
- $leaves_{old}$=`collectLeavesADD` $(value_{DD}^i)$;
- $\{leaves_{old} \rightarrow leaves_{new}\}$ =`mergeLeaves` $(leaves_{old}, error)$;
- **return** `createNewDD` $(value_{DD}^i, \{leaves_{old} \rightarrow leaves_{new}\})$;

end

We note that setting the tolerance tol according to (6) guarantees ϵ-optimality for MDP-IPs since the same termination conditions used for MDPs directly generalize to MDP-IPs in the discounted case ($\gamma < 1$). At each iteration the `Regress` algorithm is called and V_{DD}^t is updated with the max over all Q_{DD}^t for each action a computed by regress(V_{DD}^{t-1}, a). After this, $BE = max_{\vec{x}} |V^t(\vec{x}) - V^{t-1}(\vec{x})|$ is computed and tested for termination. δ, APP, Obj, and $Vmax$ play no role now; they are used for approximation in the next section.

`Regress` (Algorithm 2) computes Q_{DD}^t, i.e, it regresses V_{DD}^{t-1} through action a that provides the values Q_{DD}^t that could be obtained if executing a and acting so as to obtain V_{DD}^{t-1} thereafter. During regression we "prime" the variables by converting each X_i to X_i' (since the V_{DD}^i is now part of the "next" state) and the CPTs for action a are multiplied in and summed out.[2] After this, the multilinear solver is called to find the minimizing $\vec{p}$ for each leaf in the PADD w.r.t. the side linear constraints on the DCN, resulting in an ADD. We note that *if* a leaf is already a constant, *then* the multilinear solver call can be avoided altogether; this observation will prove important later when we introduce objective pruning. Finally, the future value is discounted and the reward ADD is added in to complete the regression. Obj and $error$ are used for approximate value iteration and will be discussed later.

Factored MDP-IP Approximate Value Iteration

The previous SPUDD-IP exact value iteration solution to factored MDP-IPs often yields an improvement over flat value iteration as we will demonstrate in our experiments. But as the number of state variables in a problem grows larger, it often becomes impossible to obtain an exact solution due to time and space limitations.

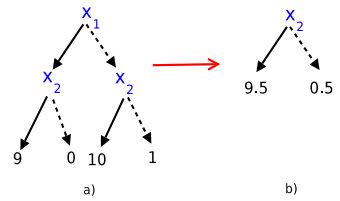

Figure 5: a) The value function V_{DD}^t represented as an ADD. b) the result of `ApproxADD` applied to V_{DD}^t with approximation $error = 1$; note that the leaves within $error$ of each other have been merged and averaged and the resulting ADD simplified.

Approximate value iteration (AVI) is one way to trade off time and space with error by approximating the value function after each iteration. In this section, we propose two (bounded) AVI extensions of SPUDD-IP. Each method uses a different way to approximate the value, but both methods incur a maximum of $\delta \cdot Vmax$ error per iteration where $Vmax$ as computed in `Solve` represents the maximum possible value at each step of value iteration. By making the approximation error sensitive to $Vmax$ we prevent overaggressive value approximation in the initial stages of AVI when values are relatively small as suggested in (St-Aubin, Hoey, and Boutilier 2000). Even with this value approximation at every iteration, satisfying the termination condition $BE < tol$ for some tol still yields strict guarantees on the overall approximation error given by (6) as discussed previously for SPUDD-IP.

[2] We assume there are no synchronic arcs among variables X_i', X_j' for $i \neq j$ in the DCN. If sychronic arcs are present, the algorithm can be modified to multiply in all relevant CPTs.

APRICODD-IP Algorithm The APRICODD algorithm (St-Aubin, Hoey, and Boutilier 2000) provides an efficient way of *approximating the ADD value representation* for a factored MDP, reducing its size and thus computation time per iteration. This approach immediately generalizes to MDP-IPs since V^t_{DD} is also an ADD. To execute *APRICODD-IP* AVI for MDP-IPs, we simply call `Solve` (Algorithm 1) with $APP = true$ and set δ ($0 < \delta \leq 1$) to some fraction of the maximum possible value $Vmax$ with which to approximate using `ApproxADD`:

`ApproxADD` (Algorithm 3) collects all leaves of the ADD and determines which can be merged to form new values without approximating more than $error$. The old values are then replaced with these new values creating a new (minimally reduced) ADD that represents the approximated value function. An illustrative example is shown in Figure 5.

