AN ABSTRACT OF THE DISSERTATION OF

Aswin Raghavan for the degree of Doctor of Philosophy in Computer Science presented on January 27, 2017.

Title: Domain-Independent Planning for Markov Decision Processes with Factored State and Action Spaces

Abstract approved: _____

Prasad Tadepalli

Markov Decision Processes (MDPs) are the de-facto formalism for studying sequential decision making problems with uncertainty, ranging from classical problems such as inventory control and path planning, to more complex problems such as reservoir control under rainfall uncertainty and emergency response optimization for fire and medical emergencies. Most prior research has focused on exact and approximate solutions to MDPs with factored states, assuming a small number of actions. In contrast to this, many applications are most naturally modeled as having factored actions described in terms of multiple action variables. In this thesis we study domain-independent algorithms that leverage the factored action structure in the MDP dynamics and reward, and scale better than treating each of the exponentially many joint actions as atomic. Our contributions are three-fold based on three fundamental approaches to MDP planning namely exact solution using symbolic dynamic programming (DP), anytime online planning using heuristic search and online action selection using hindsight optimization.

The first part is focused on deriving optimal policies over all states for MDPs whose state and action space are described in terms of multiple discrete random variables. In order to capture the factored action structure, we introduce new symbolic operators for computing DP updates over all states efficiently by leveraging the abstract and symbolic representation of Decision Diagrams. Addressing the potential bottleneck of diagrammatic blowup in these operators we present a novel and optimal policy iteration algorithm

that emphasizes the diagrammatic compactness of the intermediate value functions and policies. The impact is seen in experiments on the well-studied problems of inventory control and system administration where our algorithm is able to exploit the increasing compactness of the optimal policy with increasing complexity of the action space.

Under the framework of anytime planning, the second part expands the scalability of our approach to factored actions by restricting its attention to the reachable part of the state space. Our contribution is the introduction of new symbolic generalization operators that guarantee a more moderate use of space and time while providing non-trivial generalization. These operators yield anytime algorithms that guarantee convergence to the optimal value and action for the current world state, while guaranteeing bounded growth in the size of the symbolic representation. We empirically show that our online algorithm is successfully able to combine forward search from an initial state with backwards generalized DP updates on symbolic states.

The third part considers a general class of hybrid (mixed discrete and continuous) state and action (HSA) MDPs. Whereas the insights from the above approaches are valid for hybrid MDPs as well, there are significant limitations inherent to the DP approach. Existing solvers for hybrid state and action MDPs are either limited to very restricted transition distributions, require knowledge of domain-specific basis functions to achieve good approximations, or do not scale. We explore a domain-independent approach based on the framework of hindsight optimization (HOP) for HSA-MDPs, which uses an upper bound on the finite-horizon action values for action selection. Our main contribution is a linear time reduction to a Mixed Integer Linear Program (MILP) that encodes the HOP objective, when the dynamics are specified as location-scale probability distributions parametrized by Piecewise Linear (PWL) functions of states and actions. In addition, we show how to use the same machinery to select actions based on a lower-bound generated by straight-line plans. Our empirical results show that the HSA-HOP approach effectively scales to high-dimensional problems and outperforms baselines that are capable of scaling to such large hybrid MDPs. In a concluding case study, we cast the real-time dispatch optimization problem faced by the Corvallis Fire Department as an HSA-MDP with factored actions. We show that our domain-independent planner significantly improves upon the responsiveness of the baseline that dispatches the nearest responders.

Domain-Independent Planning for Markov Decision Processes with Factored State and Action Spaces

by

Aswin Raghavan

A DISSERTATION

submitted to

Oregon State University

In partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented January 27, 2017
Commencement June 2017

Doctor of Philosophy dissertation of Aswin Raghavan presented on January 27, 2017.

APPROVED:

Major Professor, representing Computer Science

Director of the School of Electrical Engineering and Computer Science

---

Dean of the Graduate School

I understand that my dissertation will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my dissertation to any reader upon request.

Aswin Raghavan, Author

ACKNOWLEDGEMENTS

I would like to start by thanking Prasad Tadepalli for offering me the opportunity to pursue my doctorate at Oregon State University, followed by years of guidance, encouragement, academic and financial support. As a new graduate student, it can be daunting to choose a research topic. Prasad gave me the freedom to identify and pursue any suitable topic within my interest in sequential decision making in AI, and financially supported me with teaching assistantships during this time. His encouragement to particiapte in the International Planning Competition (2011) directly led to the realisation that factored action spaces represent an interesting aspect that is largely unexplored. Despite his busy schedule, I could always rely on extended discussions on research topics beyond this dissertation, where I learnt to present my ideas in a succinct and technically sound manner. His critical feedback and countless hours of proofreading made it possible to publish the contents of this dissertation over multiple conference papers. As a graduate researcher I strived to match his critical thinking and will continue to do so in the future.

