Solution Techniques for First-order MDPs with Sum/Product Aggregators and Factored Actions

Scott Sanner and Craig Boutilier
Department of Computer Science
University of Toronto
[ssanner,cebly]@cs.toronto.edu

January 10, 2007

Abstract

This document extends the symbolic dynamic programming framework for first-order MDPs (FOMDPs) to handle sum/product aggregators and factored action effects. It also introduces first-order approximate linear programming (FOALP) and first-order approximate policy iteration (FOAPI) algorithms for efficient value function approximation in the presence of sum/product aggregators. This document assumes familiarity with the framework and terminology used in factored MDPs [1], FOMDPS [2], and previous work on the FOALP/FOAPI solution of FOMDPs [3, 4].

The fundamental result of this work is that it permits the lifted solution of first-order MDPs with reward and transition functions containing indefinite additive and multiplicative structure. This accounts for most problems specified by planning languages such as STRIPS [5] and PPDDL [6] as well as lifted versions of most discrete factored MDPs currently studied in the literature. This is the *first* fully lifted method for tackling such stochastic planning problems and permits the solution of these problems in a manner that *does not scale with the number of domain objects.*

Contents

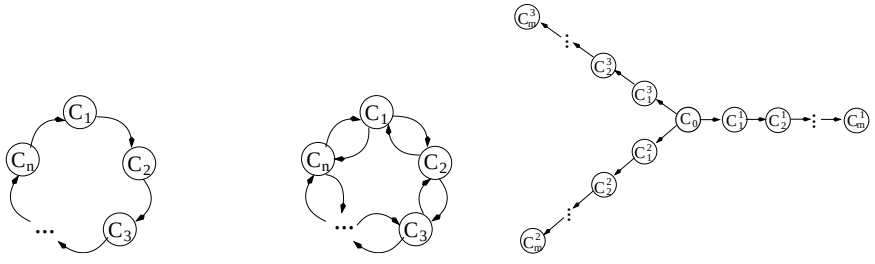
1 Introduction

The formalism and algorithms in this document were motivated by an attempt to formalize a fully first-order MDP (FOMDP) approach to solving the SysAdmin problem [7]. Previous solution approaches to this problem [7, 8] specified it as a propositionally factored MDP [1]. These approaches all have the drawback that their *space and time complexity scale exponentially with the size of the domain*, thus placing an inherent limit on the domain sizes that can be practically solved. Yet, the SysAdmin domain (and many like it) is highly structured, involving a great deal of redundancy and independence that can be exploited using a first-order representation and appropriately adapted solution techniques. Thus, our goal in this work is to extend the FOMDP representation and solution algorithms to a broad class of problems including SysAdmin in a manner that does not scale with domain size.

2 SysAdmin Domain

While the techniques that we introduce in this paper apply to a general class of problems, we begin by introducing the SysAdmin problem since it is the motivation for this work and is used as a running example throughout this paper. In the SysAdmin problem, we have n computers connected via some directed graph topology. At every time step, we can reboot any *single* computer. Whether a computer is running in each time step depends on whether it was rebooted (in which case it must be running), otherwise the probability that it is running depends on the status of the computer in the previous time step as well as that of the computers with incoming connections. The reward in this problem is the sum of computers that are running at any time step. Consequently, an optimal policy in this problem will reboot the computer that has the most impact on the expected future discounted reward given the network configuration.

There are three commonly referenced connection topologies for the SysAdmin problem shown in Figure 1 that we focus on in this document. It is impor-



(a) Unidirectional Ring (b) Bidirectional Ring (c) Star Configuration

Figure 1: Diagrams of the three different SysAdmin topologies that we will focus on in this document.

tant to have a first-order axiomatization of these SysAdmin topologies since it will allow us to *prove* properties of actions at a first-order level that we will exploit in our FOMDP solutions.

The following axiomatizations of the three SysAdmin connection topology specifications all take the same form:

1. A notation for uniquely specifying all of the computers in the network.
2. Topological constraints on the network connections between computers.

In axiomatizing these domains, it is important to note that we are not assuming a fixed number of computers, rather we are specifying generic constraints on various network topologies that hold for networks of any size.

In the following axiomatization, we use a sorted logic where variables i, j, k and constants m and n are of the Natural number sort, and all function symbols beginning with c or d are of the computer sort C. We also assume that our knowledge base of constraints includes the Peano axioms required to formalize the comparison predicates $>$,$<$,$\geq$,$\leq$ and the arithmetic function $+$.

2.1 Unidirectional Ring Network

For the unidirectional ring network in Figure 1(a), we have a set of computers $C = \{c_1, \ldots, c_n\}$ where we can we think of the indexed notation for each computer c_i as a shorthand for the functional form $c(i)$. Then we have the following unique names axiom:

$$\forall i, j.\ i \neq j \ \supset\ c_i \neq c_j \tag{1}$$

Now we can axiomatize the structure of links using the $Conn(x, y)$ predicate (where there is a *directed* link from x towards y) in this unidirectional ring:

$$Conn(c_n, c_1) \tag{2}$$
$$\forall i, j.\ \ i \geq 1 \wedge j = i + 1 \wedge j \leq n \ \supset\ Conn(c_i, c_j) \tag{3}$$

2.2 Bidirectional Ring Network

The bidirectional ring network in Figure 1(b) has the same set of computers C and unique names axiom as the unidirectional ring, except that it's topology constraints are given by the following axioms:

$$Conn(c_n, c_1) \wedge Conn(c_1, c_n) \tag{4}$$
$$\forall i, j.\ \ i \geq 1 \wedge j = i + 1 \wedge j \leq n \ \supset\ Conn(c_i, c_j) \wedge Conn(c_j, c_i) \tag{5}$$

2.3 Star Network

For the star network in Figure 1(c) there are $n = 3m + 1$ computers. We label each of these computers as follows: The central computer gets the label c_0.

There are 3 branches in the network that we label $j = 1 \ldots 3$.[1] In each branch j there are m computers $i = 1 \ldots m$ that we label as c_i^j with smaller i being closer to the central computer c_0.

We consider c_0 to be an abbreviation for the function $c(0,0)$ and c_i^j to be an abbreviation for the function $c(i,j)$. Thus we can obtain the following unique names axiom:

$$\forall i, j, k, l.\ i \neq k \vee j \neq l \ \supset\ c_i^j \neq c_k^l \tag{6}$$

Finally, we can axiomatize the structure of links in the star network:

$$Conn(c_0, c_1^1) \tag{7}$$
$$Conn(c_0, c_1^2) \tag{8}$$
$$Conn(c_0, c_1^3) \tag{9}$$
$$\forall i, k, j.\ \ i \geq 1 \wedge k = i + 1 \wedge k \leq m \wedge j \geq 1 \wedge j \leq 3 \ \supset\ Conn(c_i^j, c_k^j) \tag{10}$$

3 Sum and Product Aggregators

There is no way to specify an indefinite sum in the current language of first-order MDPs.[2] Such a modeling construct is required in SysAdmin to express that the reward scales linearly with the number of running computers (c.f., Sec 4.1) and that the transition probability specifying that a non-rebooted computer is running scales linearly with the number of computers with incoming connections that are running (c.f., Sec 4.3). Likewise there is no way to specify indefinitely large products that are needed to formalize factored transition probability distributions (c.f., Sec 4.3).

Consequently, we introduce sum and product aggregators to allow the specification of indefinite sums and products in FOMDPs. The sum/product aggregators are defined in terms of the $\oplus/\otimes$ operators [2] as follows:

$$\sum_{c \in C} case(c, s) = \bigoplus_{c \in C} case(c, s) \tag{11}$$
$$\prod_{c \in C} case(c, s) = \bigotimes_{c \in C} case(c, s) \tag{12}$$

For notational simplicity, we often omit the sort C when it can be inferred from context.

Although we will assume that $|C| = n$ throughout his paper, we do not know the exact value of the constant n, only that it is a Natural number. Thus, we should informally treat $\sum_c/\prod_c$ just as we would $\forall c$ or $\exists c$ – it expresses a

[1] We have chosen 3 branches as an example, but making the number of branches a variable will not prevent the application of the algorithms presented in this document; it will simply make their solutions more complex.

[2] Although finite sums can be expressed in FOL using existential quantifiers and pairwise inequalities, this representation is both inefficient – growing quadratically as a function of the count – and it can not be extended to indefinitely large sums of an *apriori* unknown size.

sum/product over an indefinite number of objects and thus cannot be expanded to an equivalent finite representation until a domain closure assumption is made for c's particular sort. However, when solving FOMDPs specified in terms of the sum and product aggregators, we wish to avoid making such a domain closure assumption in order to achieve a fully lifted first-order solution.

