Lifted Forward Planning in Relational Factored Markov Decision Processes with Concurrent Actions

Florian Andreas Marwitz
University of Hamburg
Hamburg, Germany
florian.marwitz@uni-hamburg.de

Tanya Braun
University of Münster
Münster, Germany
tanya.braun@uni-muenster.de

Ralf Möller
University of Hamburg
Hamburg, Germany
ralf.moeller@uni-hamburg.de

Marcel Gehrke
University of Hamburg
Hamburg, Germany
marcel.gehrke@uni-hamburg.de

ABSTRACT

When allowing concurrent actions in Markov Decision Processes, whose state and action spaces grow exponentially in the number of objects, computing a policy becomes highly inefficient, as it requires enumerating the joint of the two spaces. For the case of indistinguishable objects, we present a first-order representation to tackle the exponential blow-up in the action and state spaces. We propose Foreplan, an efficient relational forward planner, which uses the first-order representation allowing to compute policies in space and time polynomially in the number of objects. Thus, Foreplan significantly increases the number of planning problems solvable in an exact manner in reasonable time, which we underscore with a theoretical analysis. To speed up computations even further, we also introduce an approximate version of Foreplan, including guarantees on the error. Further, we provide an empirical evaluation of both Foreplan versions, demonstrating a speedup of several orders of magnitude. For the approximate version of Foreplan, we also empirically show that the induced error is often negligible.

KEYWORDS

Planning Markov Decision Process Lifting Probabilistic Graphical Models

ACM Reference Format:
Florian Andreas Marwitz, Tanya Braun, Ralf Möller, and Marcel Gehrke. 2026. Lifted Forward Planning in Relational Factored Markov Decision Processes with Concurrent Actions. In *Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), Paphos, Cyprus, May 25 – 29, 2026*, IFAAMAS, 9 pages. https://doi.org/10.65109/GOIS4183

1 INTRODUCTION

To compute a policy for a Markov decision processes (MDPs), combinations of the state and action spaces have to be enumerated to find the best possible action for each state. With an increasing number of (indistinguishable) objects, the state space grows exponentially, leading to drastically more combinations to be enumerated. In case the actions also depend on the states or objects, the action space grows as well, leading to even more combinations to enumerate. Further, if one allows concurrent actions, all possible combinations of the actions of the already enlarged action space have to be accounted for during the enumeration, yielding an exponentially-sized action space. Therefore, given the number of combinations that have to be enumerated, concurrency is rarely modelled. However, consider the following example: A small town is haunted by an epidemic. To fight the epidemic, the town's mayor can impose travel bans on the town's citizens. Certainly, the mayor can confine all citizens to their homes, stopping the epidemic. However, the citizens' overall welfare is important as well. Therefore, the mayor is interested in the *best decision* of imposing travel bans (concurrent actions) w.r.t. confining the epidemic, while keeping the citizens' welfare above a certain threshold. Computing this problem on a propositional level, with every citizen explicitly represented, such as with current MDP approaches, blows up the state and action space, as it requires exponential enumeration of all subsets of the population. However, there are groups of citizens behaving identically w.r.t. getting sick (and well again) as well as regarding their welfare if a travel ban is imposed, i.e., these citizens are indistinguishable for the mayor. Within these groups, it does not matter on *which exact* citizen the mayor imposes a travel ban, only on *how many* she imposes a travel ban is important. Additionally, many computations over subsets of indistinguishable citizens are redundant. Thus, we propose to drastically reduce the search and action space using a first-order representation, grouping indistinguishable citizens, and use this representation to plan group-wise by reasoning about a representative and then projecting the result to the whole group. We publish a version of this paper with appendix including extended examples and with all (complete) proofs on arXiv[1].

Contribution. First, to be able to represent numerous indistinguishable objects and actions, we use probabilistic relational models as a representation in (factored) MDPs (fMDPs), which yields *relational factored MDPs* (rfMDPs). Second, as our key contribution, we propose *Foreplan* to carry out exact planning in rfMDPs. Foreplan uses our first-order representation to keep the action and state space only polynomial in the number of objects, which we prove

Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), C. Amato, L. Dennis, V. Mascardi, J. Thangarajah (eds.), May 25 – 29, 2026, Paphos, Cyprus. © 2026 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). https://doi.org/10.65109/GOIS4183

[1]https://arxiv.org/abs/2505.22147

by using our so-called *relational cost graph*. Foreplan remains exponential in the number of cliques c and size w of the largest clique in the relational cost graph, but both parameters are mostly small and independent of the domain sizes. Therefore, Foreplan is an efficient planner w.r.t the number of objects. We show a speedup of at least four orders of magnitude is achievable. Last, following the approximation ideas of Guestrin et al. [16], we can reduce the runtime even further. We propose Approximate Foreplan, whose runtime is polynomial in c unlike Foreplan. For Approximate Foreplan, we show a speedup of at least four orders of magnitude.

Related Work. Bellman [2] introduces MDPs, which Boutilier et al. [4] extend to fMDPs by factorizing the transition function. Factorizing also the value function, Guestrin et al. [16] provide two approximate algorithms for solving planning in fMDPs. Dean et al. [10] cluster the state space of fMDPs to reduce the state space even further. Givan et al. [15] group equivalent states based on the notion of bisimulation. Both approaches lack the ability to handle concurrent actions efficiently. MDPs can be generalized to partially observable MDPs, in which the agent is uncertain in which state the environment currently is [19]. Sanner and Kersting [29] add the first-order perspective to partially observable MDPs, but do not consider concurrent actions. Bernstein et al. [3] extend partially observable MDPs to have a set of decentralized agents. Braun et al. [6] group indistinguishable agents. This is similar to our approach, in which we handle sets of indistinguishable state variables. However, Braun et al. [6] only solve the problem inefficiently via brute-force.

The idea of lifting is to carry out computations over representatives for groups of indistinguishable random variables [9, 20, 25]. There are online decision making approaches adding action and utility nodes to this representation [1, 13, 14], here, we focus on offline planning. To carry out even more lifted computations, Taghipour [30] extends lifted probabilistic inference by a generalized counting framework, which we extend later on. Using a first-order representation for states, actions and objects, Boutilier et al. [5] exploit the relational structure for planning using MDPs, still without concurrent actions. To specify factored transition models in first-order MDPs, Sanner and Boutilier [28] introduce factored first-order MDPs using a backward search, and give only an approximate algorithm without providing error bounds. In contrast, we propose an exact algorithm by applying lifting. Later, we also introduce an approximate version with error estimation. Moreover, we prove on which models our algorithm runs efficiently. For a survey on planning using first-order representations, we refer to Corrêa and De Giacomo [7].