I am extremely thankful to Roni Khardon for his continuous guidance throughout this research. He made significant technical contributions to this thesis. He was the first to identify the theoretical underpinning of one of our algorithms and significantly helped with the development of the technical proofs that lead to our novel opportunistic policy iteration algorithm. In addition to years of remote collaboration, he generously agreed to be a remote member of my doctoral committee.

I would like to thank Alan Fern for his guidance and feedback on each component of this dissertation. In the initial stages, he encouraged me to think about factored actions and motivated me by their occurence in his other projects. In the face of this unexplored dimension, his emphasis on empirical performance had a significant impact on the direction of this research and algorithmic development.

I would like to thank Scott Sanner for creating a new set of benchmarks for domain independent planning via his RDDL domain description language. I used RDDL extensively for the experiments in this dissertation and was even able to extend it thanks to the code that he made publicly available. Having followed Scotts' work on related topics

very closely, I was fortunate to collaborate with him at OSU on a significant portion of this thesis.

One aspect of this dissertation concerns the optimization of dispatch actions in response to real-world 9-1-1 emergencies that occur in Corvallis. I am thankful to Carl Neidner and Coelo Company of Design for providing the dataset of emergencies that occured in Corvalllis. I am also thankful to Ronald Bjarnason for making his code available from his earlier work on this problem.

I am thankful to Saket Joshi for initially encouraging me to work on symbolic planning and factored actions. I am also thankful to Balaraman Ravindran for inspiring me to pursue my doctorate and providing me with an example of the academic research-publish cycle.

On a personal note, I am very lucky to have the love and support of my wife Gayathry during my graduate education. Her infectious enthusiasm energized my research as she enriches all aspects of my life. I am also thankful to my fellow graduate student Jesse Hostetler for his friendship and insightful discussions on research, as well as conversations ranging from philosophy to matters of everyday life. Finally, I would like to thank my parents Latha and Raghavan for their love, support and personal sacrifices. They taught me the value of hard work and encouraged my curiosity in computer programming and science. They supported me at every stage of my education in every possible way. I look up to the kindness and affection they show to everyone around them.

I am thankful for the funding provided by the National Science Foundation (NSF) grants IIS-0964457, IIS-0964705, IIS-1219258, IIS-1616280, and IIS-1619433.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

Figure Page

LIST OF FIGURES (Continued)

Figure Page

LIST OF TABLES

LIST OF TABLES (Continued)

Chapter 1: Introduction

Markov Decision Processes (MDPs) [118] have been the de-facto formalism for studying sequential decision making problems with uncertainty. The MDP planning problem concerns choosing actions that effect states stochastically with the goal of maximizing the expected reward accumulated over time, with respect to some reward function over states. It is well known that the planning problem in MDPs is very hard in theory, and it is challenging to design domain-independent algorithms whose performance scales well with the size of the MDP.

Many planning problems involve choosing actions from a combinatorial set of decisions at every time step, where each action can be described as an assignment to a set of *action variables*. These include many planning problems of interest that involve controlling multiple actuators simultaneously as in the Mars Rover problem [107], controlling multiple units in real-time strategy games [6], with factored exogenous events as in elevators control [7], and water reservoirs control under rainfall uncertainty [124], and in Smart Grids [2]. Factored actions frequently arise in planning problems with spatial spreading, e.g. conservation planning for birds [150], or controlling forest fires and eradicating invasive species [37], where each land parcel has a corresponding action. Domains with temporally extended actions are naturally modeled with factored actions e.g. emergency response optimization [14] where the action variables correspond to responding vehicles.

On the other hand, the computational complexity of most of the standard domain-independent planning approaches scales with the number of actions, that is exponential in the number of action variables. Dynammic Programming (DP) approaches must tackle the maximization over an exponential number of action-value functions. Online approaches based on heuristic search such as Real-Time Dynamic Programming (RTDP) [8] and UCT [84] suffer from the large branching factor caused by the large number of actions. This issue is exacerbated when continuous and discrete valued states and actions abound. Therefore, there is a need for planning algorithms that can leverage the structure of such factored actions, in order to scale better than treating each of the

exponentially many joint actions as atomic. Earlier work on Concurrent MDPs [148] tackles concurrency of actions but ignores the factored structure in the state dynamics. Although some notable frameworks from classical planning have incorporated concurrency, these lack in generality and are not applicable in factored state and action MDPs. For example, the works of [152, 94] can only tackle goal-based problems with specified initial states.