Objective-IP Algorithm APRICODD is an effective extension of SPUDD for factored MDPs (not MDP-IPs) because it reduces the size of the value function ADDs, which largely dictate the time complexity of the algorithm. However, in solving (factored) MDP-IPs, the time is dictated less by the size of the value ADD and more by the number of calls to the nonlinear solver to compute the $\min_{\vec{p}}$. SPUDD-IP started to attack this source of time complexity by aggregating states with the same objective for the $\min_{\vec{p}}$. Our goal with the *Objective-IP* pruning algorithm will be to more closely target the source of time complexity in an AVI version of SPUDD-IP by *approximating the nonlinear objectives*. We do this by moving the approximation step into the `Regress` algorithm where the nonlinear solver is called. Noting that each PADD leaf in `Regress` is a nonlinear objective, we simplify it by calling `ApproxPADDLeaves`:

`ApproxPADDLeaves` (Algorithm 4) is called for a PADD (by `Regress` if $Obj = true$) just prior to carrying out the nonlinear optimization at the leaves of that PADD. The main loop of the algorithm attempts to approximate each leaf in PADD. Approximation is done in the inner loop by pruning out multilinear terms $d_i \prod_j p_{ij}$ from the leaf expression and replacing them with the average of their maximum and minimum values. This requires knowing the absolute upper p^U_{ij} and lower bounds p^L_{ij} for any p_{ij}, which can be easily precomputed once for the entire MDP-IP by calling the nonlinear solver to compute $p^U_{ij} = \max p_{ij}$ and $p^L_{ij} = \min p_{ij}$ subject to the side linear constraints on all CPTs. In some cases the complexity of the leaf expression may be reduced, in others, it may actually be reduced to a constant. Note that the leaves are each approximated independently, this can be done since each leaf corresponds to a different state (or set of states) and the system can only be in one state at a time. We guarantee that no objective pruning at the leaves of the PADD incurs more than $error$:

Theorem (*`ApproxPADDLeaves` Error Bound*)**.** Given an MDP-IP, its precomputed constants p^L_{ij} and p^U_{ij} for all p_{ij}, and the maximum approximation $error$, then whenever `ApproxPADDLeaves` (Algorithm 4) reduces a leaf $d_0 + \sum_{i=1}^{\#terms} d_i \prod_j p_{ij}$ to a simpler expression, the approximation error in the objective minimization ($\min_{\vec{p}}$) of that leaf is bounded by $error$.

Algorithm 4: `ApproxPADDLeaves`($DD, error$)

begin
  foreach $leaf : d_0 + \sum_{i=1}^{\#terms} d_i \prod_j p_{ij} \in DD$ **do**
    $i = 1$, $curError = 0$;
    while $curError < error \wedge i \leq \#terms$ **do**
      $newValue = \frac{d_i}{2}\left(\prod_j p^U_{ij} + \prod_j p^L_{ij}\right)$;
      $termError_i = |\frac{d_i}{2}\left(\prod_j p^U_{ij} - \prod_j p^L_{ij}\right)|$;
      if $curError + termError_i < error$ **then**
        remove term $d_i \prod_j P_{ij}$ from $leaf$;
        $d_0 = d_0 + newValue$;
        $curError = curError + termError_i$;
      $i = i + 1$;
  return DD;
end