Structure. The remainder of this paper is structured as follows: First, we present preliminaries for (factored) MDPs, including lifting. Then, we introduce rfMDPs to group indistinguishable objects. Afterwards, we propose Foreplan, which exploits these indistinguishable objects using a compact state representation, to efficiently support decision making with concurrent actions. Further, we provide a theoretical analysis and empirical evaluation of Foreplan.

2 PRELIMINARIES

In this section, we lay the foundation for rfMDPs. We first recap (f)MDPs, which model probabilistic state changes occurring due to agents performing actions. Furthermore, states have a reward assigned, and the task is to compute the optimal policy, i.e., which action to perform in which state. Second, we recap lifting in probabilistic graphical models.

2.1 (Factored) Markov Decision Processes

In this subsection, we first define MDPs and specialize them to fMDPs by factoring the transition function T.

Definition 2.1 (Markov Decision Process). A *Markov Decision Process* is a tuple $(\mathbf{S}, \mathbf{A}, T, R)$ with a finite set of states $\mathbf{S}$, a finite set of actions $\mathbf{A}$, a reward function $R : \mathbf{S} \mapsto \mathbb{R}$ and a transition function $T : \mathbf{S} \times \mathbf{A} \times \mathbf{S} \to [0, 1]$. The value $T(s, a, s')$ is the probability $P(s' \mid s, a)$ of transitioning from state $s \in \mathbf{S}$ to state $s' \in \mathbf{S}$ if action $a \in \mathbf{A}$ is performed.

The rewards are additive, possibly discounted by a factor $\gamma \in [0, 1)$. An MDP is fully observable and has the first order Markov property, i.e., the probability of the next state only depends on the current state and action. Let us have a look at a simple, yet incomplete example of an MDP.

Example 2.2. Suppose we have the states *healthy* and *sick*. Our agent has two possible actions: *Travelling* or *staying at home*. When the agent is in state sick and travels, she stays sick with a probability of 0.9. The agent obtains a reward of -1 if sick and 1 if healthy.

Planning in MDPs refers to calculating an optimal policy, which is a mapping from each state to an action to perform for the agent. To compute such a policy, we first define the *utility* of a state:

Definition 2.3 (Bellman Equation [2]). The utility of a state s is given by

$$U(s) = R(s) + \gamma \max_{a \in \mathbf{A}} \sum_{s' \in \mathbf{S}} P(s' \mid s, a) \cdot U(s'). \tag{1}$$

To find the utility of a state algorithmically, we determine a *value* function V satisfying the Bellman equation. The value function induces a policy by selecting the action that yields the maximum expected value. For computing a value function, we can use a linear programming formulation [12, 26]:

$$\begin{aligned}
&\text{Variables:} && V(s) \quad \forall s \in \mathbf{S}\,;\\
&\text{Minimize:} && \sum_{s \in \mathbf{S}} \alpha(s) V(s)\,;\\
&\text{Subject to:} && \forall s \in \mathbf{S},\ \forall a \in \mathbf{A}:\\
& && V(s) \geq R(s) + \gamma \sum_{s' \in \mathbf{S}} P(s' \mid s, a) \cdot V(s'),
\end{aligned} \tag{2}$$

where the coefficients $\alpha(s)$ are arbitrary positive numbers, e.g., a uniform distribution over all states [26]. Planning in MDPs can be solved in polynomial time w.r.t to the state space size [24]. But what if the state space becomes very large, e.g., exponential in the number of objects? For retaining an efficient transition model, fMDPs make use of state variables for the objects. The state space is then spanned by the state variables. For simplicity, all state variables are Boolean, but can be easily extended to non-Boolean.

Definition 2.4 (Factored MDP). A *factored MDP* is a tuple $(\mathbf{S}, \mathbf{A}, T, R)$, where the state space is the cross product $\mathbf{S} = \{0, 1\}^m$ of the subspaces corresponding´to the state variables $S_1, \ldots, S_m$ with Boolean range. The transition function T is, for each action, factored:

$$P(S' \mid S, a) = \prod_{i=1}^{m} P(S'_i \mid Pa(S'_i), a), \tag{3}$$

where $Pa(S'_i) \subseteq \mathbf{S}$ denotes the set of *parents* of S'_i in the previous state, S the old state, S' the new state and S'_i the i-th state variables in the respective state. For a given state $s \in \mathbf{S}$, we denote with s_i the assignment of state variable S_i in state s.

For large m, the state space explodes. But when these objects are indistinguishable, we can use parameterized graphical models to encode state spaces and transition models in a compact way.

2.2 Parameterized Graphical Models

Having *indistinguishable* random variables leads to redundant computations. We can tackle redundant computations by parameterizing our probabilistic model and using representatives of groups of indistinguishable variables, so that inference in the probabilistic model becomes tractable w.r.t. domain sizes using representatives during calculations [23]. We first define *parameterized random variables* (PRVs) to group same behaving variables:

Definition 2.5 (Parameterized Random Variable [31]). Let $\mathbf{W}$ be a set of random variable names, $\mathbf{L}$ a set of logical variable (logvar) names, and $\mathbf{D}$ a set of constants (universe). All sets are finite. Each logvar L has a domain $\mathcal{D}(L) \subseteq \mathbf{D}$. A *constraint* C is a tuple $(\mathcal{X}, C_{\mathcal{X}})$ of a sequence of logvars $\mathcal{X} = (X_1, \dots, X_n)$ and a set $C_{\mathcal{X}} \subseteq \times_{i=1}^{n} \mathcal{D}(X_i)$. The symbol $\top$ for C marks that no restrictions apply, i.e., $C_{\mathcal{X}} = \times_{i=1}^{n} \mathcal{D}(X_i)$. A *PRV* $B(L_1, \dots, L_n), n \geq 0$, is a syntactical construct of a random variable $B \in \mathbf{W}$ possibly combined with logvars $L_1, \dots, L_n \in \mathbf{L}$. If $n = 0$, the PRV is parameterless and constitutes a propositional random variable (RV). The term $\mathcal{R}(B)$ denotes the possible values (range) of a PRV B. An *event* $B = b$ denotes the occurrence of PRV B with range value $b \in \mathcal{R}(B)$.

We create PRVs for the health status of each person in the town:

Example 2.6. Let $\mathbf{W} = \{Sick, Epidemic\}$, $\mathbf{L} = \{M\}$, $\mathcal{D}(M) = \mathbf{D} = \{a, b, c, d, e, f, g, h\}$ with Boolean-valued PRVs $Sick(M)$ and $Epidemic$. $Epidemic$ is a parameterless PRV.