In this thesis, we present new optimal and online algorithms that are tailored to factored states and actions. The takeaway is that explicitly accounting for the factored nature of the effects of joint actions on the dynamics and reward leads to improved empirical performance and scalability against the curse of dimensionality that is the number of state and action variables. This is exemplified in a well studied problem with factored states and actions where increasing the number of actions does not necessarily lead to increased planning time as seen in existing MDP solvers, but on the contrary our algorithm is able to exploit the increased simplicity of the optimal solution with decreasing planning time. Our contributions span different settings for domain independent model based planning, in that a declarative description of the true MDP dynamics and reward is assumed as input. Similarly, the theoretical guarantees of our algorithms also span provably optimal policies to anytime optimal online action selection to heuristic online action selection for a wide class of factored state and action MDPs.

Factored MDPs [18] were originally invented as a compact description for very large MDPs with multidimensional discrete-valued states. Factored planning algorithms attempt to exploit regularities and context-specific dependencies in the problem description in order to compute a compact representation of the optimal value function and policy. Algorithms that exploit factored state dynamics are successfully able to tackle the apparent state-space explosion, that is the exponential growth of the number of states with the number of state dimensions. However, most prior work has focused on exact Symbolic Dynamic Programming (SDP) [66, 130] and approximate solutions [129, 137] to factored MDPs with factored states only. This includes symbolic Value Iteration(VI) using Algebraic Decision Diagrams (ADDs) [66], symbolic Real Time Dynamic Programming [44], and symbolic heuristic search [46].

The bottleneck for existing SDP planners applied to factored actions is that they work by iteratively computing the result of decision-theoretic regression over each joint action. These algorithms ignore any structure in the action space, causing both plan-

ning and execution times to scale at least linearly in the number of actions, which is exponential in the number of action variables. In Chapter 3, we introduce new SDP operators based on our previous work [120] and [121], which exploit a compact factored-action MDP representation in order to compute value functions in the form of algebraic decision diagrams over state and action variables. Addressing the potential bottleneck of diagrammatic blowup in these operators, we present a novel and optimal policy iteration algorithm that emphasizes the diagrammatic compactness of the intermediate value functions and policies (Section 3.6).

The optimal SDP planners proposed in Chapter 3 sometimes scale to large MDPs, but depend on compactly representing the optimal value function of the entire MDP [66]. Due to this requirement, these algorithms exceed practical memory and time limits in many problems of interest. The success of online planning in MDPs depends crucially on the extent to which the information gathered from search is generalized to unseen states. In the absence of generalization and heuristic guidance, the planner must explore the entire reachable state space.

Symbolic Real-Time Dynamic Programming (sRTDP) [44, 46] aims to combine the benefits of the symbolic methods and online planning by incorporating symbolic state generalization into the computation of the online planner. However, sRTDP is a general framework, and its performance is sensitive to the definition of generalized states. Existing definitions in prior work lead to algorithms that exceed memory limits in many cases. Despite the aim for generalization, the resulting planner is often inferior to the corresponding algorithms working in the flat state space (e.g. RTDP). We give symbolic RTDP algorithms that guarantee superior anytime performance than the corresponding tabular representation in Chapter 4 based on our previous work [122].

Our main contribution in Chapter 4 is the introduction of new symbolic generalization operators that guarantee a more moderate use of space and time, while providing non-trivial generalization. The first set of operators guarantees bounded growth in the size of the symbolic representation. These operators are combined with the pruning operator developed in Chapter 3 to further control the size of intermediate results. The resulting anytime planning algorithms are convergent and provide generalization only when it does not increase space requirements compared to a flat state space search. To the best of our knowledge, it is the first symbolic algorithm to yield a sound generalization while guaranteeing not to use more memory than flat RTDP. We show that in

some domains, the symbolic online planner is able to identify succinct and meaningful generalized states. These results show the potential for symbolic speedup learning for intra-problem generalization for online planning.

The next chapter considers a more general class of hybrid (mixed discrete and continuous) state and action (HSA) MDPs. Many real-world decision-theoretic planning problems are naturally modeled using concurrent, hybrid (discrete and continuous) state and action (HSA) MDPs. Existing approaches to solving expressive HSA-MDPs largely fall into two categories: dynamic programming for special restricted classes of HSA-MDPs, and approximate optimization of restricted value function or policy representations. Unfortunately, each category has critical limitations discussed in Chapter 5. Whereas the insights from the above approaches are valid for hybrid MDPs as well, there are significant limitations inherent to the DP approach. Existing solvers for hybrid state and action MDPs are either limited to very restricted transition distributions, require knowledge of domain-specific basis functions to achieve good approximations, or do not scale.

In that chapter, based on our previous work [123], we explore a qualitatively different approach to solving a broad class of HSA-MDPs that does not require domain-specific assumptions on the value function or policy representation. Our approach is based on developing the framework of hindsight optimization (HOP) [25, 26] for HSA-MDPs. HOP provides an upper bound on the finite-horizon action values in the current state, which can be used for action selection. But the challenge is to compute this bound efficiently. We develop a generic linear space and time compilation of an expressive subset of HSA-MDPs to a mixed integer linear program (MILP). This compilation is augmented with action constraints to yield different algorithmic variations. This generalizes previous work on HOP [72] to handle both continuous random variables and state-action dependent stochasticity. We develop a second variant based on straight line plans which is complementary in that it provides a lower bound on action values.