3.1 Operations and Properties of the Sum Aggregator

Since the sum aggregator is defined in terms of the $\oplus$ case operator, the standard properties of commutativity, associativity, and distributivity of $\otimes$ over $\oplus$ hold. Likewise, since the product aggregator is defined in terms of the $\otimes$ case operator, the standard properties of commutativity and associativity hold. In this paper, we will make use of the following equivalences that can be derived as corollaries of these properties:

$$\sum_{c \in C} case_1(c, s) \oplus \sum_{d \in C} case_2(d, s) \equiv \sum_{c \in C} [case_1(c, s) \oplus case_2(c, s)] \quad (13)$$

$$case_1(s) \otimes \sum_{c \in C} case_2(c, s) \equiv \sum_{c \in C} [case_1(s) \otimes case_2(c, s)] \quad (14)$$

$$[case_1(s) \oplus case_2(s)] \otimes \prod_{c \in C} case_3(c, s) \equiv case_1(s) \otimes \prod_{c \in C} case_3(c, s) \oplus case_2(s) \otimes \prod_{d \in C} case_3(d, s) \quad (15)$$

Additionally, since we know that $Regr(\cdot)$ distributes through the $\oplus/\otimes$ case operators, we can easily infer the following:

$$Regr\Big(\sum_{c} case(c, do(a, s))\Big) \equiv \sum_{c} Regr\Big(case(c, do(a, s))\Big) \quad (16)$$

$$Regr\Big(\prod_{c} case(c, do(a, s))\Big) \equiv \prod_{c} Regr\Big(case(c, do(a, s))\Big) \quad (17)$$

4 First-order MDPs with Sum/Product Aggregators and Factored Actions

To represent the SysAdmin problem as a FOMDP, we need only provide a first-order representation of the reward and the transition dynamics for *any* domain instantiation.

4.1 Reward

The reward can be represented in the following manner:

$$rCase(s) = \sum_{c} \left(\begin{array}{|c|} \hline Running(c, s) \ : \ 1 \\ \hline \neg Running(c, s) : \ 0 \\ \hline \end{array} \right) \quad (18)$$

This states that we receive one unit of reward for every computer c that is running. If the cardinality of c's sort is n, we could apply the $\oplus$ operator and obtain a resulting case statement with 2^n case partitions and after some simplification, $n+1$ partitions of distinct value (i.e., $n+1$ distinct counts of computers running).

4.2 Factored Actions

Before we specify the transition dynamics for SysAdmin, we must digress for a moment to discuss the factored actions that will be crucial to the compact specification of these dynamics.

In the SysAdmin problem, if the user chooses to reboot a specific computer, say c_2, it is useful to view this as implicitly choosing to perform a *noop* for each of the remaining computers. Thus, one can view this *joint stochastic action* as a set of *stochastic sub-actions*:

$$\{noop(c_1), reboot(c_2), noop(c_3), \ldots, noop(c_n)\} \tag{19}$$

Since we will need to use the joint stochastic action in a logical framework, we use an associative-commutative composition function $\circ$ to express this set of sub-actions as a single first-order term representing the joint action:

$$noop(c_1) \circ reboot(c_2) \circ noop(c_3) \circ \cdots \circ noop(c_n) \tag{20}$$

In the following exposition, we consider the set and functional representations to be interchangeable.

In the FOMDP framework, we must now specify how these stochastic actions decompose according to Nature's choice probability distribution into deterministic actions so that we can use them within the deterministic situation calculus. As done for stochastic actions, we treat a *joint deterministic action* as a composition of *deterministic sub-actions*. This decomposition is expressed in terms of sub-actions: for each stochastic sub-action, $reboot(c_i)$ or $noop(c_i)$, Nature chooses a corresponding deterministic sub-action, $runningS(c_i)$ or $runningF(c_i)$ (S for success, F for failure), according to a pre-specified probability distribution.

Because the specification of this distribution is somewhat involved for the SysAdmin problem, we defer it to the next section, In it's place for now, we will simply use the general notation $P(A(c_i)|U(c_i))$. Here $A(c_i) = \{runningS(c_i), runningF(c_i)\}$ is a set of deterministic sub-action outcomes for a specific c_i and $A = \{A(c_1) \times \cdots \times A(C_n)\}$ will be used to represent the full set of joint deterministic actions. Likewise, $U(c_i) = \{reboot(c_i), noop(c_i)\}$ is the set of user's stochastic sub-actions for a specific c_i and $U = \{U(c_1) \times \cdots \times U(C_n)\}$ will be used to represent the full set of user's joint stochastic actions. We commonly refer to $A(c_i)$ as the set of potential deterministic sub-action outcomes for $U(c_i)$.

Given Nature's distribution $P(A(c_i)|U(c_i))$ over deterministic sub-actions conditioned on their corresponding stochastic sub-actions, we can easily express

the distribution over the joint deterministic actions in a factored manner:

$$P(A|U) = P(A(c_1) \circ \cdots \circ A(c_n)|U(c_1) \circ \cdots \circ U(c_n)) \tag{21}$$

$$= \prod_{c_i} P(A(c_i)|U(c_i)) \tag{22}$$

It is straightforward to see that this defines a proper probability distribution over the decomposition of joint stochastic actions into joint deterministic actions.

In dealing with joint deterministic actions within a situation calculus framework, we need some additional logical machinery to test when a joint deterministic action contains a specific sub-action (e.g., when checking for a particular effect that a joint action may have on a fluent). For this purpose, we introduce a predicate $\sqsupseteq$ that tests whether the LHS term is a compositional superset of the RHS term. Formally, we define the predicate $\sqsupseteq$ with the following two axioms:

$$\forall a, b.\ a = b \ \supset\ a \sqsupseteq b \tag{23}$$

$$\forall a, b, c.\ a \sqsupseteq b \circ c \ \supset\ a \sqsupseteq b \tag{24}$$

Of course, a number of obvious corollaries follow from the associative-commutative nature of the $\circ$ function.

Given the $\sqsupseteq$ predicate, it is easy to specify the successor-state axiom (SSA) for the fluent $Running(c, s)$ in terms of the full joint deterministic action:

$$Running(c, do(a, s)) \equiv a \sqsupseteq runningS(c) \vee Running(c, s) \wedge \neg(a \sqsupseteq runningF(c)) \tag{25}$$

One advantage of factoring joint stochastic actions into their stochastic sub-actions is that we can identify when the regression of a formula ϕ is *independent* of a sub-action. We will show how this independence can be exploited later during first-order decision theoretic regression; for now, we simply show how this independence can be detected and proved.

In order to make these proofs, we first need to assert a few crucial axioms. The first axiom asserts that every joint deterministic action includes an outcome for every deterministic sub-action:

$$\forall a \in A,\ c_i \in C.\ \bigvee_{b \in A(c_i)} a \sqsupseteq b \tag{26}$$

To see where this axiom may be used, one immediate consequence of this axiom is that the SSA in Axiom 25 simplifies to the following:

$$Running(c, do(a, s)) \equiv a \sqsupseteq runningS(c) \tag{27}$$

Proof Sketch. *We know from Axiom 26 that for each $c \in C$, $a \sqsupseteq runningS(c) \vee a \sqsupseteq runningF(c)$ or rearranged, $\neg(a \sqsupseteq runningF(c)) \supset a \sqsupseteq runningS(c)$. Thus, we need only apply Modus Ponens and simplify to obtain Axiom 27 from Axiom 25.*

The next unique names axiom ensures distinctness of all sub-action outcomes from each other:

$$\forall c_i \in C, c_j \in \{C \setminus c_i\},\, a \in A(c_i),\, b \in A(c_j).\, a \neq b \tag{28}$$

With these axioms in place, we can now make the following definition:

Definition 4.1. *We say a deterministic sub-action b is irrelevant to a formula $\phi(s)$ (abbreviated $Irr[\phi(s), b]$)* iff *the following equivalence holds:*

$$Regr(\phi(do(b, s))) \equiv \phi(s)\ . \tag{29}$$

In general, we can prove case equivalence by converting the case representation to its logical equivalent [2] and querying an off-the-shelf theorem prover. Most often though, a simple structural comparison will allow us to show simple syntactic equivalence without the need for theorem proving.

To fully exploit irrelevance of sub-actions, we would like to prove that the following 2nd-order axiom holds:

$$\forall \phi,\, a \in A.\, a = b \circ c \wedge Irr[\phi(s), b] \supset \\ \phi(do(b \circ c, s)) \equiv \phi(do(c, s)) \tag{30}$$

We will see that such an axiom is crucial to simplifying the value function representation during decision-theoretic regression. But we first need to define the conditions under which this axiom holds true.

The key to proving this axiom relies on assuming that the composition of two sub-actions $b \circ c$ can have no more effects than given by the union of the effects of b and c individually. To this end, let us informally introduce the following assumption:

Assumption 4.2. *No two different stochastic sub-actions can affect the same ground fluent.*

Effectively we are claiming here that the effects of all sub-actions are orthogonal and thus $b \circ c$ cannot interfere with each other. It is easy to see that this assumption holds for SysAdmin because each stochastic sub-action outcome $a \in A(c_i)$ affects only $Running(c_i, s)$ (and no other $Running(c_j, s)$ when $c_j \neq c_i$).