To relate PRVs, we use *parameterized factors*:

Definition 2.7 (Parfactor model [31]). Let Φ be a set of factor names. We denote a *parameterized factor* (*parfactor*) g by $\phi(\mathcal{B})_{|C}$ with $\mathcal{B} = (B_1, \dots, B_n)$ a sequence of PRVs, $\phi : \times_{i=1}^{n} \mathcal{R}(B_i) \to \mathbb{R}^+$ a *potential function* with name $\phi \in \Phi$, and C a constraint on the logvars of $\mathcal{B}$. A set of parfactors forms a *model* $G := \{g_i\}_{i=1}^{n}$. With Z as normalizing constant, G represents the full joint distribution $P_G = \frac{1}{Z} \prod_{f \in gr(G)} f$, with $gr(G)$ referring to the groundings of G w.r.t. given constraints. A grounding is the instantiation of each logvar in each parfactor with an allowed constant.

Continuing Example 2.6, let us define the parfactor model:

Example 2.8. Let $\forall m \in M : \phi(Sick(m), Epidemic)_{\top}$ be the parfactor with potential ϕ defining the probability to be sick for all persons from $\mathbf{D}$, given there is an epidemic (or not). The grounded model then consists of the eight factors $\phi(Sick(a), Epidemic), \dots, \phi(Sick(h), Epidemic)$ compared to one parfactor in the lifted model.

Next, we integrate parfactors into fMDPs to achieve a compact representation of state and action spaces.

3 RELATIONAL FACTORED MDPS

Now, we present the first-order representation for the state and the action space as well as the transition function, which allows to efficiently represent indistinguishable objects as well as concurrent action, which then in turn can be exploited by an appropriate algorithm. To that end, we present Foreplan in the next section.

We define rfMDPs based on fMDPs, but include a parfactor model inside the state and action space and transition function. That is, the set of state variables can now contain PRVs, which are then used in the transition model and the reward function. Also, we have *action PRVs*, whose value is chosen by the agent. Basically, we keep the same semantics as in Definition 2.1 and only change the representation to exploit indistinguishable variables. In the following definition we use the term *interpretation* of a set for a truth-value assignment to each element in the set.

Definition 3.1 (Relational Factored MDPs). A *relational factored MDP* is a tuple $(\mathbf{D}, \mathbf{L}, \mathbf{B}, \mathbf{A}, G, \mathbf{R})$. The set $\mathbf{D}$ is a set of constants and the set $\mathbf{L}$ is a set of logvars over $\mathbf{D}$. The set $\mathbf{B}$ is a set of PRVs defined over $\mathbf{L}$. The set of possible interpretations $\mathcal{I}_{\mathbf{B}}$ for the groundings of the set $\mathbf{B}$ defines the state space, i.e., all possible assignments. The set $\mathbf{A}$ is a set of action PRVs. A parfactor model G over $\mathbf{A}$ and $\mathbf{B}$ represents the transition function $T : \mathcal{I}_{\mathbf{B}} \times \mathcal{I}_{\mathbf{A}} \times \mathcal{I}_{\mathbf{B}} \to \mathbb{R}_0^+$, with the set $\mathcal{I}_{\mathbf{A}}$ of possible interpretations of the groundings of $\mathbf{A}$, and specifies the transition probability given an action and a previous state. The set $\mathbf{R}$ contains *parameterized* local reward functions $R_i : \bigtimes_j \mathcal{R}(B_{i,j}) \to \mathbb{R}$, defined over all PRVs $B_{i,j}$, which contribute to the local reward R_i. The reward function R is decomposed as a sum operation over $\mathbf{R}$.

Let us generalize Example 2.2 to an arbitrary number of persons behaving in the same way as the agent in Example 2.2:

Example 3.2 (Epidemic). There is a set $\mathbf{D}$ of persons living in a small town, represented by the logvar M with $\mathcal{D}(M) = \mathbf{D}$. Each person can be sick or healthy, leading to the PRV $Sick(M)$. The government gets a reward of 1 for each healthy person and a reward of -1 for each sick person. To combat an epidemic, the government can impose travel bans on persons, resulting in the action $Restrict(M)$ to impose a travel ban on a subset of persons. Moreover, each person can travel or not, leading to the PRV $Travel(M)$. The government gets a reward of 2 for each person travelling. The PRV $Epidemic$ is influenced by the number of people travelling and influences the healthiness of each person. Figure 1 shows the (lifted) transition model for this example.

Since an action can be applied to each person concurrently, the amount of possible actions is exponential due to the power set, i.e., all possible combinations for all persons. Foreplan avaoids the exponential blow-up by exploiting the indistinguishability of action PRVs.

With Definition 3.1, we require the rewards to be represented as a sum of local reward functions, and not as some arbitrary function. Actually, this is too restrictive: It is sufficient that an operation running over same behaving variables has a hyperoperation repeating the operation multiple times. For a sum, the hyperoperation is multiplication. Also, multiple terms of such operations can be linked together in any way. For simplicity and the proofs, we use the definition as it is throughout the paper.

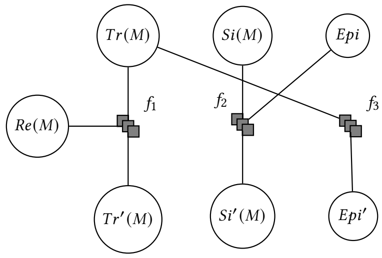

Figure 1: Lifted representation of the transition model for Example 3.2. We abbreviate by using only the first letter(s) for each symbol.

We exploit the symmetries in Definition 3.1 with Foreplan in the next section. In the remainder of this section, we explain what we mean by *action PRVs* and by *parameterized* local reward functions.

3.1 Parameterizing Actions

In our epidemic example, the mayor, representing the government, can impose travel bans on all parts of the town's population. We extend the action definition in this subsection to account for groups of objects. Having groups, we circumvent enumerating of all possible subsets and model the example action of imposing travel bans on a subset of the population efficiently.

Definition 3.3 (Action PRV). An action PRV A is a Boolean-valued PRV. A concrete action is a set of events, i.e., each grounding of A receives an assignment $a \in \mathcal{R}(A)$.

Action PRVs allow for a more general action setting. When writing action, we refer to Definition 3.3. In our example, the mayor can restrict multiple persons from travelling at once:

Example 3.4 (Action PRV). The action PRV $Restrict(M)$ models the possible travel bans on the population of the town. For a concrete action, the mayor has to specify which persons are restricted from travelling. When restricting the persons a, b and f, the mayor specifies $Restrict(M) = true$ with constraint $(M, \{a, b, e\})$ and $Restrict(M) = false$ with constraint $(M, \{c, d, f, g, h\})$.