The expressive subset of HSA-MDPs consists of location-scale probability distributions for the transitions of each state variable. The importance of location-scale distributions is that they allow a compile-time encoding of the sampled next states at some time step in the future, even though the parameters of the corresponding density function involves state and action variables that are unknown at compile time. In our instantiation of this approach, the MILP solver then maximizes over the unknown policy over multiple sampled futures. This instantiation assumes that the parameters are piecewise

linear functions of state and action variables, or location-scale distributions over them. To the best of our knowledge, this is the first characterization of a subset of HSA-MDPs, and corresponding fragment of the RDDL planning domain description language, that have a linear time compilation to an MILP of linear size. The general approach is more broadly applicable by using a more complex solver that matches the function space of the parameters, e.g. using an MIQP solver when a quadratic function is the appropriate reward function.

In each chapter we extensively compare the empirical performance of our algorithms to those based on flat actions, as well as simple hand coded baselines, on multiple benchmark planning problems. In a concluding case study in Chapter 6, we cast the real-time dispatch optimization problem faced by the Corvallis Fire Department as an HSA-MDP with factored actions. There are no existing domain independent planners that are applicable to this HSA-MDP, thus prior research primarily relies on a heuristic policy that always dispatches the nearest responders.

After briefly reviewing the vast literature on this problem, we justify the assumptions of our domain description that allows HSA-HOP to be applied. We adapt HSA-HOP to sample futures from real-world data collected in the city of Corvallis. We show that HSA-HOP significantly improves the responsiveness by significantly lowering the fraction of unresponded emergencies as well as the average response time over the responded calls. Secondly, we illustrate model learning in the space of RDDL expressions that are usable by HSA-HOP. Empirically, the responsiveness of HSA-HOP is further improved by incorporating the learned model because it generalizes the emergencies beyond the observed data.

Chapter 2: Problem Formulation

After presenting the relevant background on MDPs in Section 2.1, we develop our formulation of MDPs with factored state and action spaces in Section 2.2. We build upon the Modified Policy Iteration in Chapter 3.

2.1 Markov Decision Processes (MDP)

MDPs [118] and factored MDPs [16] have been used successfully to solve sequential decision problems under uncertainty. An MDP is a tuple $(\mathbb{S}, \mathbb{A}, T, R, \gamma)$ where $\mathbb{S}$ is a finite state-space, $\mathbb{A}$ is a finite action space, $T : \mathbb{S} \times \mathbb{A} \times \mathbb{S} \to [0, 1]$ denotes the transition function such that $T(s, a, s') = Pr(s'|s, a)$, $R : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$ denotes the immediate reward of taking action a in state s, and the parameter $\gamma \leq 1$ is used to discount future rewards. We refer to this representation as the *flat* MDP representation as it uses flat or atomic states and actions. The following notions of value function and policy are used throughout the thesis.

A policy indicates the action to choose in a state. To facilitate a general view, we emphasize that the policy can be a partial mapping or multi-map $\pi : \mathbb{S} \to 2^{\mathbb{A}}$ that assigns each state to a set of actions. The value function $V^\pi : \mathbb{S} \to \mathbb{R}$ is the expected discounted accumulated reward $E[\sum_{i=0}^{n} \gamma^i r(s_i, \pi(s_i)) \mid \pi]$ where s_i is the i'th state visited when following π. An optimal policy π^* is a policy that maximizes the value for all states simultaneously. The action-value function $Q^\pi : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$ indicates the expected reward accumulated by executing action a in state s and following policy π thereafter. For every MDP, there is a deterministic optimal policy π^* and unique optimal value function V^* such that $V^*(s) = \max_a Q^*(s, a)$ and $\pi^*(s) = \arg\max_a Q^*(s, a)$.

Dynamic Programming is one approach to arrive at the optimal policy via computing the optimal value function. For example, the Value Iteration (VI) algorithm identifies V^* and π^* by the fixed point of iteration below. Each iteration is a Dynamic Programming

update for all states and is known as the bellman backup operator $\mathcal{T}(V_n)$,

$$V_{n+1}(s) = \mathcal{T}(V_n)(s) = \max_a \left[R(s,a) + \gamma \sum_{s'} Pr(s'|s,a) V_n(s') \right] \tag{2.1}$$

with $V_0^\pi(s') = 0 \ \forall s' \in \mathbb{S}$ and,

$$\pi_{n+1}(s) = \arg\max_a \left[R(s,a) + \gamma \sum_{s'} Pr(s'|s,a) V_n(s') \right]$$