Proof. We begin by showing that the approximation error induced by removing a single term i from the objective is bounded by $termError_i$. To do this, we first find upper and lower bounds on term i ($d_i \Pi_j p_{ij}$) based on the legal values of $\vec{p}$. We know the maximal (minimal) possible value for each p_{ij} is p^U_{ij} (p^L_{ij}). Thus for any possible legal values of $\vec{p}$ the term i must be bounded in the interval $[L_i, U_i]$ with L_i and U_i defined as follows:

$$L_i = \begin{cases} d_i > 0 & d_i \Pi_j p^L_{ij} \\ d_i < 0 & d_i \Pi_j p^U_{ij} \end{cases}, \quad U_i = \begin{cases} d_i > 0 & d_i \Pi_j p^U_{ij} \\ d_i < 0 & d_i \Pi_j p^L_{ij} \end{cases}$$

Let g be a value for term i and $\widehat{g} = \frac{L_i + U_i}{2}$, then $max_g |g - \widehat{g}|$ occurs at $g = L_i$ or $g = U_i$. So the max $termError_i = max\left(|L_i - \widehat{g}|, |U_i - \widehat{g}|\right) = \max\left(|\frac{L_i - U_i}{2}|, |\frac{U_i - L_i}{2}|\right) = |\frac{L_i - U_i}{2}|$ as computed in Algorithm 4.

Now, let $OBJ1 = d_0 + \sum_{i=1}^{\#terms} d_i \Pi_j p_{ij}$ be the original *non-approximated objective expression* to minimize and v_1 the optimal objective value using $\vec{p} = \vec{p_1}$. Let $OBJ2 = d_0 + \widehat{g} + \sum_{i=2}^{\#terms} d_i \Pi_j p_{ij}$ be the *approximated objective expression* to minimize after replacing term 1 with $\frac{L_1 + U_1}{2}$ and v_2 the optimal objective using $\vec{p} = \vec{p_2}$.

We want to prove that $-\left|\frac{L_1 - U_1}{2}\right| < v1 - v2 < \left|\frac{L_1 - U_1}{2}\right|$. First we prove the second part of this inequality. Using $\vec{p_2}$ in $OBJ1$ we obtain $v'_1 = d_0 + eval(d_1 \Pi_j p_{1j}, \vec{p_2}) + v_2 - d_0 - \widehat{g}$ (where $eval$ is a function to evaluate the term with the assigned values). Because v_1 is optimal $v_1 \leq v'_1$ then:

$$v_1 - v_2 \leq eval(d_1 \Pi_j p_{1j}, \vec{p_2}) - \widehat{g} \tag{14}$$

Additionally for any possible legal values of $\vec{p}$ and for $\vec{p_2}$, $|eval(d_1 \Pi_j p_{1j}, \vec{p_2}) - \widehat{g}| < |\frac{L_1 - U_1}{2}|$, i.e., $-\left|\frac{L_1 - U_1}{2}\right| < eval(d_1 \Pi_j p_{1j}, \vec{p_2}) - \widehat{g} < \left|\frac{L_1 - U_1}{2}\right|$. From this equation and (14) we obtain $v1 - v2 < \left|\frac{L_1 - U_1}{2}\right|$. The proof of the first inequality follows by the same reasoning, but this time substituting $\vec{p_1}$ into $OBJ2$.

This bounds the objective approximation error for one term approximation and by simple induction, we can additively bound the accumulated error for multiple approximations as calculated using $curError$ in Algorithm 4. □

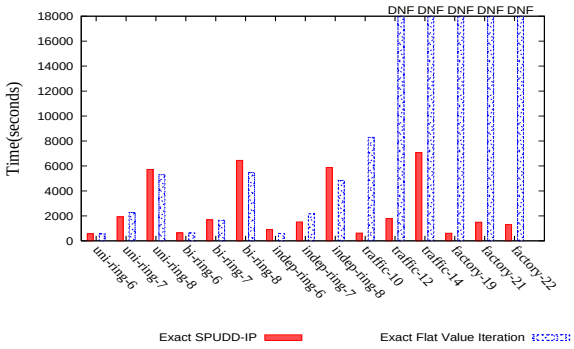

Figure 6: Time performance comparison for TRAFFIC, SYSADMIN and FACTORY problems using SPUDD-IP and Flat Value Iteration. The name includes the number of variables in each problem, so the corresponding number of states is $2^{\#variables}$.