However, when the persons a to h are indistinguishable, it is irrelevant which selection is used for the constraints, only the amounts are relevant. We describe the impact of parameterization on the rewards next.

3.2 Parameterized Local Reward Functions

The reward function in fMDPs maps from the (joint) state to the reward of the state. For evaluating the reward function, we thus have to construct the joint state and cannot exploit the factorization, breaking our aim of compact representation. To further use our compact representation, we introduce a decomposable reward function: For simplicity, we assume that the reward function is factored as $R = \sum_i R_i$, with local reward functions R_i with scope restricted to a subset of the state variables. Other operations are possible too, as long as they have a hyperoperation, as already mentioned. As we have indistinguishable state variables, we reduce redundant computations by using representatives in the reward functions analogously to a parfactor, but using a sum instead of a product:

Definition 3.5. A local reward function $R_i : \bigtimes_j \mathcal{R}(B_{i,j}) \to \mathbb{R}$ is defined over all PRVs $B_{i,j}$, which contribute to the local reward R_i. The semantics of a single parameterized local reward function R_i is defined as the sum $\sum_z R_i(z)$ over the interpretations $z \in \bigtimes_j \mathcal{R}(B_{i,j})$ in the current state of all groundings of R_i.

In other words, a parameterized local reward function serves as a placeholder for the set of local reward functions, obtained by replacing all logvars by their possible instantiations. We illustrate parameterized reward functions in our epidemic example:

Example 3.6. The parameterized local reward functions for Example 3.2 are $R_1(Sick(M))$, evaluating to -1 (1) for each person (not) being sick, and $R_2(Travel(M))$, evaluating to 2 for each person travelling. If five persons are sick, three are not sick and four people are travelling, the total reward is $-5 + 3 + 8 = 6$.

With rfMDPs, we can efficiently represent the action and state space as well as the transition function for numerous indistinguishable objects. Further, we have a compact representation for concurrent actions. In the next section, we propose Foreplan, our main contribution, which uses our representation to solve planning in rfMDPs in time polynomial in the number of objects.

4 FOREPLAN: EFFICIENT FORWARD PLANNING

In this section, we propose Foreplan, our exact forward relational planner for rfMDPs. The input to Foreplan is an rfMDP. The output is a value function, which induces a policy. Foreplan computes the value function using a compact state representation for the rfMDP to exploit the indistinguishability of the objects and then running a linear program based on the state representation to calculate the value function.

The first-order representation in rfMDP contains dependencies between PRVs. For using the representation in computation, we first need to recognize the dependencies and find a suitable representation of them to compute the value function efficiently. We first describe how Foreplan recognizes the dependencies and identifies a suitable representation. Afterwards, we outline how Foreplan uses the identified representation to compute a value function.

4.1 Obtaining a Compact State Representation

Foreplan needs to encode the current state compactly to efficiently reason about indistinguishable variables. Thus, in this subsection, we describe how Foreplan treats indistinguishable objects for computations using rfMDPs. Foreplan does not need to keep track of objects that could be differentiated by their history. Rather, with each action and new time step, the history is swept away and the objects remain indistinguishable because of the first-order Markov assumption. To compactly represent the state, the basic idea is to *count* the objects for which the PRVs are true or false, using the idea of Counting Random Variables (CRVs) [30]. When evaluating a parfactor, the joint assignment to the PRVs is needed, rather than individual counts for each PRV. It is sufficient to count the number of occurrences of each possible truth-value assignment to the groundings of the *input* PRVs of a parfactor in a histogram. The *input* PRVs of a parfactor are the ones representing the current state. Counting only the input PRVs is sufficient, because all possible next states are iterated separately later. But some parfactors may be defined over PRVs defined over different logvars, like f_3 in Figure 1 is defined over $Travel(M)$ and $Epidemic$. Given the counts for $Travel(M)$ and $Epidemic$ separately is enough, since these PRVs do not depend on each other. But how does Foreplan recognize which PRVs depend on which PRVs? For answering this question, we propose the *relational cost graph* to identify the dependencies between PRVs and identify the groups that need to be counted jointly. Focusing only on the counts for the groups enables Foreplan to use a much simpler state space representation, namely the set of possible histograms, compared to the grounded state space representation, which is the Cartesian product over the domains of all state variables. We now describe how to count the assignments in more detail.

Counting the assignments of each PRV separately is insufficient, as PRVs can be defined over the same logvars and thus interfere with each other. However, the parfactors can be evaluated separately since, in Equation 2, we have the full current and next state available. Thus, it is sufficient to count PRVs together if they share a logvar and occur together in a parfactor. To obtain the representation and later on quantify its complexity, we define the relational cost graph:

Definition 4.1 (Relational Cost Graph). The *relational cost graph* of an rfMDP has a vertex for each PRV in the current state. Two vertices are connected by an edge if and only if the PRVs associated with these two vertices share a logvar and occur together in a parfactor as input PRVs or a parameterized local reward function. We denote the number of (maximal) cliques by c and the size of the largest clique by w.

Let us take a look at the relational cost graph of Example 3.2:

Example 4.2. The relational cost graph for Example 3.2 consists of three isolated vertices corresponding to $Sick(M)$, $Travel(M)$ and $Epidemic$. The first two do not occur together in a parfactor or local reward function and both do not share a logvar with the last one.

The key insight now is that (maximal) cliques in the relational cost graph correspond to sets of PRVs that Foreplan needs to count together as they interfere with each other. For one logvar per PRV, basic CRVs [30] already return the result. To lift the limitation of one logvar per PRV, we extend the definition by Taghipour [30]:

Definition 4.3 (Extended Counting Random Variable). A counting formula $\gamma = \#_C[B_1, \dots, B_k]$ is defined over PRVs B_i with a constraint $C = (\mathcal{L}, C_{\mathcal{L}})$ over the logvars $\mathcal{L}$ of the PRVs B_i. The counting formula represents a *counting random variable* (CRV) whose range is the set of possible histograms that distribute $n = |C_{\mathcal{L}}|$ elements into $\prod_{i=1}^{k} |\mathcal{R}(B_i)|$ buckets. The state of γ is the histogram function $h = \{(r_i, n_i)\}_{i=1}^{r}$ stating for each $r_i \in \times_{i=1}^{k} \mathcal{R}(B_i)$ the number n_i of tuples $(B_i)_i$ whose state is r_i.