Policy Iteration (PI) is a complimentary approach that identifies the optimal policy by alternating policy evaluation and policy improvement. The value function V^π for any fixed partial policy π is the fixed point of the following iteration,

$$V_{n+1}^\pi(s) = \mathcal{T}_\pi(V_n)(s) = \left[R(s,\pi(s)) + \gamma \sum_{s'} T(s,\pi(s),s') V_n^\pi(s') \right] \tag{2.2}$$

with $V_0^\pi(s') = 0 \ \forall s' \in \mathbb{S}$. This operator is known as policy evaluation $\mathcal{T}_\pi$, and the next operator is known as policy improvement over the policy π using V_n^π,

$$V_0^{\pi'}(s) = \max_a \left[R(s,a) + \gamma \sum_{s'} Pr(s'|s,a) V_n^\pi(s') \right]. \tag{2.3}$$

Note that $V_{n+1} = \mathcal{T}(V_n) = \max_a Q^a(V_n)$ where $Q^a(V_n) = \mathcal{T}_a(V_n)$. In fact, both VI and PI belong to the family of Modified Policy Iteration (MPI) algorithms. Until now we described the MDP and its update equations in terms of "flat" or "atomic" states and actions. In matrix-vector notation, an implementation of Equation 2.2 stores V and R as vector of dimension $|\mathbb{S}| \times 1$ and T as a matrix of dimension $|\mathbb{S}| \times |\mathbb{S}|$. The outer maximization in Equation 2.1 involves the maximum over $|\mathbb{A}|$ matrices of dimension $|\mathbb{S}| \times 1$. Clearly, this is prohibitive for exponentially large state and action spaces as is our case. However, most problems of interest have structure inherent among state and action variables. Factored MDPs [18] exploit this structure and compute bellman backups in a more compact form.

Modified PI [119] is a general PI framework for solving MDPs, that contains the two popular approaches to planning in MDPs namely Value Iteration and PI. PI is generally

Algorithm 2.1.1: MPI(k)

//Policy Improvement
$(V^0_{n+1}, \pi) \leftarrow \mathcal{T}(V_n)$
if $||V^0_{n+1} - V_n|| \leq \epsilon(1-\gamma)/2\gamma$ **return** (V_n, π)

//Partial Policy Evaluation
$j \leftarrow 0$
while $j < k$
do $\begin{cases} V^{j+1}_{n+1} \leftarrow \mathcal{T}_\pi(V^j_{n+1}) \\ j \leftarrow j+1 \end{cases}$
$V_{n+1} \leftarrow V^k_{n+1}; n \leftarrow n+1.$

Figure 2.1: One iteration of Modified Policy Iteration (MPI) [119] : $\mathcal{T}$ and $\mathcal{T}_\pi$ denote a bellman and policy backup respectively. The value function V_0 is initialized with a lower bound on V^*.

preferred over VI as it directly optimizes the function of interest namely the policy. Although VI like PI converges to the optimal policy, the value of a state in VI may be slow to converge to its true value, whereas the optimal action may converge quickly.

The general MPI algorithm for an MDP is shown in 2.1. MPI alternates between policy improvement and steps of policy evaluation. When $k = 0$ ($k = \infty$) we have VI (PI, respectively). When $k > 0$, we can expect larger jumps in the policy because the improvement is based on a more accurate value of the current policy in comparison to $k = 0$. MPI for any value of k has the following theoretical properties which will be used later in proving convergence of our algorithm. MPI is guaranteed to converge to the optimal value and policy if V_0 is initialized with a lower bound for value of states. We study the symbolic implementation of MPI via the introduction of backup operators tailored to Factored Actions.

2.2 MDPs with Factored State and Action Spaces

Our formulation of a factored-action MDP (FA-MDP) is the tuple $(\mathbb{S}, \mathbb{A}, T, R, \gamma, A_C)$. The state space $\mathbb{S}$ is specified by a finite set of state variables $\mathbf{X} = \{X_1, \ldots, X_l\}$ so that $|\mathbb{S}| = 2^l$, and an atomic state s is an assignment of values to them. Following [18], we

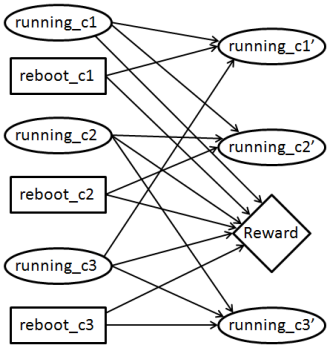

Figure 2.2: Illustration of the factored actions in the SysAdmin example.

assume that $X_1, \ldots, X_n$ are binary because of our intention of finding optimal policies. While factored MDPs with real valued state variables have been considered, for example in [45, 130], it is yet unknown whether optimal solutions can be derived in general.