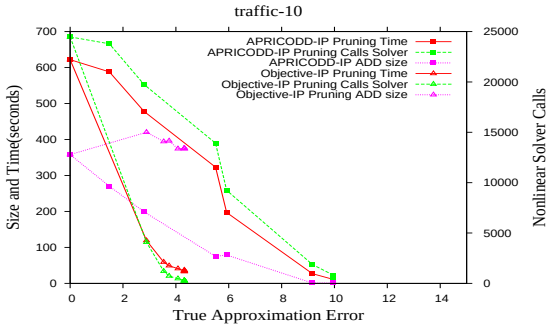

Figure 7: Detailed comparison between APRICODD-IP pruning and Objective-IP pruning for the traffic problem with 10 variables. True Approximation Error is $max_{\vec{x}}|V^*(\vec{x}) - V_{approx}(\vec{x})|$.

Experimental Results

In this section, we empirically evaluate four algorithms: *Flat Value Iteration* from (8) and our three previously defined contributions: *SPUDD-IP*, *APRICODD-IP*, and *Objective-IP*. We evaluate these algorithms on factored MDP-IP versions of three domains: (1) FACTORY (St-Aubin, Hoey, and Boutilier 2000) where the success (failure) probability p_1 (p_2) of the *bolt* action is constrained by $0.2 + p_2 \leq p_1 \leq 1$ and $0.5 \leq p_2 \leq 1$. (2) SYSADMIN (Guestrin et al. 2003) (three topologies: (a) unidirectional ring, (b) bidirectional ring, (c) independent bidirectional rings of pairs of computers) where the probability p_i (p_i') that each computer i fails (reboots) is governed by the constraints: $0.85 + p'_i \leq p_i \leq 0.95$ and $0 \leq p'_i \leq 0.1$. (3) We introduce TRAFFIC, a factored MDP-IP domain motivated by a real traffic intersection control problem. Space limitations preclude a detailed description, but we note that the objective is to minimize the count of occupied road cells around an intersection. Estimating turn probabilities is difficult and so we model them with a non-stationary distribution where for two lanes of traffic with turn probabilities p_1 and p_2, we have the constraint $p_1 \geq p_2$ (cars turn more often from one lane than the other). For all algorithms, we set $maxIter = 50$ for SYSADMIN and $maxIter = 75$ for the other domains and used $\gamma = 0.9$.

Flat Value Iteration vs. SPUDD-IP In Figure 6 we compare the running time of the two exact solution methods: SPUDD-IP and Flat Value Iteration. Solutions not completing in five hours are marked *Did Not Finish (DNF)*. We note that SPUDD-IP did not outperform Flat Value Iteration on the SYSADMIN domains because the exact value function has little structure as an ADD. However, both TRAFFIC and FACTORY had highly structured value functions and *up to two orders magnitude time improvement* is demonstrated by SPUDD-IP, largely due to the ability of the PADDs to aggregate common nonlinear objectives, thus saving a substantial number of calls to the nonlinear solver and therefore time.

APRICODD-IP vs. Objective-IP For approximate value iterations comparisons of each algorithm, we ran `Solve` for a range of δ; in the plots we refer to the actual approximation error resulting from this δ, which is $\max_{\vec{x}} |V_{DD}^{maxIter}(\vec{x}) - V_{DD}^{*}(\vec{x})|$ with V_{DD}^{*} computed using $\delta = 0$, i.e., SPUDD-IP.

In Figure 7 we present a detailed comparison of the time, space, and number of nonlinear solver calls required by APRICODD-IP and Objective-IP plotted vs. the true approximation error for *traffic-10*. We note little relationship between the space required by the ADD value representation and the running times of the algorithms (space actually increases slightly for Objective-IP while running time decreases). But what is striking about this plot is that the running time of each algorithm is directly correlated with the number of nonlinear solver calls made by the algorithm (taking up to 100ms in some cases), reflecting our intuitions that the time complexity of solving MDP-IPs is governed by the computational overhead of nonlinear optimization.