Now we can present the following informal proof sketch of Axiom 30 given Definition 4.1 and Assumption 4.2.

Proof Sketch. *For any $a \in A$ and b such that $a = b \circ c \wedge Irr[\phi(s), b]$, we know that b has no effect on $\phi(s)$ as a result of Definition 4.1. Furthermore, Assumption 4.2 tells us that the effects of sub-actions b and c cannot interfere with each other so the sole effects that can affect the regression of $\phi(s)$ are from c. This allows us to arrive at the conclusion that $\phi(do(b \circ c, s)) \equiv \phi(do(c, s))$, thus showing that Axiom 30 must hold in this context.*

A complete formal proof would place restrictions on the effect axioms directly and analyze regression in terms of the successor-state axioms that are derived from the effect axioms.

4.3 Transition dynamics

We will now proceed to specify Nature's probability distribution over deterministic sub-actions $P(A(c_i)|U(c_i))$ that we deferred from the last section. To begin, we specify the probabilities that $A(c_i)$ takes on the successful outcome $runningS(c_i)$ when $U(c_i)$ takes on the two possibilities $reboot(c_i)$ and $noop(c_i)$:

$$P(runningS(c_i)|reboot(c_i)) = \boxed{\top : 1} \tag{31}$$

$$P(runningS(c_i)|noop(c_i)) = \tag{32}$$

$$\boxed{\begin{array}{l} Running(c_i, s) \ : \ 0.95 \\ \neg Running(c_i, s) : \ 0.05 \end{array}} \otimes \frac{\sum_d \left(\boxed{\begin{array}{l} Conn(c_i, d) \wedge Running(d, s) \ : \ 1 \\ \neg Conn(c_i, d) \vee \neg Running(d, s) : \ 0 \end{array}} \right)}{\sum_d \left(\boxed{\begin{array}{l} Conn(c_i, d) \ : \ 1 \\ \neg Conn(c_i, d) : \ 0 \end{array}} \right)}$$

Here we see that the probability that a computer will be running if it was explicitly rebooted is 1. Otherwise, if it was not explicitly rebooted, this will depend on its previous state and the status of other computers with incoming connections.

Note that the probability of the failure outcome $runningF(c_i)$ is just the complement of the success case:

$$P(runningF(c_i)|U(c_i)) = \boxed{\top : 1} \ominus P(runningS(c_i)|U(c_i)) \tag{33}$$

Given these sub-action distributions $P(A(c_i)|U(c_i))$, we can easily combine them into a joint probability $P(A|U)$ as given previously by Equation 22.

5 Symbolic Dynamic Programming for FOMDPs with Sum/Product Aggregators and Factored Actions

5.1 First-order Decision-theoretic Regression

Now that we have specified a compact representation of Nature's probability distribution over deterministic actions, we will exploit this structure in the backup operators that are the building blocks of symbolic dynamic programming.

We begin with the basic definition of the first-order decision-theoretic regression (FODTR) of a value function through a stochastic joint action chosen from U $(= U(c_1) \circ \cdots \circ U(c_n))$:

$$FODTR(vCase(s), U) = \tag{34}$$

$$\gamma \bigoplus_{a_1 \in A(c_1), \ldots, a_n \in A(c_n)} \left[\left(\prod_{i=1}^{n} P(a_i|U(c_i)) \right) \otimes Regr(vCase(do(a_1 \circ \cdots \circ a_n, s))) \right]$$

On the inside of the $\bigoplus$, we have the probability of the joint deterministic action $a_1 \circ \cdots \circ a_n$ multiplied by the regression of the value function through the action $a_1 \circ \cdots \circ a_n$. On the outside of this, the $\bigoplus$ sums over all possible deterministic action outcomes given the stochastic joint action U. This is essentially just the factored version of standard FODTR [2].

We can exploit the fact that certain sub-actions are irrelevant to $vCase(s)$ and can be summed out to 1. This is just a generalization of the DBN independence that can be exploited in factored MDPs. However, in factored MDPs, irrelevance can be determined by a simple syntactic analysis of propositions, whereas here we must prove irrelevance by satisfying Definition 4.1.

As a hint of what is to come, we now demonstrate how to exploit irrelevance of sub-actions. First, we can arbitrarily reorder the sum as needed. Say that we have pushed a_n to the innermost sum and expanded it into its two cases:

$$
\begin{aligned}
&FODTR(vCase(s), U) \qquad (35)\\
&= \gamma \bigoplus_{a_1 \in A(c_1), \ldots, a_{n-1} \in A(c_{n-1})} \Big[\Big(\prod_{i=1}^{n-1} P(a_i | U(c_i)) \Big) \otimes P(runningS(c_n) | U(c_n)) \\
&\qquad \otimes Regr(vCase(do(a_1 \circ \cdots \circ a_{n-1} \circ runningS(c_n), s))) \\
&\qquad \oplus \Big(\prod_{i=1}^{n-1} P(a_i | U(c_i)) \Big) \otimes P(runningF(c_n) | U(c_n)) \\
&\qquad \otimes Regr(vCase(do(a_1 \circ \cdots \circ a_{n-1} \circ runningF(c_n), s))) \Big]
\end{aligned}
$$

If we can prove both $Irr[vCase(s), runningS(c_n)]$ and $Irr[vCase(s), runningF(c_n)]$ according to Definition 4.1, we can then derive the following simplified representation as a consequence of Axiom 30.

$$
\begin{aligned}
&FODTR(vCase(s), U) \qquad (36)\\
&= \gamma \bigoplus_{a_1 \in A(c_1), \ldots, a_{n-1} \in A(c_{n-1})} \Big[\Big(\prod_{i=1}^{n-1} P(a_i | U(c_i)) \Big) \otimes Regr(vCase(do(a_1 \circ \cdots \circ a_{n-1}, s))) \\
&\qquad \otimes \Big(P(runningS(c_n) | U(c_n)) \oplus P(runningF(c_n) | U(c_n)) \Big) \Big]
\end{aligned}
$$

Noting from Axiom 33 that the second line sums to 1, we can remove it and obtain the following simplified FODTR of $vCase(s)$ through U:

$$
\begin{aligned}
&FODTR(vCase(s), U) \qquad (37)\\
&= \gamma \bigoplus_{a_1 \in A(c_1), \ldots, a_{n-1} \in A(c_{n-1})} \Big[\Big(\prod_{i=1}^{n-1} P(a_i | U(c_i)) \Big) \otimes Regr(vCase(do(a_1 \circ \cdots \circ a_{n-1}, s))) \Big]
\end{aligned}
$$

Thus, we can exploit irrelevance between the value function and the factored action space to efficiently perform FODTR without summing explicitly over the full joint deterministic action space.

As an aside, we note that FODTR is a linear operator and thus can be distributed through the $\oplus$ case operator. We will heavily exploit this fact later when we introduce value-approximation using basis functions.

5.2 Backup Operators

Based on FODTR, we can define two backup operators [3, 4]:

$$B^{U(\vec{x})}(vCase(s)) = rCase(s,a) \oplus FODTR(vCase(s), U(\vec{x})) \tag{38}$$

$$B^{U}_{\max}(vCase(s)) = \max(\exists \vec{x}\ B^{U(\vec{x})}(vCase(s))) \tag{39}$$

Applying $B^{U(\vec{x})}(vCase(s))$ follows directly from the definition. Here we depart from our previous notation of U for stochastic joint actions and instead use $U(\vec{x})$ to explicitly indicate that the stochastic joint action U may have parameters $\vec{x}$.

Applying $B^{U}_{\max}(vCase(s))$ is much harder. While in principle, we can keep the max and $\exists \vec{x}$ in symbolic form and have an immediate solution, this prevents us from exploiting structure that is crucial to simplification and thus central to the tractability of a symbolic dynamic programming solution approach.

Up to this point, we have specified an FODTR operator that is independent of any specific SysAdmin connection topology. However, to efficiently perform backups and maximization, we must have more in-depth knowledge about the constraints on the topological structure of the domain. Here we will draw on the axiomatizations of the SysAdmin unidirectional ring (UNI), bidirectional ring (BI), and star networks (STAR) as previously discussed in Section 2.

In addition to these axiomatizations, we will also exploit the following assumption:

Assumption 5.1. *For all joint stochastic actions $U = U(c_1) \circ \cdots \circ U(c_n)$ and all deterministic sub-action outcomes $A(c_i)$, there must be a finite, constant upper bound on the number of distinct probability distributions for $P(A(c_i)|U(c_i))$.*

Clearly, this assumption holds for all SysAdmin problems discussed so far. The number of distinct probability distributions for $P(A(c_i)|U(c_i))$ is clearly limited by the maximum number of incoming connections that a computer can have (c.f. Equations 31 and 32). One scenario that would be prohibited by this assumption is a computer that can be connected to all other computers in the network – then the number of unique $P(A(c_i)|U(c_i))$ distributions would grow linearly with the size of the network (i.e., a distinct probability for every count of incoming connections) thereby violating the constant upper bound assumption.