If no restrictions apply, we omit C, and $n = \prod_{L \in \mathbf{L}} |\mathcal{D}(L)|$, where $\mathbf{L}$ is the set of logvars of the PRVs B_i. We do not define the operations to manipulate CRVs as we do not need them in this paper. The CRV corresponding to a clique gives us the number of occurrences for each possible instantiation of the PRVs in that clique. For illustrative purposes, we give a small example for a CRV using an additional PRV $Friends(M, Y)$, with M and Y having the same domain:

Example 4.4. To have a PRV with two logvars, assume we have three PRVs, $Travel(M)$, $Friends(M, Y)$ and $Sick(Y)$ with $|\mathcal{D}(\mathcal{M})| = |\mathcal{D}(Y)| = 8$. A possible state for the CRV $\#[Travel(M), Friends(M, Y), Sick(Y)]$ is $\{(ttt, 6), (ttf, 30), (tft, 4), (tff, 0), (ftt, 6), (ftf, 0), (fft, 0), (fff, 18)\}$ and we have $n = 64$. The first histogram entry shows that, in this state, there are six tuples for which $Travel(m) = t$, $Friends(m, y) = t$, and $Sick(y) = t$ holds.

Example 4.4 illustrates that we need only eight buckets in a histogram regardless of the domain size of M and Y. The number of buckets depends on the range values of the PRVs and not on the domain sizes. We formalize the state representation, which is a set of CRVs per (maximal) clique in the relational cost graph:

Definition 4.5 (State Representation). For counting the corresponding PRVs in a clique, we create one CRV S_i for each (maximal) clique in the relational cost graph. For a single propositional RV in the relational cost graph, we use the RV for S_i instead of a CRV. A *state* assigns a value to each S_i. The resulting *state space* is the set of possible states. We denote the *state space* by the tuple $(S_i)_i$, which contains one CRV (or RV) per clique.

We give the representation of the state space for Example 3.2:

Example 4.6. As the vertices are not connected in the relational cost graph, so the state representation is $(\#[Sick(M)], \#[Travel(M)], Epidemic)$.

The relational cost graph in Example 4.2 tells us that we can count the number of healthy persons separately from the number of travelling persons. In particular, Foreplan does not store which sick persons are travelling, since this information is irrelevant. We prove that our state representation is correct, i.e., retains the same semantics, namely exactly representing $\mathbf{S}$ of a ground MDP:

Theorem 4.7. *The representation in Definition 4.5 is correct.*

Proof Sketch. Given groundings for the state PRVs, we derive the histograms for the CRVs by counting the assignments for each parfactor. Given a representation as in Definition 4.5, we reconstruct, for each parfactor, the groundings of the PRVs by using the counts from the CRVs to instantiate the respective parfactor. □

To advance through an action to the next state, the action now has to use the same state representation, i.e., the action is specified on the counts for all PRVs of the current state for all parfactors the action is mentioned in:

Example 4.8. The action $Restrict(M)$ uses the PRV $Travel(M)$. Thus, the mayor needs to specify how many persons of those (not) travelling are (not) allowed to travel. Suppose that in the current state five out of eight persons are travelling and two persons are sick. The action $Restrict(M)$ is defined over $\#[Travel(M), Restrict(M)]$. A concrete action is, e.g., $a = \{(tt, 3), (ft, 2)\}$, which means that three persons currently travelling are restricted from travelling and two persons not travelling. The action a does not need to specify the counts of people no travel ban is imposed on (tf, ff), as these are determined by a and the current state.

The mayor no longer needs to specify individual persons (c.f. Example 3.4), but rather the number of persons (not) travelling, which are restricted from travelling. It is irrelevant on which exact persons the action is performed. With this action representation, we reduce the action space from exponential to polynomial, which we prove in Theorem 5.2 in the next section.

In the next subsection, we show how Foreplan uses the action space to compute the value function by solving a linear program.

4.2 Computing the Value Function

Let us have a look on how Foreplan computes the value function based on the introduced state representation. Foreplan uses the linear programming formulation given in Equation 2 to compute the value function. For the linear program, Foreplan uses the introduced state and action representations to iterate over all states and actions.

For instantiating the linear program in Equation 2, we next describe how Foreplan calculates the transition probability. Since we have full evidence provided, Foreplan evaluates each state CRV S'_i separately. For a fixed state CRV S'_i, the value s'_i is fixed since the whole state space is iterated. For evaluating $P(s'_i \mid s, a)$, Foreplan iterates over all possible assignment transitions, e.g., from each bucket in the histogram s to each bucket in the histogram s'. In our epidemic example, this is, e.g., the number of people getting sick and healthy. For each possible assignment transition, Foreplan calculates the transition probability: The probability for a representative transition is given by the involved parfactors, i.e., the parfactors defined over the PRVs mentioned in S'_i. But this representative probability has to be weighted by how many times this assignment transition is applicable using the multinomial coefficient. Take for example the state in which five persons are sick. When three sick persons get healthy, this assignment transition has weight $\binom{5}{3}$ as any three of the five persons can get healthy. The final probability of $P(s'_i \mid s, a) = \sum_t w_t p_t$ for a fixed CRV is the sum over all possible assignment transitions t, each with representative probability p_t and weight w_t. Since the state space is factored, the full transition probability $P(s' \mid s, a)$ is factored as $P(s' \mid s, a) = \prod_i P(s'_i \mid s_i, a)$.

Looking at Foreplan's linear program, there is one constraint added per state and action combination. Thus, Foreplan uses both spaces for generating only the necessary constraints: When an initial state is given, the reachable state space can be pruned. Also, additional checks like mutual exclusion, capacities or other constraints can be added, either on PRV level or on search space level.

With Foreplan, we are able to cope with numerous indistinct objects and actions on collections of those objects. We do so by successfully applying lifting in the field of MDPs. While traditional approaches can represent actions on sets of objects, they fail to do so efficiently. Therein, the actions for each subset would be represented on their own, resulting in exponentially many actions. In the next section, we show the complexity of Foreplan.

5 COMPLEXITY ANALYSIS OF FOREPLAN

Having outlined Foreplan, we analyze the complexity of Foreplan in this section. We start by quantifying the state representation and using the complexity of the state representation to derive the runtime complexity of Foreplan.

We derive the following theorem about the size of the state representation from Definition 4.5.