In Figure 8, we show a comparison of true approximation error vs. running times for three problems and three different sizes of each problem. The results here echo one conclusion: Objective-IP consistently takes less time than APRICODD-IP to achieve the same approximation error and *over one order of magnitude less time* in the case of FACTORY.

Related Work

The *Bounded-parameter Markov Decision Process (BMDP)* (Givan, Leach, and Dean 2000) is a special case of an MDP-IP, where the probabilities and rewards are specified by constant intervals. Recent solutions to BMDPs include robust versions of real-time dynamic programming (Buffet and Aberdeen 2005). The *Markov Decision Process with Set-valued Transitions (MDPSTs)* (Trevizan, Cozman, and de Barros 2007) is another subclass of MDP-IPs where probability distributions are given over finite sets of states. The algorithms defined in this paper clearly apply to both BMDPs and MDPSTs, however *all* of the above techniques do not generalize to the MDP-IPs we examined in this paper that use linear constraints on a DCN probability specification. This representation was used in all of our MDP-IP domains.

Concluding Remarks

Motivated by the real-world need to solve MDPs with transition uncertainty, we proposed the compact factored MDP-IP model and developed a decision diagram-based factored value iteration extension of the SPUDD algorithm. We contributed the parameterized ADD (PADD) data structure and

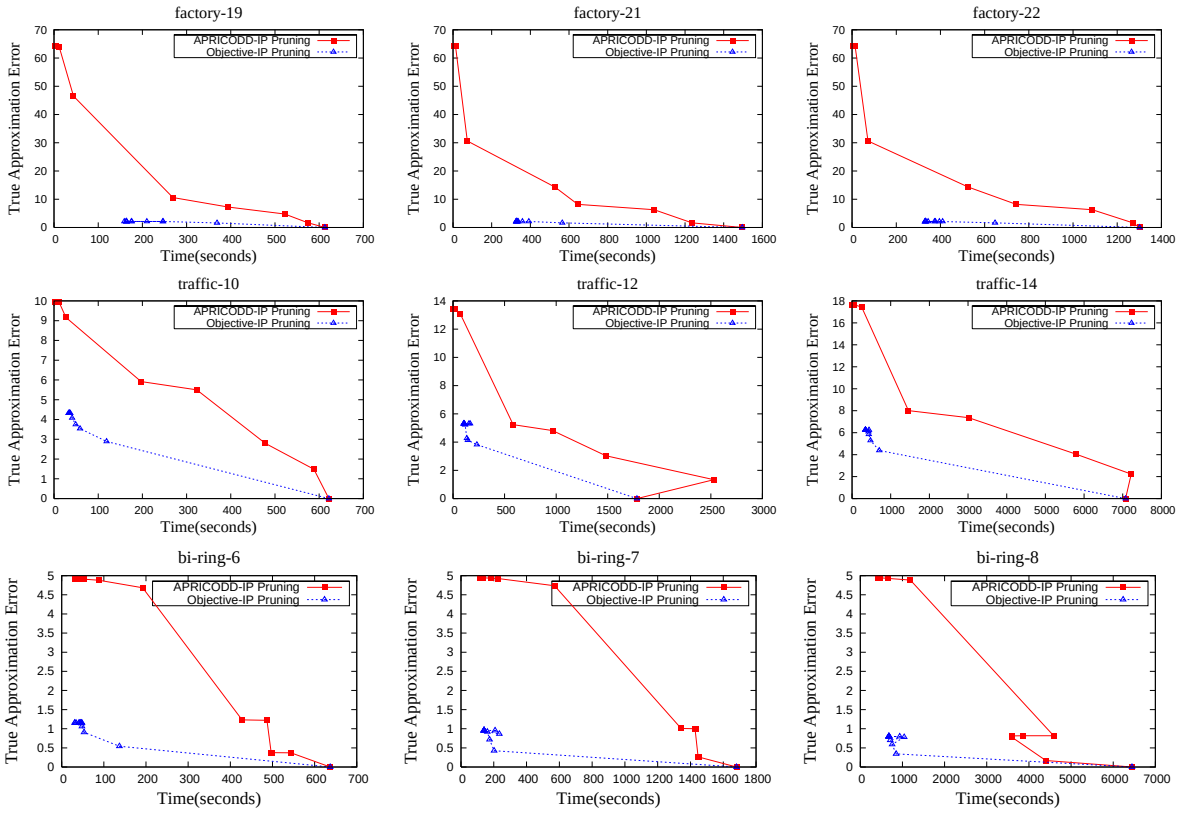

Figure 8: True Approximation Error ($max_{\vec{x}}|V^*(\vec{x}) - V_{approx}(\vec{x})|$) vs. time required for APRICODD-IP and Objective-IP.

operations and used PADDs in novel inference algorithms with compact factored MDP-IP transition credal sets represented as Dynamic Credal Networks (DCNs). The resulting SPUDD-IP algorithm yielded up to two orders of magnitude speedup over pre-existing value iteration techniques.

Seeking to scale SPUDD-IP further, we contributed two novel approximate value iteration extensions: APRICODD-IP and Objective-IP. While APRICODD-IP is the obvious extension based on previous work, it did not specifically target the main source of time complexity for solving MDP-IPs — calls to the nonlinear solver during MDP-IP value iteration. Based on this observation, we developed an alternate and novel approximation method