Why is this assumption important? Very crudely, it means that the representation resulting from a $B^{U(\vec{x})}(vCase(s))$ will only have a finite portion of its structure affected by an action, leaving an indefinite amount of identical structure. Firstly, this is important for obtaining a compact representation of the backup operator using our current notation. Secondly, this is important because when performing the maximization in $B^{U}_{\max}(vCase(s))$, we can exploit the identical infinite structure to obtain a compact representation of the result.

Let us provide a formal example of how we can perform $B^U_{\max}(vCase(s))$ for the SysAdmin problems. Say that we start with $vCase(s) = rCase(s)$ from Equation 18. Then we apply FODTR to this value function for stochastic action U and push the $Regr$ in as far as possible to obtain the following:

$$FODTR(vCase(s), U) = \tag{40}$$
$$\gamma \bigoplus_{a_1 \in A(c_1), \ldots, a_n \in A(c_n)} \Big[\Big(\prod_{i=1}^{n} P(a_i|U) \Big) \otimes \sum_c \boxed{\begin{array}{l} Regr[Running(c, do(a_1 \circ \cdots \circ a_n, s))] \ : \ 1 \\ Regr[\neg Running(c, do(a_1 \circ \cdots \circ a_n, s))] : \ 0 \end{array}} \Big]$$

Now we distribute the $\prod$ through the $\sum$ and reorder $\sum$ with $\bigoplus$:

$$FODTR(vCase(s), U) = \tag{41}$$
$$\gamma \sum_c \Big[\bigoplus_{a_1 \in A(c_1), \ldots, a_n \in A(c_n)} \Big(\prod_{i=1}^{n} P(a_i|U) \Big) \otimes \boxed{\begin{array}{l} Regr[Running(c, do(a_1 \circ \cdots \circ a_n, s))] \ : \ 1 \\ Regr[\neg Running(c, do(a_1 \circ \cdots \circ a_n, s))] : \ 0 \end{array}} \Big]$$

This last step is extremely important because it captures the factored probability distribution $\prod$ inside a specific c being summed over. Thus, for all SysAdmin problems, we now exploit the fact that based on their axiomatization, we can prove $Irr[Running(c, s), a]$ for all $a \in A(d)$ s.t. $d \neq c$. Thus, we can again simplify using Axiom 30:

$$FODTR(vCase(s), U) = \gamma \sum_c \Big[\bigoplus_{a \in A(c)} P(a|U) \otimes \boxed{\begin{array}{l} Regr[Running(c, do(a, s))] \ : \ 1 \\ Regr[\neg Running(c, do(a, s))] : \ 0 \end{array}} \Big]$$

Now we go ahead and perform the $\bigoplus$ over the explicit sub-actions. To expand the $P(a|U)$ into something concrete, we assume that $U = noop(c_1) \circ \cdots \circ reboot(x) \circ \cdots \circ noop(c_n)$.

$$FODTR(vCase(s), U = \cdots \circ reboot(x) \circ \cdots) = \gamma \boxed{\top : 1} \oplus \gamma \sum_{c \in C \setminus \{x\}} \tag{42}$$
$$\frac{\sum_d \left(\boxed{\begin{array}{l} Conn(c,d) \wedge Running(d,s) \ : \ 1 \\ \neg Conn(c,d) \vee \neg Running(d,s) : \ 0 \end{array}} \right)}{\sum_d \left(\boxed{\begin{array}{l} Conn(c,d) \ : \ 1 \\ \neg Conn(c,d) : \ 0 \end{array}} \right)} \otimes \boxed{\begin{array}{l} Regr[Running(c, do(RunningS(c), s))] \ : \ 1 \\ Regr[\neg Running(c, do(RunningS(c), s))] : \ 0 \end{array}}$$
$$\oplus \left(1 - \frac{\sum_d \left(\boxed{\begin{array}{l} Conn(c,d) \wedge Running(d,s) \ : \ 1 \\ \neg Conn(c,d) \vee \neg Running(d,s) : \ 0 \end{array}} \right)}{\sum_d \left(\boxed{\begin{array}{l} Conn(c,d) \ : \ 1 \\ \neg Conn(c,d) : \ 0 \end{array}} \right)} \right) \otimes \boxed{\begin{array}{l} Regr[Running(c, do(RunningF(c), s))] \ : \ 1 \\ Regr[\neg Running(c, do(RunningF(c), s))] : \ 0 \end{array}}$$

Note that we've separated out the case where $x = c$ and replaced it with $\boxed{\top : 1}$ (the reader can expand out the probability and regression manually to verify this). In addition, we note the following results of $Regr$:

$$Regr[Running(c_i, do(RunningS(c_i), s))] = \top \tag{43}$$
$$Regr[Running(c_i, do(RunningF(c_i), s))] = \bot \tag{44}$$

Noting that the bottom product simplifies to 0 and the regressed portion of the top product simplifies to 1, we obtain the final pleasing result:

$$FODTR(vCase(s), U = \cdots \circ reboot(x) \circ \cdots) = \tag{45}$$
$$\gamma \boxed{\top : 1} \oplus \gamma \sum_{c \in C \setminus \{x\}} \frac{\sum_d \left(\begin{array}{|l} \hline Conn(c,d) \wedge Running(d,s) \quad : 1 \\ \hline \neg Conn(c,d) \vee \neg Running(d,s) : 0 \\ \hline \end{array} \right)}{\sum_d \left(\begin{array}{|l} \hline Conn(c,d) \quad : 1 \\ \hline \neg Conn(c,d) : 0 \\ \hline \end{array} \right)}$$

Now, if we apply $B^U_{\max}(vCase(s))$, this involves existentially quantifying the action parameters (in this case x) and applying the case maximization operator max:

$$B^{U=\cdots\circ reboot(x)\circ\cdots}_{\max}(vCase(s)) = \sum_c \begin{array}{|l} \hline Running(c,s) \quad : 1 \\ \hline \neg Running(c,s) : 0 \\ \hline \end{array}$$
$$\oplus\, \gamma \max \exists x. \left(\boxed{\top : 1} \oplus \gamma \sum_{c \in C \setminus \{x\}} \frac{\sum_d \left(\begin{array}{|l} \hline Conn(c,d) \wedge Running(d,s) \quad : 1 \\ \hline \neg Conn(c,d) \vee \neg Running(d,s) : 0 \\ \hline \end{array} \right)}{\sum_d \left(\begin{array}{|l} \hline Conn(c,d) \quad : 1 \\ \hline \neg Conn(c,d) : 0 \\ \hline \end{array} \right)} \right) \tag{46}$$

Since we have simplified the structure within the "$\max \exists x$" w.r.t. to UNI constraints, we can now simplify this further – this would be *much* harder if we had an indefinite $\prod$ to deal with.

Clearly, for every possible ground state configuration, to achieve the "$\max \exists x$", we want to choose the instantiation of x that yields the maximum evaluation. So, we need only specify a policy that selects the best x. To specify the policy, we will Skolemize the x in $\exists x$ to a constant c^* and specify additional KB axioms for "choosing" the maximizing c^*.

We begin by noting that if we choose one c^* over some other c^*, we get a tradeoff (or differential) in value. We can express this differential as a case statement $\Delta case(\vec{x}, s)$ that is a function of A's action parameters $\vec{x}$ and the situation s. For the above case we get:

$$\Delta case(x,s) = 1 \ominus \frac{\sum_d \left(\begin{array}{|l} \hline Conn(x,d) \wedge Running(d,s) \quad : 1 \\ \hline \neg Conn(x,d) \vee \neg Running(d,s) : 0 \\ \hline \end{array} \right)}{\sum_d \left(\begin{array}{|l} \hline Conn(x,d) \quad : 1 \\ \hline \neg Conn(x,d) : 0 \\ \hline \end{array} \right)} \tag{47}$$

Note the $\ominus$ to indicate that while we gain 1 on the LHS of $\ominus$ for choosing x, we lose the complex term on the RHS of $\ominus$ by removing it from the sum.

Given a generic $\Delta case(\vec{x}, s)$, we can now formulate a policy axiom that

chooses the maximizing $\vec{c}^*$ for any situation:[3]

$$\forall \vec{x}, \vec{y}, v_1, v_2.\ \vec{x} \neq \vec{y} \wedge v_1 = \Delta case(\vec{x}, s) \wedge v_2 = \Delta case(\vec{y}, s) \wedge v_1 \geq v_2 \wedge \\ [\forall \vec{z}, v_3.\ \vec{x} \neq \vec{z} \wedge v_3 = \Delta case(\vec{z}, s) \wedge v_1 = v_3 \supset lex(\vec{x}, \vec{z})] \supset \vec{c}^* = \vec{x} \quad (48)$$

This straightforwardly compares the differential of all possible action parameters and chooses the parameters that maximize this differential. The extra condition inside brackets uses a lexicographic ordering predicate $lex(\cdot, \cdot)$ to enforce that the highest lexicographic precedence is chosen among maximal choices that are equal (this prevents more than one c^* from being "chosen"). We note that if we take into account the topological constraints to generate a finite form for $\Delta case(\vec{x}, s)$, we can compile this axiom into a simple finite-length FOL axiom required for use with a theorem prover.