Theorem 5.1. *The state representation is in $O(c \times 2^w)$.*

Proof. For each clique in the relational cost graph, the size of the histogram function is exponential in the number of vertices in the clique, as we enumerate all possible assignments. Thus, the size of the state representation is bounded by $c \times 2^w$. □

Note that c and w are independent of the domain sizes and determined only by the structure of the relational cost graph, and thus in general small. Also w is bounded by the number of PRVs and c by the number of parfactors in the parfactor model. Theorem 5.1 overapproximates the size of the state representation, as not all cliques have the same size and not both, c and w are large at the same time. Building on the size of the state representation, we give the complexity of the state and action space:

Theorem 5.2. *The state and action spaces are both polynomial in the number of objects and exponential in c and w.*

Proof. We need to iterate over all possible instantiations of the state representation. For each clique (resp. CRV), the number of possible instantiations is polynomial in the number of objects and exponential in w. The joint state requires one instantiation per clique, resulting in the number c of CRVs as exponent. The size of the action space is bounded by the size of the state space, as the action has to specify a (subset of a) state. □

Since Foreplan uses a linear program to compute the value function, we analyze the complexity of solving the linear program Foreplan builds. Linear programs can be solved in polynomial time w.r.t the variables and constraints [33]. Let us therefore take a closer look at the number of constraints and variables Foreplan generates:

Theorem 5.3. *The number of linear programming constraints and variables Foreplan creates are polynomial in the state space.*

Proof. By Eq. 2, Foreplan generates one variable per possible state and one constraint for each state and action combination. □

Plugging Theorem 5.2 into Theorem 5.3 leads to:

Theorem 5.4. *The runtime of Foreplan is polynomial in the number of objects and exponential in c and w.*

Therefore, with Foreplan, we have introduced an exact planner with a runtime polynomial instead of exponential w.r.t domain sizes for concurrent actions. Thus, we have already achieved an exponential speed up.

Our main contribution, Foreplan, significantly advances the state of the art. However, as it is an exact algorithm, the value function is still not factorized. In case the runtime is of utter importance, we can also approximate the value function, following the idea of Guestrin et al. [16], which has also been picked up by, e.g., Sanner and Boutilier [27]. By approximating the value function, we can prevent iterating over the joint state space leading to a blazingly fast, but approximate, version of Foreplan.

6 FOREPLAN: EVEN FASTER BY APPROXIMATION

While Foreplan runs in time polynomial in the number of objects, the runtime still exponentially depends on c. In this section, we present an approximation technique inspired by the Approximate Linear Programming (ALP) approach [16] to prevent iterating the whole state space. Our approach follows the same idea as ALP and ALP for first-order MDPs [27]. We first describe the approximation idea and then how Foreplan uses the approximation for our new representation and for concurrent actions. Last, we give bounds on the runtime and on the approximation quality. We call the approximate version *Approximate Foreplan*.

Foreplan needs to iterate over the whole state space, because the value function maps from a state to the value of that state. We approximate the value function by a set h_i of basis functions, whose scope is a subset of S, $V \approx \sum_i w_i h_i$, where the goal is to find the most suitable weights w_i [16]. Approximate Foreplan also needs the value of all possible next states in terms of the same approximation. Thus, Approximate Foreplan uses *backprojections* g_i^a of the basis functions h_i [16], stating the influence of x on the next state:

$$g_i^a(x) = \sum_{x'} P(x' \mid x, a) \cdot h_i(x') \tag{4}$$

To compute the basis functions and backprojections in a lifted way, we parameterize the basis functions analogously to the reward functions and calculate them in the same way. The basis functions should capture the important dynamics in the model [21], most importantly the rewards [27]. As proposed by Koller and Parr [21] and picked up in Sanner and Boutilier [27], we also use basis functions for capturing the rewards alongside a constant basis function:

Example 6.1 (Basis Function). We have three basis functions: $h_0 := 1$, $h_1(Sick(M)) := R_1(Sick(M))$ as well as $h_2(Travel(M)) := R_2(Travel(M))$.

The backprojections are computed in a lifted way:

Definition 6.2 (Lifted Backprojection). Given a basis function h_i and Boolean assignments $\tilde{x}$ and $\tilde{a}$ to the state and action, respectively, the backprojection is defined as $g_i^{\tilde{a}}(\tilde{x}) = \sum_{\tilde{x}'} P(\tilde{x}' \mid \tilde{x}, \tilde{a}) \cdot h_i(\tilde{x}')$. The *lifted backprojection* $G_i^a(x)$ for a state x and action a then sums $g_i^{\tilde{a}}(\tilde{x})$ for each possible propositional assignment $\tilde{x}$ and $\tilde{a}$ and weights the term with the counts given by the state x.

Let us apply the backprojection in our running example:

Example 6.3 (Lifted Backprojection). Suppose we have three sick persons and two healthy ones and are interested in the backprojection of h_1. Then, we have $G_1([(t, 3), (f, 2)], epi) = 3 \cdot g_1(true, epi) + 2 \cdot g_1(false, epi)$.

Approximate Foreplan precomputes all backprojections and then builds the following linear program [16], whose result is the approximated value function and thus constitutes the result:

$$\begin{aligned} &\text{Variables:} && w_1, \ldots, w_n\ ; \\ &\text{Minimize:} && \sum_{i=1}^{n} \alpha_i w_i\ ; \\ &\text{Subject to:} && \forall a \in A: \\ & && 0 \geq \max_x \left\{ R(x) + \sum_{i=1}^{n} w_i \left(\gamma G_i^a(x) - h_i(x) \right) \right\}. \end{aligned} \tag{5}$$

The α_i's are effectively coefficients for a linear combination over the w_i, stating how important the minimization of each w_i is [8, 16]. The maximum operator is no operator in linear programs and is removed in an operation similar to variable elimination (VE) [35].

For the runtime analysis, we introduce the *cost network* briefly: The cost network for a constraint has a vertex for each appearing variable and there is an edge in the cost network between two vertices if the corresponding variables appear together in the same reward or basis function, or backprojection. For the complexity analysis, we use the structural graph parameter *induced width*, which provides an upper bound for the largest intermediate result [11]:

Theorem 6.4. *Approximate Foreplan runs in time polynomial in the number of objects, polynomial in c and exponential in the induced width of each cost network, when w is bounded.*

Proof Sketch. Approximate Foreplan has to precompute the backprojections and solve the linear program in Equation 5. The first involves constantly many iterations of the state and action spaces. The second is linear in the action space and exponential in the induced width of each cost network [11, 16]. Because Approximate Foreplan does not iterate over the whole state space, but treats each clique independently in the maximum operator, the effective state space is no longer exponential in c, but polynomial, and the action space is bound by the state space. □

Most notably, w and induced width in Theorem 6.4 are mostly small and fixed, leading to a polynomial runtime, as the growth in the number of objects is of more interest. Combining the relational cost graph and the cost networks in a single *total relational cost graph*, one can show that the runtime of Approximate Foreplan is polynomial in the number of objects when the treewidth of the total relational cost graph is bounded.