However, in contrast with previous work on exact solutions to factored MDPs where $|\mathbb{A}|$ is assumed to be small, the action space $\mathbb{A}$ is also specified by a finite number of binary variables $A_1, \ldots, A_m$, herein called *action variables* and $|\mathbb{A}| = 2^m$. In what follows we use $a \in \mathbb{A}$ to represent a ground action where action variables $\mathbf{A} = \{A_1, \ldots, A_m\}$ are instantiated to particular binary values.

The markovian transition probability function $T(s, a, s') = Pr(s'|s, a)$ is factored as a product of conditional probability distributions over next state variables $\mathbf{X}'$. Each $P_i(q_i)$ is a function of some (typically small) subset $q_i \subseteq \{\mathbf{X}, \mathbf{A}, \mathbf{X}'\}$ where $\mathbf{X}'$ are the next state variables and the set of dependencies is acyclic. At any time t, $x_i^{(t+1)} \sim P_i(q_i^{(t)}) = Pr(X_i'|q_i^{(t)})$. The reward function is modeled in a similar manner.

Note that the factorization is similar to a Dynamic Bayesian Network (DBN) [34, 111] (a two time slice bayesian network). Previous work on optimal planning for factored state MDPs [18, 66] required as input a separate DBN describing the effects of each action. The inclusion of action variable nodes and reward nodes, in addition to current and next state variable nodes, allows a compact representation of the effects of factored actions. For example, localized and independent effects on disjoint state variables can be compactly specified. This representation is sometimes known as an Influence Diagram [68]. As noted by [18], the inclusion of action variables increases the treewidth of the network that require corresponding algorithms to deal with the blowup in value computation.

For the purpose of exposition, we will use the prototypical system administration problem (SysAdmin [57] for short) as a running example. A SysAdmin problem instance (Section 3.7) consists of a network of n computers, each either running or failed with an associated action of rebooting. The objective is to maximize the uptime of computers under a fixed cost of rebooting, and under stochastic spreading of failures according to the network topology. Our interest is when we increase the number of computers that can be rebooted in one time step. On one hand, the hardness in terms of planning time increases for optimal DP algorithms, due to the maximization over actions, or due to sampling of actions as in online tree search algorithms [81]. On the other hand, the optimal policy is increasingly simple due to fewer constraints, that in the limit of factored actions the optimal policy is to simply reboot all failed computers.

Returning to our running example, Figure 2.2 shows the transition and reward representation in the SysAdmin domain. The factorization captures the fact that the computers `c1`, `c2` and `c3` are arranged in a directed ring network so that the running status of each computer in the network is influenced by its own reboot action and the status of its predecessor.

Finally, the last element of our formulation is a function $A_C : \mathbb{A} \to \{1, -\infty\}$ which denotes the set of valid actions. In our empirical evaluation, we use A_C to restrict the number of actions via restricting the number of action variables that can be set in any state, allowing us to evaluate the scaling of our algorithms with the size of the action space while keeping the state space the same. Our encoding using action variables can potentially create combinations of assignments to action variables that are not valid, and can lead to invalid states that never occur in practice.

For example, consider the FA-MDP for controlling multiple elevators in a building. Here the state variables include the location of elevators (one binary variable for each floor) and action variables include one for moving each elevator one floor above and one for moving each elevator below its current location. Roughly speaking, the update rule for each action is to simply set the corresponding binary variable for the location of the elevator. Here the factorization of the effects of actions causes a problem namely, the simultaneous action of moving an elevator up and down causes the elevator to be in two locations at the same time.

We use the input FA-MDP *as is* throughout this thesis, with the following implied assumptions relating invalid states and invalid actions.

Assumption 1. *1. Any initial state is valid.*

2. Any state that is valid has at least one valid action.

3. Executing any valid action in a valid state cannot lead to any invalid state with probability greater than zero.

The above assumptions ensure that the stochastic dynamics of the FA-MDP can be used as specified, without the need for setting the probability of any transition to zero followed by normalization. Using induction on the length of any sequence of states, it can be shown that a policy that outputs legal actions does not reach invalid states. In our algorithms, we force the Q-value to $-\infty$ whenever A_C is $-\infty$, so that the corresponding greedy policy outputs legal actions only. Using induction on n, it can be shown that V_{n+1} is not equal to $-\infty$ for every valid state.

Chapter 3: Optimal Planning using Symbolic Dynamic Programming

3.1 Introduction

There are several approaches addressing factored actions, but unlike our work either do not give optimal policies over all states or not applicable in a domain independent setting, or do not leverage the jointly factored state-action space or make strong assumptions about the MDP. Our work builds on optimal planning with factored states alone as in [18, 66]. We review these next under the framework of Symbolic Dynamic Programming (SDP).