And now with policy Axiom 48 enforcing the max for the Skolemized version of Equation 49, we have arrived at a specification of $B^U_{\max}(vCase(s))$:

$$B^{U=\cdots\circ reboot(x)\circ\cdots}_{\max}(vCase(s)) = \sum_c \boxed{\begin{array}{l} Running(c,s) \ : \ 1 \\ \neg Running(c,s) : \ 0 \end{array}} \\ \oplus \gamma \left(\boxed{\top : 1} \oplus \gamma \sum_{c \in C \setminus \{c^*\}} \frac{\sum_d \left(\boxed{\begin{array}{l} Conn(c,d) \wedge Running(d,s) \ : \ 1 \\ \neg Conn(c,d) \vee \neg Running(d,s) : \ 0 \end{array}} \right)}{\sum_d \left(\boxed{\begin{array}{l} Conn(c,d) \ : \ 1 \\ \neg Conn(c,d) : \ 0 \end{array}} \right)} \right) \quad (49)$$

This result is in summation form, which is important for exploiting structure in subsequent backups.

While this has been a specific example, we note that the technique of 1) simplifying the FODTR w.r.t. network topology, 2) putting the result in sum form, and 3) analyzing tradeoffs in the sum to generate a policy will produce proper backups in summation form for all cases in the SysAdmin problem.

5.3 Symbolic Dynamic Programming

So long as our domain satisfies Assumption 5.1, we can easily perform symbolic dynamic programming for multiple actions of different names: Because of the assumption, we know that only a finite number of sum elements in the backed up value function for action Y are "different" from those for Z, let's place these different elements into two sets Y_{diff} and Z_{diff}. Then we obtain the following abstract representation of the backup of the same value function for these two actions:

$$B^Y_{\max}(vCase(s)) = \sum_{c \in Y_{diff}} case_Y(c,s) \oplus \sum_{c \notin Y_{diff}} case_{YZ}(c,s) \quad (50)$$

$$B^Z_{\max}(vCase(s)) = \sum_{c \in Z_{diff}} case_Z(c,s) \oplus \sum_{c \notin Z_{diff}} case_{YZ}(c,s) \quad (51)$$

[3]This axiom relies on the original definition of the case notation [2].

Now, when we go to perform the max over these two actions, we can exploit our above assumption and note that the case elements not in Y_{diff} and Z_{diff} are the same:

$$\begin{aligned}
&\max[B^Y_{\max}(vCase(s)), B^Z_{\max}(vCase(s))] \qquad (52)\\
&= \max\Big[\sum_{c \in Y_{diff}} case_Y(c,s) \oplus \sum_{c \notin Y_{diff}} case_{YZ}(c,s),\\
&\qquad \sum_{c \in Z_{diff}} case_Z(c,s) \oplus \sum_{c \notin Z_{diff}} case_{YZ}(c,s)\Big]\\
&= \max\Big[\sum_{c \in Y_{diff}} case_Y(c,s) \oplus \sum_{c \in Z_{diff}} case_Y(c,s) \oplus \sum_{c \notin Y_{diff} \cup Z_{diff}} case_{YZ}(c,s),\\
&\qquad \sum_{c \in Y_{diff}} case_Z(c,s) \oplus \sum_{c \in Z_{diff}} case_Z(c,s) \oplus \sum_{c \notin Y_{diff} \cup Z_{diff}} case_{YZ}(c,s)\Big]\\
&= \max\Big[\sum_{c \in Y_{diff}} case_Y(c,s) \oplus \sum_{c \in Z_{diff}} case_Y(c,s),\\
&\qquad \sum_{c \in Y_{diff}} case_Z(c,s) \oplus \sum_{c \in Z_{diff}} case_Z(c,s)\Big] \oplus \sum_{c \notin Y_{diff} \cup Z_{diff}} case_{YZ}(c,s)
\end{aligned}$$

Because we know that the sets Y_{diff} and Z_{diff} are finite, we can perform the explicit max over these sets and obtain an explicit version of the max over the backup of actions Y and Z with an indefinite sum. If at most k case statements are affected differently per action backup, we have to deal with at most a 2^{2k} blowup in the representation size, i.e. a blowup of 2^k potentially incurred by explicitly summing over Y_{diff} and Z_{diff} followed by a quadratic blowup after taking the cross product of all cases to determine the max.[4]

6 Linear-value Approximation

6.1 Basis Functions

In this section, we demonstrate how we can represent the value function for FOMDPs defined with rewards expressed using sum aggregators. We represent each first-order basis function as a sum of k basis functions, each consisting of a sum over all c of a $bCase_i(c,s)$ statement:

$$vCase(s) \quad = \quad \bigoplus_{i=1}^{k} w_i \sum_c bCase_i(c,s)$$

Even though we are only finding an approximation of the value function, it should not be hard to see that only a few simple first-order basis functions

[4] There is another technique for performing the max that unions case partitions, sorts them, and renders them disjoint that has a linear rather than quadratic space complexity. However, this approach generally causes the size of the FOL formulae to become so large that a theorem prover cannot tractably refute the inconsistent formulae.

should account for a large portion of the optimal policy. A similar version of this approach using unary, binary, and ternary conjunctions of propositions worked well in the propositional version of SysAdmin [7], and there the number of basis functions could be combinatorically explosive (e.g., a basis function for every pair of computers in a network of 40 computers). Here, we can exploit symmetry to represent all of these basis functions very simply. In fact, for our initial solution, we will simply work with the following value function representation that *accounts for the single (unary) and pair (binary) basis functions* used in the current literature:

$$vCase(s) = w_1 \sum_c \begin{array}{|c|}\hline Running(c,s) : 1 \\ \hline \neg Running(c,s) : 0 \\ \hline \end{array} \oplus w_2 \sum_c \begin{array}{|c|}\hline Running(c,s) \wedge \exists c_2 Conn(c,c_2) \wedge Running(c_2) : 1 \\ \hline \neg(Running(c,s) \wedge \exists c_2 Conn(c,c_2) \wedge Running(c_2)) : 0 \\ \hline \end{array}$$

There are a few reasons for representing the value function in this manner:

1. Our approximate value function *should* be able to exactly represent the reward. Clearly the sum over the first basis function allows us to exactly represent the reward, while if it were defined with an $\exists c$ as opposed to a $\sum_c$, it would be impossible for a fixed-weight value function to *scale proportionally* to the reward as the domain size increased.

2. This value function representation, which succinctly represents all single and pair basis functions for the SysAdmin domain with just two first-order basis functions, *uses implicit parameter tying to reduce the number of parameters to estimate*. That is, each basis function represents an unbounded set of ground basis functions for every c and each of these ground basis function instantiations is required to have the same weight. In the absence of any method for diffentiating one basis function from another, this parameter tying considerably reduces the complexity of the weight estimation problem.

We make one additional note that the variable type being summed over in the reward and the variable type in the actions matches that in the $\sum$ aggregators of the basis functions (i.e., they are all c). In general, to exploit the techniques that we describe in this section, it is important that the object being summed over in the reward and basis functions is also the action parameter in a factored action specification.

6.2 Basis Function Generation

Just as done in [4], we can generate basis functions as regressions of the reward. For example, if the reward $rCase(s)$ is as given previously in Equation 18, then this will be our first basis function:

$$bCase_1(s) = \sum_c \left(\begin{array}{|c|}\hline Running(c,s) : 1 \\ \hline \neg Running(c,s) : 0 \\ \hline \end{array} \right) \tag{53}$$

Our next basis function could be a regression of $bCase_1(s)$ through a stochastic joint action for a *noop*:

$$bCase_2(s) = \sum_c \frac{\sum_d \left(\begin{array}{|l} \hline Conn(c,d) \wedge Running(d,s) \quad : 1 \\ \hline \neg Conn(c,d) \vee \neg Running(d,s) : 0 \\ \hline \end{array} \right)}{\sum_d \left(\begin{array}{|l} \hline Conn(c,d) \quad : 1 \\ \hline \neg Conn(c,d) : 0 \\ \hline \end{array} \right)} \tag{54}$$

6.3 Redefining the Backup Operators

Before we provide approximate solution algorithms for FOMDPs with sum aggregators, we digress for a moment to redefine the backup operators from Section 5.2. When performing these backups, we can use the linearity of FODTR to exploit the structure of a value function represented as a linear combination of basis functions. First, we'll start with $B^{U(\vec{x})}(vCase(s))$ and rewrite it for the case when $vCase(s) = \bigoplus_{i=1}^{k} w_i \cdot \sum_c bCase_i(c,s)$:

$$B^{U(\vec{x})}(vCase(s)) = rCase(s,a) \oplus FODTR\left(\bigoplus_{i=1}^{k} w_i \cdot \sum_c bCase_i(c,s), U(\vec{x}) \right) \tag{55}$$

$$= rCase(s,a) \oplus \bigoplus_{i=1}^{k} w_i \cdot \sum_c FODTR\left(bCase_i(c,s), U(\vec{x})\right) \tag{56}$$

Here, we have exploited the linearity of the FODTR operator to push it through to each individual basis function. As such, since $B^{U}_{\max}(vCase(s))$ is defined in terms of $B^{U(\vec{x})}(vCase(s))$, we do not need to modify the definition of this operator. However, we note that portions of multiple basis functions may now be bound together by the existential quantifier. While this can be avoided by Skolemizing the quantifier as done previously, the policy must now reflect the joint sum of changes across all basis functions that are affected by an action selection. This means that the policy representation will likely grow exponentially in size with the number of basis functions.