Moreover, we can show that Approximate Foreplan and ALP compute the same solutions:

Theorem 6.5. *Given an rfMDP R, Approximate Foreplan and ALP are equivalent on R and the grounded version of R, respectively.*

Proof Sketch. With appropriate α_i, the objective function carries out lifted computation of the grounded basis functions. Each constraint is correct, because the lifted backprojections and lifted basis functions compute the same value as for the grounded model. The action representation covers the whole action space. □

With ALP and Approximate Foreplan being equivalent, we transfer the approximation guarantee for ALP to Approximate Foreplan:

Corollary 6.6 (Approximation Guarantee [8, 16]). *Approximate Foreplan provides the best approximation of the optimal value function in a weighted $\mathcal{L}_1$ sense, where the weights in the $\mathcal{L}_1$ norm are the state relevance weights α.*

Proof. Since the claim holds for ALP [8, 16], the proof follows by the equivalence of ALP and Approximate Foreplan. □

With Approximate Foreplan, we reduce the runtime further from exponential in c to polynomial in c. In the next section, we evaluate the runtimes of Foreplan and Approximate Foreplan empirically.

7 EMPIRICAL EVALUATION

(Approximate) Foreplan runs in time polynomial in the number of objects, but other terms are unavoidably exponential. In contrast to current approaches, the exponential terms of both Foreplan variants depend only on the structure of the rfMDP and not on the number of objects. To underline our theoretical results, we evaluate (Approximate) Foreplan against ALP and an implementation of symbolic value iteration (VI) using extended algebraic decision diagrams (XADDs) [18, 32] for the epidemic example introduced in Example 3.2 as well as for the BoxWorld [5] and a fully-connected SysAdmin [16] example, with the latter two extended to concurrent actions. We also assess the quality of the policy given by Approximate Foreplan. We use Python 3.12 and HiGHS for solving the linear programs [17]. We run all implementations on an AMD EPYC 7452 32-Core Processor with 504 GB of RAM.

Figure 2 shows the runtime for the epidemic example for (Approximate) Foreplan, ALP and XADD Symbolic VI for up to 191 persons with a time limit of two hours. XADD Symbolic VI exceeds the time limit after ten persons, ALP after 15 persons. Foreplan times out after 20 persons and Approximate Foreplan after 191 persons. For ten persons, Foreplan is 91 times faster than XADD Symbolic VI. Approximate Foreplan is even 14.000 times faster, which are four orders of magnitude. For 15 persons, Foreplan is more than four times faster than ALP and Approximate Foreplan is more than 4.217 times faster than ALP. For 20 persons, Approximate Foreplan is more than 4.077 times faster than Foreplan. Thus, when using a symbolic solver, we can only solve the epidemic example for up to ten persons. With a factored and approximate approach, we can go up to 15 persons. In contrast, when using Foreplan, we can solve the epidemic example even for 20 persons and with Approximate Foreplan we can go further to 191 persons, which is ten times more than what ALP can solve.

We also run the four algorithms on a fully connected SysAdmin example with a timeout of two hours. XADD Symbolic VI times out after eleven computers and ALP after 12. With Foreplan, we can go up to 64 computers and with Approximate Foreplan even up to 94 computers. At eleven computers, Foreplan is more than 44.413 times faster than XADD Symbolic VI and Approximate Foreplan is even more than 69.944 times faster. At 12 computers, Foreplan is more than 8.359 times faster than ALP and Approximate Foreplan even more than 16.173 times. For 64 computers, Approximate Foreplan is more than 696 times faster than Foreplan.

For the BoxWorld example, we use three cities and set a time limit of 15 hours. XADD Symbolic VI times out after two boxes and trains. ALP and Foreplan time out after nine boxes and trains. Approximate Foreplan manages to go up to 30 boxes and trains. At two boxes and trains, Foreplan is more than 83.227 times faster than XADD Symbolic VI and Approximate Foreplan even more than 137.502 times. At nine boxes and trains, Foreplan is more than three times faster than ALP and Approximate Foreplan is even more than 33.136 times faster than ALP.

For assessing the quality of the policy given by Approximate Foreplan, we calculate the ratio of wrong actions over the total number of actions in every ground state. For ten persons, ALP and Approximate Foreplan both deviate in 1.2 % of all actions. For two to ten persons, the deviation was 2.98 % at most, with reducing errors when increasing the number of individuals. In all cases, ALP and Approximate Foreplan have an identical error. For SysAdmin, ALP and Approximate Foreplan both return the optimal policy in the tests going up to nine computers.

Overall, (Approximate) Foreplan achieves a speedup of several orders of magnitude and computes policies for significantly more objects within the same time and memory limits. Moreover, the policy given by Approximate Foreplan is correct for SysAdmin and for more than 97 % of all actions in the epidemic example.

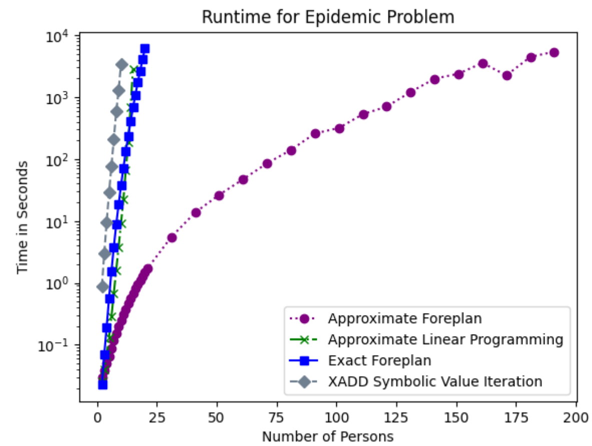

Figure 2: Runtime of (Approximate) Foreplan, ALP and XADD Symbolic VI on the epidemic example for up to 191 persons with a time limit of two hours.

8 CONCLUSION

Propositional planning approaches struggle with having numerous indistinct objects and different types of actions applied concurrently. While first-order MDPs can cope with numerous objects, current approaches for solving planning in them neglect concurrent actions and still need to represent actions for each subset individually, resulting in exponentially many actions. In this paper, we present Foreplan, a relational forward planner solving the exponential explosion by lifting the objects: Foreplan groups indistinguishable objects by a representative and carries out calculations on a representative-level. Afterwards, the result is projected to the whole group. Using histograms and focusing only on the number of objects an action is applied to, we effectively reduce the action space from exponential to polynomial in the number of objects.

In future work, we aim to develop a hybrid approach combining Foreplan and Golog [22]. With the forward search in Foreplan, we can identify states reachable from the initial state while the backwards search in Golog computes the exact optimal policy. Furthermore, the techniques from Foreplan can be transferred to first-order partially observable MDPs [34].