Factored MDPs [18] were originally invented as a compact description for very large MDPs with multidimensional discrete-valued states. Factored planning algorithms attempt to exploit regularities in the problem description in order to compute a compact representation of the optimal value function and policy. Algorithms that exploit factored state dynamics are successfully able to tackle the apparent state-space explosion, that is the exponential growth of the number of states with the number of state dimensions. However, most prior work has focused on exact Symbolic Dynamic Programming (SDP) [66, 130] and approximate solutions [129, 137] to factored MDPs with factored states only.

The literature on SDP generally considers variants of Dynamic Programming (DP) that compute the optimal value function as the fixed point of a symbolic bellman backup operator. This includes symbolic Value Iteration(VI) using Algebraic Decision Diagrams (ADDs) [66], symbolic Real Time Dynamic Programming [44], and symbolic heuristic search [46]. These DP algorithms have been extended to new representations that are more compact in some cases, such as Affine ADDs [129], and extended to deal with continuous quantities [130] and relational states [77]. Similar to our approach of explicitly reasoning about action variables, [154, 144] incorporate factored actions but are restricted to deterministic MDPs.

Notwithstanding these and other representational improvements [31], the bottleneck

for existing SDP planners applied to factored action MDPs (FA-MDPs) is that they work by iteratively computing the bellman update for every joint action. Thus, these algorithms ignore any structure in the action space, causing both planning and execution times to scale at least linearly in the number of joint actions, which is exponential in the number of action variables. Our contribution is to address this issue by extending the exact SDP approach to FA-MDPs with boolean state and action variables.

3.1.1 Outline

We propose two new symbolic operators specific to action variables that let us derive value functions efficiently (Section 3.4). The Factored Action Regression (FAR) operator [120] (Section 3.4.1), generalizes the symbolic operator for factored states, to factored states and actions by incorporating explicit action variables in its symbolic computation leading to a significant improvement in wall-clock solution time. However, the inclusion of action variables exacerbates the high memory usage common to symbolic methods.

The Memory Bounded FAR (MBFAR) operator [120] (Section 3.4.2) addresses this issue and provides a parametric trade-off between memory usage and solution time. MBFAR computes bellman backups over a dynamic set of (partial) action assignments determined by the compactness of intermediate value functions. MBFAR is parameterized by an upper bound on memory usage, where at one extreme (minimal space) we get the standard SDP operator which ignores action variables (e.g. SPUDD), and at the other extreme (unbounded space) we get factored action regression (FAR).

We then consider Modified Policy Iteration (MPI) [119] using the symbolic operators as our approach to the planning problem. MPI generalizes the above SDP approaches which were based on VI, but in comparison, is under-explored in the symbolic planning literature. MPI works by iterating DP backups, but adds a few policy evaluation steps between consecutive backups. This makes MPI especially attractive for factored-action spaces because policy evaluation does not require reasoning about all actions at all states, but rather only about the current policy's action at each state.

Interestingly, the first approach to symbolic planning in MDPs was a version of MPI for factored states called Structured Policy Iteration (SPI) [19], which was later adapted to relational problems [146]. SPI represents the policy as a decision tree with state-variables labeling interior nodes and a joint action as a leaf node. Although SPI

leverages the factored state representation, it represents the policy in terms of joint actions, which fails to capture the structure among the action variables in FA-MDPs. The value of a policy is computed using the graphical form of the policy. For each every joint action $\mathbf{a}$, its Q-function $Q^{\mathbf{a}}$ is first computed, which is impractical in factored action spaces. In addition, the space required for policy backup can be prohibitive because each $Q^{\mathbf{a}}$ is joined to each leaf of the policy. A policy backup in SPI can potentially be more expensive than a bellman backup, which challenges our intuition and motivates a new policy backup operator. In fact, we are not aware of any implementations of SPI that scales well even without the additional complexity of factored actions.

The key to efficient symbolic MPI in FA-MDPs is viewing a policy backup as a symbolic bellman backup with the policy as a constraint. Unfortunately, similar to SPI, a naive handling of the policy constraints in symbolic MPI could significantly increase the size of the ADD. Our algorithmic contribution in Section 3.6, symbolic Opportunistic Policy Iteration (OPI) [121] is a novel PI algorithm with increased flexibility over the sequence of policies in PI. OPI is motivated by the fact that the intermediate policies generated by MPI may have complex value functions not easily representable using ADDs. OPI opportunistically broadens the policy space to policies whose value function is at least as compact as the value function of the current policy. We prove that OPI is guaranteed to converge to the optimal policy at least as quickly as optimal MPI while representing ADDs no larger than optimal MPI at each iteration.

Our empirical evaluation (Section 3.7) in factored-action benchmarks shows that FAR and MBFAR can be significantly more efficient than current approaches of symbolic VI that ignore action variables. We show that OPI is empirically superior to MPI in terms of wall clock time and compactness of intermediate value functions whereas MPI exceeds memory limits and crashes. We close with a discussion of the relevant literature on planning in large action spaces (Section 3.8) and compare them with our approach. Some real-world applications of this work and its extensions for future work are briefly discussed.