7 First-order Approximate Linear Programming

Now, we generalize the first-order approximate linear programming (FOALP) approach [3] to solving FOMDP domains containing the sum aggregator:

Variables: w_i ; $\forall i \leq k$

$$\text{Minimize:} \sum_s \bigoplus_{i=1}^{k} w_i \cdot \sum_c bCase_i(c,s) \tag{57}$$

$$\text{Subject to:} 0 \geq \left[B^U_{\max} \left(\bigoplus_{i=1}^{k} w_i \cdot \sum_c bCase_i(c,s) \right) \ominus \bigoplus_{i=1}^{k} w_i \cdot \sum_c bCase_i(c,s) \right] ; \forall U, s$$

Since we are summing over infinitely many situations in the FOALP objective, the objective is ill-defined. Thus, we redefine the FOALP objective in a manner that preserves the intention of the original approximate linear programming solution for MDPs. Rather than count ground state configurations for every c in our FOALP objective—of which there would be an infinite number per partition—we suppose that each basis function partition is chosen because it represented a potentially useful partitioning of state space, and thus weight each case *partition* equally. Consequently, we approximate the above FOALP objective as the following:

$$\begin{aligned} \sum_s \bigoplus_{i=1}^{k} w_i \cdot \sum_c bCase_i(c,s) &= \bigoplus_{i=1}^{k} \sum_s \sum_c w_i \cdot bCase_i(c,s) \\ &\sim \sum_{i=1}^{k} w_i \sum_{\langle \phi_j, t_j \rangle \in bCase_i} \frac{t_j}{|bCase_i|} \end{aligned} \tag{58}$$

Here, $|bCase_i|$ represents the number of partitions in $bCase_i$ and for each basis function, we sum over the value t_j of each partition $\langle \phi_j, t_j \rangle \in bCase_i$ normalized by $|bCase_i|$. This gives an approximation of the importance of each weight w_i in proportion to the overall value function.

Now we turn to solving the constraints via constraint generation. We will not cover the general constraint generation method here, we refer the reader to [3] for the basic ideas of solving first-order LPs via constraint generation.

Given the current solution to the first-order LP in Equation 57 (i.e., settings for the weights w_i) during constraint generation, our goal is to determine the maximally violated constraint of the following general form:

$$0 \geq \max_s \left(\sum_c case_1(c,s) \oplus \ldots \oplus \sum_c case_k(c,s) \right) \tag{59}$$

By virtue of the structure of the basis functions, the properties of the $B^U_{\max}$ operator, and the commutativity of $\sum$ and $\bigoplus$, it is easy to verify that the constraints of Equation 57 can always be converted to this form. By additionally making use of Property 13, we can pull out the $\sum_c$ to obtain the following constraint form:

$$0 \geq \max_s \sum_c \left(case_1(c,s) \oplus \ldots \oplus case_k(c,s) \right) \tag{60}$$

Now to solve the constraint generation problem, we need only (efficiently) evaluate the $\max_s$ when the constraints are given by a sum over case statements and compare the value to 0 to determine if it is a constraint violation. If there is no violation, then we have the optimal solution to the first-order LP.

While we know how to evaluate a $\max_s$ for the first-order case without sum aggregators [3], it is a little more difficult with sum aggregators because the constraints are indefinitely large (i.e., a sum aggregator is an indefinitely long summation). However, there is a lot of structure in the constraints and thus in many cases we will be able to exploit this to obtain a closed-form representation of the constraints without a sum aggregator.

7.1 Obtaining a Closed-form Representation of Constraints

To simplify the general problem, we will compute the explicit $\oplus$ inside of the $\sum_c$ in Equation 60 to obtain the simplest representation of the constraints:

$$0 \geq \max_s \sum_c case(c, s) \tag{61}$$

Now, if we examine the structure of $case(c, s)$, we note that it is a normal case statement except that it may have predicates that mention the free variable c. We are going to use relation elimination in the same way as [3] to efficiently compute the max in a factored manner. But we would also like to eliminate the c's in a specified order that minimizes the tree-width of the connectivity for the elimination order. If we were to simply eliminate a relation $R(c, \vec{x})$ then we would immediately need to sum over *all* summands of $\sum_c$. To avoid this, we specify the following axiom template that allows us to redefine $R(c, \vec{x})$ into n *different predicates*:[5]

$$\forall c \in C \; [R(c, s) \equiv Rc(s)] \tag{62}$$

One axiom must be created for every $c \in C$. Note that we introduce a new reified predicate $Rc(s)$ for every possible c. Having done this, we can now eliminate the Rc's in a given order that minimizes the tree-width of the elimination.

For the SysAdmin problem, since we have axioms specifying how the c's influence each other, it is straightforward to compute a good elimination order. Then we can exploit the uniformity of the constraints in Equation 61 to perform the $\max_s$ without analyzing each individual c. Next we show how we can do this for a variety of SysAdmin networks.

7.1.1 Single Line Network

Although it is not one of the original networks that we had specified in Section 2, we will find that the simple directed line configuration of Figure 2 is a simple building block, which we can use to perform relation elimination on the more complex SysAdmin networks described in the previous section.

[5] This is not an FOL axiom itself, but rather a syntactic template for creating an FOL axiom for every c. For example, for c_1, the axiom would be $R(c_1, s) \equiv Rc_1(s)$.

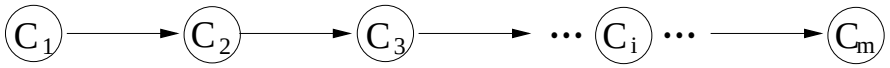


Figure 2: A directed graph depicting a line of computers.

For demonstrative purposes, we assume that we want to compute the max of the following constraint:

$$0 \geq \max_{Rc_1,\ldots,Rc_m} \sum_c \left(\begin{array}{|l:l|} \hline R(c,s) \wedge \exists d.\, Conn(c,d) \wedge R(d,s) & :\ 1 \\ \hline \neg R(c,s) \vee \forall d.\, \neg Conn(c,d) \vee \neg R(d,s) & :\ 0 \\ \hline \end{array} \right) \tag{63}$$

Note that the maximization is over the relations $\{Rc_1, \ldots, Rc_m\}$ including the grounded, renamed versions of R given in Equation 62 in place of the relation R. Thus, when eliminating Rc_1 we need to take into account an axiomatization of the connectivity given in Figure 2 to determine all possible summands that could involve Rc_1.

Each predicate Rc_i is connected to only two other predicates via the $Conn(\cdot, \cdot)$ relation, so eliminating the Rc_i's in order from $Rc_1 \ldots Rc_m$ should minimize the other summands of $\sum_c$ that need to be explicitly summed. This relation elimination order will minimize the tree-width.