ACKNOWLEDGMENTS

The research for this paper was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC 2176 'Understanding Written Artefacts: Material, Interaction and Transmission in Manuscript Cultures', project no. 390893796. The research was conducted within the scope of the Centre for the Study of Manuscript Cultures (CSMC) at Universität Hamburg.

REFERENCES

[1] Udi Apsel and Ronen I Brafman. 2011. Extended lifted inference with joint formulas. In *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence*. 11–18.

[2] Richard Bellman. 1957. Dynamic programming. *Princeton University Press* (1957).

[3] Daniel S Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. 2002. The complexity of decentralized control of Markov decision processes. *Mathematics of operations research* 27, 4 (2002), 819–840.

[4] Craig Boutilier, Richard Dearden, and Moisés Goldszmidt. 2000. Stochastic dynamic programming with factored representations. *Artificial intelligence* 121, 1-2 (2000), 49–107.

[5] Craig Boutilier, Raymond Reiter, and Bob Price. 2001. Symbolic dynamic programming for first-order MDPs. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, Vol. 1. 690–700.

[6] Tanya Braun, Marcel Gehrke, Florian Lau, and Ralf Möller. 2022. Lifting in multi-agent systems under uncertainty. In *Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence*. 233–243.

[7] Augusto B Corrêa and Giuseppe De Giacomo. 2024. Lifted Planning: Recent Advances in Planning Using First-Order Representations. In *Proceedings of the 33rd International Joint Conference on Artificial Intelligence*. 8010–8019.

[8] D. P. de Farias and B. Van Roy. 2003. The Linear Programming Approach to Approximate Dynamic Programming. *Operations Research* 51, 6 (2003), 850–865.

[9] Luc De Raedt, Kristian Kersting, Sriraam Natarajan, and David Poole. 2016. Statistical relational artificial intelligence: Logic, probability, and computation. *Synthesis lectures on artificial intelligence and machine learning* 10, 2 (2016), 1–189.

[10] Thomas Dean, Robert Givan, and Sonia Leach. 1997. Model reduction techniques for computing approximately optimal solutions for Markov decision processes. In *Proceedings of the Thirteenth conference on Uncertainty in Artificial Intelligence*. 124–131.

[11] Rina Dechter. 1999. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence* 113, 1-2 (1999), 41–85.

[12] F d'Epenoux. 1960. Sur un probleme de production et de stockage dans l'aléatoire. *Revue Française de Recherche Opérationelle* 14, 3-16 (1960), 4.

[13] Marcel Gehrke, Tanya Braun, and Ralf Möller. 2019. Lifted Temporal Maximum Expected Utility. In *Advances in Artificial Intelligence*. 380–386.

[14] Marcel Gehrke, Tanya Braun, Ralf Möller, Alexander Waschkau, Christoph Strumann, and Jost Steinhäuser. 2019. Lifted Maximum Expected Utility. In *Artificial Intelligence in Health*. 131–141.

[15] Robert Givan, Thomas Dean, and Matthew Greig. 2003. Equivalence notions and model minimization in Markov decision processes. *Artificial Intelligence* 147, 1-2 (2003), 163–223.

[16] Carlos Guestrin, Daphne Koller, Ronald Parr, and Shobha Venkataraman. 2003. Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research* 19 (2003), 399–468.

[17] Qi Huangfu and JA Julian Hall. 2018. Parallelizing the dual revised simplex method. *Mathematical Programming Computation* 10, 1 (2018), 119–142.

[18] Jihwan Jeong, Parth Jaggi, Andrew Butler, and Scott Sanner. 2022. An Exact Symbolic Reduction of Linear Smart Predict+Optimize to Mixed Integer Linear Programming. In *Proceedings of the 39th International Conference on Machine Learning*, Vol. 162. 10053–10067.

[19] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. 1998. Planning and acting in partially observable stochastic domains. *Artificial intelligence* 101, 1-2 (1998), 99–134.

[20] Kristian Kersting. 2012. Lifted Probabilistic Inference. In *Proceedings of the 20th European Conference on Artificial Intelligence*. 33–38.

[21] Daphne Koller and Ronald Parr. 1999. Computing factored value functions for policies in structured MDPs. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, Vol. 2.

[22] Hector J Levesque, Raymond Reiter, Yves Lespérance, Fangzhen Lin, and Richard B Scherl. 1997. GOLOG: A logic programming language for dynamic domains. *The Journal of Logic Programming* 31, 1-3 (1997), 59–83.

[23] Mathias Niepert and Guy Van den Broeck. 2014. Tractability through exchangeability: A new perspective on efficient probabilistic inference. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 28. 2467–2475.

[24] Christos H Papadimitriou and John N Tsitsiklis. 1987. The complexity of Markov decision processes. *Mathematics of operations research* 12, 3 (1987), 441–450.

[25] David Poole. 2003. First-order probabilistic inference. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, Vol. 3. 985–991.

[26] Martin L Puterman. 1990. Markov decision processes. *Handbooks in operations research and management science* 2 (1990), 331–434.

[27] Scott Sanner and Craig Boutilier. 2005. Approximate linear programming for first-order MDPs. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence* (Edinburgh, Scotland) *(UAI'05)*. AUAI Press, Arlington, Virginia, USA, 509–517.

[28] Scott Sanner and Craig Boutilier. 2007. Approximate Solution Techniques for Factored First-Order MDPs.. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling*. 288–295.

[29] Scott Sanner and Kristian Kersting. 2010. Symbolic dynamic programming for first-order POMDPs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 24. 1140–1146.

[30] Nima Taghipour. 2013. Lifted probabilistic inference by variable elimination. *PhD dissertation, KU Leuven* (2013).

[31] Nima Taghipour, Daan Fierens, Jesse Davis, and Hendrik Blockeel. 2013. Lifted Variable Elimination: Decoupling the Operators from the Constraint Language. *Journal of Artificial Intelligence Research* 47 (2013), 393–439.

[32] Ayal Taitler, Michael Gimelfarb, Sriram Gopalakrishnan, Martin Mladenov, Xiaotian Liu, and Scott Sanner. 2022. pyRDDLGym: From RDDL to Gym Environments. *arXiv preprint arXiv:2211.05939* (2022).

[33] P.M. Vaidya. 1989. Speeding-up linear programming using fast matrix multiplication. In *30th Annual Symposium on Foundations of Computer Science*. 332–337.

[34] Jason D Williams, Pascal Poupart, and Steve Young. 2005. Factored partially observable Markov decision processes for dialogue management. In *Proceedings of the 4th IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems*. 76–82.

[35] Nevin L Zhang and David Poole. 1994. A simple approach to Bayesian network computations. In *Proceedings of the 10th Canadian Conference on AI*. 171–178.