that directly approximated the objective of nonlinear solver calls, proving the theoretical correctness of this approach and substantially reducing the number of nonlinear solver calls and thus running time of approximate value iteration. Altogether these novel extensions enable the modeling and (bounded approximate) solution of factored MDP-IPs that can scale orders of magnitude beyond existing flat value iteration approaches to MDP-IPs.

References

Bahar, R. I.; Frohm, E. A.; Gaona, C. M.; Hachtel, G. D.; Macii, E.; Pardo, A.; and Somenzi, F. 1993. Algebraic decision diagrams and their applications. In *ICCAD-93*, 188–191.

Boutilier, C.; Friedman, N.; Goldszmidt, M.; and Koller, D. 1996. Context-specific independence in Bayes nets. In *UAI-96*,115–123.

Boutilier, C.; Hanks, S.; and Dean, T. 1999. Decision-theoretic planning: Structural assumptions and computational leverage. *JAIR* 11:1–94.

Buffet, O., and Aberdeen, D. 2005. Robust planning with LRTDP. In *IJCAI-05*, 1214–1219.

Cozman, F. G. 2000. Credal networks. *Artif. Intell.* 120:199–233.

Cozman, F. G. 2005. Graphical models for imprecise probabilities. *Internat. Journal of Approx. Reasoning* 39(2-3):167–184.

Dean, T., and Kanazawa, K. 1990. A model for reasoning about persistence and causation. *Comput. Intell.* 5(3):142–150.

Givan, R.; Leach, S.; and Dean, T. 2000. Bounded-parameter Markov decision processes. *Artif. Intell.* 122:71–109.

Guestrin, C.; Koller, D.; Parr, R.; and Venkataraman, S. 2003. Efficient solution algorithms for factored MDPs. *JAIR* 19:399–468.

Hoey, J.; St-Aubin, R.; Hu, A.; and Boutilier, C. 1999. Stochastic planning using decision diagrams. In *UAI-99*, 279–288.

Puterman, M. L. 1994. *Markov Decision Processes*. New York: John Wiley and Sons.

Satia, J. K., and Lave Jr., R. E. 1970. MDPs with uncertain transition probabilities. *Operations Research* 21:728–740.

St-aubin, R.; Hoey, J.; and Boutilier, C. 2000. Approximate policy construction using decision diagrams. In *NIPS-00*.

Trevizan, F. W.; Cozman, F. G.; and de Barros, L. N. 2007. Planning under risk and Knightian uncertainty. In *IJCAI-07*.

White III, C. C., and El-Deib, H. K. 1994. MDPs with Imprecise Transition Probabilities. *Operations Research* 42(4):739–749.

Zhang, N. L., and Poole, D. 1994. A simple approach to Bayesian network computations. In *CCAI-94*, 171–178.