We start by eliminating Rc_2 (we will not eliminate Rc_1 since we reuse this constraint as a building block for the star and ring networks). We can push the max through to only the summands containing Rc_2, which we infer w.r.t. the connectivity axiomatization of Figure 2. This gives us the following:

$$0 \geq \max_{Rc_1,Rc_3,\ldots,Rc_m} \sum_{c \notin \{c_1,c_2\}} \left(\begin{array}{|l:l|} \hline R(c,s) \wedge \exists d.\, Conn(c,d) \wedge R(d,s) & :\ 1 \\ \hline \neg R(c,s) \vee \forall d.\, \neg Conn(c,d) \vee \neg R(d,s) & :\ 0 \\ \hline \end{array} \right)$$
$$\oplus \max_{Rc_2} \left(\begin{array}{|l:l|} \hline \{Rc_1(s)\ ,\ Rc_2(s)\} & :\ 1 \\ \hline \{\neg Rc_1(s) \vee \neg Rc_2(s)\} & :\ 0 \\ \hline \end{array} \oplus \begin{array}{|l:l|} \hline \{Rc_2(s)\ ,\ Rc_3(s)\} & :\ 1 \\ \hline \{\neg Rc_2(s) \vee \neg Rc_3(s)\} & :\ 0 \\ \hline \end{array} \right)$$

Because of the connectivity of the network, the summands for c_1 and c_2 mention Rc_2 so we remove these summands. Now to perform the max over these two summands, we follow the first-order factored max algorithm from [3]. In short this algorithm dictates (1) converting each partition formula to a set of CNF clauses, (2) performing the maximization, (3) performing all resolutions over the eliminated relation, (4) removing inconsistent partitions, (5) removing remaining clauses containing the eliminated relation, and (6) removing dominated partitions. Applying this procedure up to step (4) gives us the following

where we have removed clauses mentioning the eliminated relation Rc_2:

$$0 \geq \max_{Rc_1, Rc_3, \ldots, Rc_m} \sum_{c \notin \{c_1, c_2\}} \left(\begin{array}{|l|} \hline R(c,s) \wedge \exists d.\, Conn(c,d) \wedge R(d,s) \quad : 1 \\ \hline \neg R(c,s) \vee \forall d.\, \neg Conn(c,d) \vee \neg R(d,s) : 0 \\ \hline \end{array} \right)$$

$$\oplus \begin{array}{|l|} \hline \{Rc_1(s)\ ,\ Rc_2(s)\ ,\ Rc_2(s)\ ,\ Rc_3(s)\} \quad : 2 \\ \hline \{Rc_1(s)\ ,\ Rc_2(s)\ ,\ \neg Rc_2(s) \vee \neg Rc_3(s)\ ,\ \neg Rc_3(s)\} : 1 \\ \hline \{\neg Rc_1(s) \vee \neg Rc_2(s)\ ,\ Rc_2(s)\ ,\ Rc_3(s)\ ,\ \neg Rc_1(s)\} : 1 \\ \hline \{\neg Rc_1(s) \vee \neg Rc_2(s)\ ,\ \neg Rc_2(s) \vee \neg Rc_3(s)\} \quad : 0 \\ \hline \end{array}$$

Finally, we obtain:

$$0 \geq \max_{Rc_1, Rc_3, \ldots, Rc_m} \sum_{c \notin \{c_1, c_2\}} \left(\begin{array}{|l|} \hline R(c,s) \wedge \exists d.\, Conn(c,d) \wedge R(d,s) \quad : 1 \\ \hline \neg R(c,s) \vee \forall d.\, \neg Conn(c,d) \vee \neg R(d,s) : 0 \\ \hline \end{array} \right)$$

$$\oplus \begin{array}{|l|} \hline \{Rc_1(s)\ ,\ Rc_3(s)\} \quad : 2 \\ \hline \{Rc_1(s)\ ,\ \neg Rc_3(s)\} : 1 \\ \hline \{Rc_3(s)\ ,\ \neg Rc_1(s)\} : 1 \\ \hline \{\} \quad : 0 \\ \hline \end{array}$$

Next, we eliminate Rc_3 and obtain the following:

$$0 \geq \max_{Rc_1, Rc_4, \ldots, Rc_m} \sum_{c \notin \{c_1, c_2, c_3\}} \left(\begin{array}{|l|} \hline R(c,s) \wedge \exists d.\, Conn(c,d) \wedge R(d,s) \quad : 1 \\ \hline \neg R(c,s) \vee \forall d.\, \neg Conn(c,d) \vee \neg R(d,s) : 0 \\ \hline \end{array} \right)$$

$$\oplus \max_{Rc_3} \left(\begin{array}{|l|} \hline \{Rc_3(s)\ ,\ Rc_4(s)\} \quad : 1 \\ \hline \{\neg Rc_3(s) \vee \neg Rc_4(s)\} : 0 \\ \hline \end{array} \oplus \begin{array}{|l|} \hline \{Rc_1(s)\ ,\ Rc_3(s)\} \quad : 2 \\ \hline \{Rc_1(s)\ ,\ \neg Rc_3(s)\} : 1 \\ \hline \{Rc_3(s)\ ,\ \neg Rc_1(s)\} : 1 \\ \hline \{\} \quad : 0 \\ \hline \end{array} \right)$$

As an intermediate step, we obtain the following:

$$0 \geq \max_{Rc_1, Rc_4, \ldots, Rc_m} \sum_{c \notin \{c_1, c_2, c_3\}} \left(\begin{array}{|l|} \hline R(c,s) \wedge \exists d.\, Conn(c,d) \wedge R(d,s) \quad : 1 \\ \hline \neg R(c,s) \vee \forall d.\, \neg Conn(c,d) \vee \neg R(d,s) : 0 \\ \hline \end{array} \right)$$

$$\oplus \begin{array}{|l|} \hline \{Rc_3(s)\ ,\ Rc_4(s)\ ,\ Rc_1(s)\ ,\ Rc_3(s)\} \quad : 3 \\ \hline \{Rc_3(s)\ ,\ Rc_4(s)\ ,\ Rc_1(s)\ ,\ \neg Rc_3(s)\} \quad : 2 \\ \hline \{Rc_3(s)\ ,\ Rc_4(s)\ ,\ \neg Rc_1(s)\ ,\ Rc_3(s)\} \quad : 2 \\ \hline \left(\begin{array}{|l|} \hline \{Rc_3(s)\ ,\ Rc_4(s)\} \quad : 1 \\ \hline \{\neg Rc_3(s) \vee \neg Rc_4(s)\ ,\ Rc_1(s)\ ,\ Rc_3(s)\ ,\ \neg Rc_4(s)\} : 2 \\ \hline \{\neg Rc_3(s) \vee \neg Rc_4(s)\ ,\ Rc_1(s)\ ,\ \neg Rc_3(s)\} \quad : 1 \\ \hline \{\neg Rc_3(s) \vee \neg Rc_4(s)\ ,\ \neg Rc_1(s)\ ,\ Rc_3(s)\} \quad : 1 \\ \hline \end{array} \right) \\ \hline \{\neg Rc_3(s) \vee \neg Rc_4(s)\} \quad : 0 \\ \hline \end{array}$$

After maxing and eliminating dominated case partitions, we obtain the follow-

ing:

$$0 \geq \max_{Rc_1, Rc_4, \ldots, Rc_m} \sum_{c \notin \{c_1, c_2, c_3\}} \left(\begin{array}{|l:l|} \hline R(c,s) \wedge \exists d.\, Conn(c,d) \wedge R(d,s) & 1 \\ \hline \neg R(c,s) \vee \forall d.\, \neg Conn(c,d) \vee \neg R(d,s) & 0 \\ \hline \end{array} \right)$$

$$\oplus \begin{array}{|l:l|} \hline \{Rc_4(s)\ ,\ Rc_1(s)\} & 3 \\ \hline \cancel{\{Rc_4(s)\ ,\ Rc_1(s)\}} & \cancel{2} \\ \hline \{Rc_4(s)\ ,\ \neg Rc_1(s)\} & 2 \\ \hline \{Rc_4(s)\} & 1 \\ \hline \{Rc_1(s)\ ,\ \neg Rc_4(s)\} & 2 \\ \hline \{Rc_1(s)\} & 1 \\ \hline \{\neg Rc_1(s)\} & 1 \\ \hline \{\} & 0 \\ \hline \end{array}$$

By disjoint the three partitions of value 1, we get the partition $Rc_1(s) \vee \neg Rc_1(s) \vee Rc_4(s) \equiv \top$. This results in the final simplification:

$$0 \geq \max_{Rc_1, Rc_4, \ldots, Rc_m} \sum_{c \notin \{c_1, c_2, c_3\}} \left(\begin{array}{|l:l|} \hline R(c,s) \wedge \exists d.\, Conn(c,d) \wedge R(d,s) & 1 \\ \hline \neg R(c,s) \vee \forall d.\, \neg Conn(c,d) \vee \neg R(d,s) & 0 \\ \hline \end{array} \right)$$

$$\oplus \begin{array}{|l:l|} \hline \{Rc_4(s)\ ,\ Rc_1(s)\} & 3 \\ \hline \{Rc_4(s)\ ,\ \neg Rc_1(s)\} & 2 \\ \hline \{Rc_1(s)\ ,\ \neg Rc_4(s)\} & 2 \\ \hline \{\} & 1 \\ \hline \end{array}$$

Now, at this point, if we repeat this process for all $Rc_4 \ldots Rc_m$, we see that there is a pattern and we can prove by induction that completing this process out to m steps yields the final result:

$$0 \geq \max_{Rc_1} \begin{array}{|l:l|} \hline \{Rc_1(s)\ ,\ Rc_{m+1}(s)\} & m \\ \hline \{\neg Rc_1(s)\ ,\ Rc_{m+1}(s)\} & m-1 \\ \hline \{Rc_1(s)\ ,\ \neg Rc_{m+1}(s)\} & m-1 \\ \hline \{\} & m-2 \\ \hline \end{array} \tag{64}$$

At this point, we cannot provide a general induction scheme for first-order factored max applications over sum aggregators and in fact, it is not clear that one exists for all cases. But now that we have solved for a line network when the constraints take the form given in Equation 63, we can use it as a building block to solve various networks having constraints of the same general form.

7.1.2 Unidirectional Ring

For the unidirectional ring network (UNI) in Figure 1(a), if we have the first-order LP constraint given by Equation 63 then it is easy to adapt our closed form version of the constraints given in Equation 64 for the Single Line Network to this case where $m = n$. Noting that the final connection $m + 1$ is actually

1 (given an appropriate definition of addition modulo n), we can derive the following version of the constraints for UNI:

$$0 \geq \max_{Rc_1} \begin{array}{|l|} \hline \{Rc_1(s)\ ,\ Rc_1(s)\} \ :\ n \\ \hline \cancel{\{\neg Rc_1(s)\ ,\ Rc_1(s)\}} \ :\ \cancel{n-1} \\ \hline \cancel{\{Rc_1(s)\ ,\ \neg Rc_1(s)\}} \ :\ \cancel{n-1} \\ \hline \{\} \ :\ n-2 \\ \hline \end{array}$$

$$\geq \boxed{\{\} : n} \tag{65}$$

7.1.3 Bidirectional Ring

While we have not worked it out here, the solution to the bidirectional ring network (BI) in Figure 1(b) will be very similar to that for UNI due to the similar linear network structure.

7.1.4 Star Network

For the star network in Figure 1(c), there are 3 legs of length m and one central hub. Thus the constraint will work out as follows (note that Rc_{m+1} does not exist in the Star network and has been removed):

$$0 \geq max_{Rc_0,Rc_1} \Bigg(\begin{array}{|l|} \hline \{Rc_1^1(s)\} : m \\ \hline \{\} : m-1 \\ \hline \end{array} \oplus \begin{array}{|l|} \hline \{Rc_1^2(s)\} : m \\ \hline \{\} : m-1 \\ \hline \end{array} \oplus \begin{array}{|l|} \hline \{Rc_1^3(s)\} : m \\ \hline \{\} : m-1 \\ \hline \end{array}$$

$$\oplus \begin{array}{|ll|} \hline \{Rc_0(s)\ ,\ Rc_1^1(s)\ ,\ Rc_1^2(s)\ ,\ Rc_1^3(s)\} & : v_1 \\ \hline \{Rc_0(s)\ ,\ Rc_1^1(s)\ ,\ Rc_1^2(s)\ ,\ \neg Rc_1^3(s)\} & : v_2 \\ \hline \{Rc_0(s)\ ,\ Rc_1^1(s)\ ,\ \neg Rc_1^2(s)\ ,\ Rc_1^3(s)\} & : v_3 \\ \hline \{Rc_0(s)\ ,\ Rc_1^1(s)\ ,\ \neg Rc_1^2(s)\ ,\ \neg Rc_1^3(s)\} & : v_4 \\ \hline \{Rc_0(s)\ ,\ \neg Rc_1^1(s)\ ,\ Rc_1^2(s)\ ,\ Rc_1^3(s)\} & : v_5 \\ \hline \{Rc_0(s)\ ,\ \neg Rc_1^1(s)\ ,\ Rc_1^2(s)\ ,\ \neg Rc_1^3(s)\} & : v_6 \\ \hline \{Rc_0(s)\ ,\ \neg Rc_1^1(s)\ ,\ \neg Rc_1^2(s)\ ,\ Rc_1^3(s)\} & : v_7 \\ \hline \{Rc_0(s)\ ,\ \neg Rc_1^1(s)\ ,\ \neg Rc_1^2(s)\ ,\ \neg Rc_1^3(s)\} & : v_8 \\ \hline \{\neg Rc_0(s)\ ,\ \ldots\} & : v_{\ldots} \\ \hline \{\neg Rc_0(s)\ ,\ \neg Rc_1^1(s)\ ,\ \neg Rc_1^2(s)\ ,\ \neg Rc_1^3(s)\} & : v_{16} \\ \hline \end{array} \Bigg) \tag{66}$$

Of course, once we know the exact values v_i for the particular constraint, we can explicitly solve this max for a closed-form result.

7.2 Solving for the Maximally Violated Constraint

Now we have the constraints in the following closed form:

$$0 \geq max_s \left(case_1(m, s) \oplus \ldots \oplus case_p(m, s) \right) \tag{67}$$

Here, we note that the case statements are parameterized by m (a parameter referring to the size of the network). We have two options for how to deal with m:

1. *Make m an explicit variable and maximize w.r.t. it.* If we do this then we will effectively get a quadratically constrained program (QCP), which could be potentially hard to solve using off-the-shelf software.

2. *Choose an explicit m to compute the constraints for.* If we do this, then we are effectively optimizing our approximation for a specific network size while obtaining an exact linear version of the constraints for this network size. It is important to note that even though we are committing a network size, the solution time and space complexity are still independent of the value of m because it is just a constant to be instantiated.

8 First-order Approximate Policy Iteration

Back in Equation 52, we determined how to perform the maximization portion of symbolic dynamic programming. Implicit in this max was also a policy: For every maximizing partition, we need only record the action that corresponds to the max value taken for that partition.

For our algorithms, it is useful to define a set of case statements for each stochastic joint action $u \in U$ that is satisfied only in the world states where u should be applied according to $\pi Case(s)$. Consequently, we recall the defininition of an action restricted policy $\pi Case_u(s)$ [4]:

$$\pi Case_u(s) = \{\langle \phi, t \rangle | \langle \phi, t \rangle \in \pi Case(s) \text{ and } \langle \phi, t \rangle \rightarrow u\} \tag{68}$$

Given a representation of the policy and the redefined backup operators, the algorithm and linear program for approximate policy iteration for first-order MDPs (FOAPI) [4] generalizes directly to the case with sum aggregators. FOAPI proceeds by calculating successive iterations of weights $w_i^{(t)}$ that represent the best approximation of the fixed point value function for each policy $\pi Case^{(t)}(s)$ at iteration t. It does this by performing the following two steps at every iteration t: (1) Obtaining the policy $\pi Case(s)$ from the current value function and weights $(\bigoplus_{i=1}^{k} w_i^{(t)} \sum_c bCase_i(c,s))$ as a byproduct of performing the max in symbolic dynamic programming based on the above description, and (2) solving the following LP that determines the weights of the Bellman-error-minimizing approximate value function for policy $\pi Case(s)$:

$$\begin{aligned} &\text{Variables: } w_1^{(t+1)}, \ldots, w_k^{(t+1)} \\ &\text{Minimize: } \phi^{(t+1)} \\ &\text{Subject to: } \phi^{(t+1)} \geq \Big| \pi Case_A(s) \oplus \bigoplus_{i=1}^{k} [w_i^{(t+1)} \sum_c bCase_i(c,s)] \\ &\qquad \ominus \bigoplus_{i=1}^{k} [w_i^{(t+1)} \sum_c B^U_{\max}(bCase_i(c,s))] \Big|; \ \forall U, s \end{aligned} \tag{69}$$

We make use of the same general method described in Section 7.1 used for FOALP to generate closed-form versions of the constraints for FOAPI. This

yields a set of constraints parameterized by various network properties that we can then solve using one of the two proposed techniques in Section 7.2.

On a final note, we mention that if approximate policy iteration converges (i.e., $\vec{w}^{(i)} = \vec{w}^{(t+1)}$), then Guestrin *et al.* [7] provide the following bound on the loss of $V^{(t+1)}$ w.r.t. the optimal value function V^* (since the Bellman error is bounded by the objective $\phi^{(t+1)}$ of the optimal LP solution at iteration $t+1$):

$$\|V^* - V^{(t+1)}(s)\|_\infty \leq \frac{2\gamma\phi^{(t+1)}}{1-\gamma} \tag{70}$$

9 Empirical Results

Forthcoming.

10 Concluding Remarks

Forthcoming.

References

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

[2] Boutilier, C., Reiter, R., Price, B.: Symbolic dynamic programming for first-order MDPs. In: IJCAI-01, Seattle (2001) 690–697

[3] Sanner, S., Boutilier, C.: Approximate linear programming for first-order MDPs. In: UAI-2005, Edinburgh, Scotland (2005)

[4] Sanner, S., Boutilier, C.: Practical linear-value approximation techniques for first-order MDPs. In: UAI-2006, Boston, MA (2006)

[5] Fikes, R.E., Nilsson, N.J.: STRIPS: A new approach to the application of theorem proving to problem solving. AI Journal **2** (1971) 189–208

[6] Younes, H., Littman, M.: PPDDL: The probabilistic planning domain definition language: *http://www.cs.cmu.edu/~lorens/papers/ppddl.pdf* (2004)

[7] Guestrin, C., Koller, D., Parr, R., Venktaraman, S.: Efficient solution methods for factored MDPs. JAIR **19** (2002) 399–468

[8] Schuurmans, D., Patrascu, R.: Direct value approximation for factored MDPs. In: NIPS-2001, Vancouver (2001) 1579–1586