3.2 Algebraic Decision Diagrams (ADDs)

We now turn to the representation of the Conditional Probability Distributions (CPD) P_i for each state variable X_i and reward function R. In tabular form the table for P_i

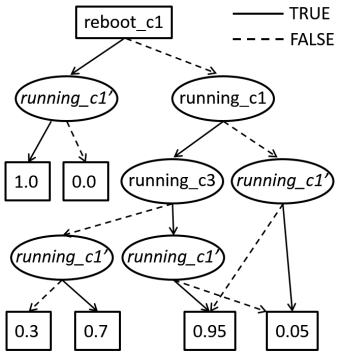

Figure 3.1: SysAdmin : ADD for the conditional probability distribution for the state variable `running_c1`.

has $O(2^{|q_i|})$ entries where $|q_i|$ is the number of parents of X_i' in the DBN. However, most of the arcs in the DBN are contextual [17], leading to tables populated with few unique non-zero probabilities. In the boolean case as in [17], Decision Trees are a natural choice for representing the discrete probability mass functions within the support of the unit hypercube. More generally, Decision Diagrams [147] are a popular implicit representation on the boolean lattice [31], capable of compactly capturing contextual, logical (e.g. XOR-structure), sparsity and algebraic structures. We build upon the SPUDD algorithm of [66], who use Algebraic Decision Diagrams (ADDs) [5] as the representation of the transition functions P_i and R. In contrast to the work of [66] who considered factored states alone, our ADDs for P_i and R will be over both state and action variables. The leaf nodes of the ADD are real-values.

An Algebraic Decision Diagram (ADD) [5] represents a real-valued function $\mathcal{B}^n \to \mathcal{R}$ over n boolean variables in the form of a rooted Directed Acyclic Graph (DAG). Each interior node of the DAG is labeled with a boolean test variable with two directed outgoing edges labeled by true or false that lead to its children.

ADDs are an example of an ordered DD, in that they assume that a total ordering O on the boolean variables is given, which allows canonicity and efficient manipulation of ADDs. Every assignment of truth values to variables traces a unique ordered path to a leaf from the root. Each leaf node represents a distinct value that the ADD function takes over all of its assignments. If D is the ADD, let $D[\mathbf{x}]$ denote the scalar value when D is evaluated on the assignment $\mathbf{x}$.

Returning to our running example, Figure 3.1 shows the ADD representing the conditional probability distribution for the state variable `running_c1`. The primed variable `running_c1'` represents the truth value of `running_c1` in the next state, the solid and dotted lines represent the true and false branches respectively. The ADD shows that `running_c1` becomes true w.p. 1 if it is rebooted (a deterministic action), and otherwise the next state depends on the status of the neighbors. When not rebooted, $c1$ fails w.p. 0.3 if its neighboring computer $c3$ has also failed, and w.p. 0.05 otherwise. Additionally, when not rebooted, a failed computer becomes operational w.p. 0.05. Note that the CPD specifies the transition dynamics for many joint actions compactly (e.g. all joint actions that have the $reboot\_c1 = true$), a consequence of adding action variables.

Ordered ADDs offer a canonical representation for any function (although their compactness depends on the ordering) and polynomial time pointwise operations over the functions they represent. The unary "restrict operator" fixes the value of a variable $\mathbf{x}$ to x (or $\bar{x}$) in an ADD D and returns a new ADD denoted by $D_{\downarrow x}$ (or $D_{\downarrow \bar{x}}$, respectively).

ADDs support binary operations over the functions they represent so that $F\ op\ G = H$ if and only if $\forall x, F(x)\ op\ G(x) = H(x)$. Operations between diagrams will be represented using the usual symbols with $op \in \{+, -, \times, \div, \max, \min\}$, and the distinction between scalar operations and operations over functions should be clear from context. The result H is computed symbolically and in polynomial time in the size of the ADDs F and G. Marginalization operations of such as $G(y) = \max_{\mathbf{x}} F(x, y)$, $G(y) = \min_{\mathbf{x}} F(x, y)$ and $G(y) = \sum_{\mathbf{x}} F(x, y)$ are defined naturally over all possible restrictions of D over values of x e.g. $\max_x F \equiv \max(F_{\downarrow x}, F_{\downarrow \bar{x}})$.

The ADD resulting from these operations has a size that is quadratic in the worst case (but much more compact on average), rather than the potentially exponentially larger number of entries in a tabular representation of the same function. We will define the ordering dependent notion of a path in an ADD.

Definition 2. *A path in an ADD is a sequence of edge traversals starting at the root node and ending in a leaf node. A path induces an assignment to a subset of the variables in O.*

A partial assignment is a truth assignment to a subset of variables in the diagram. An assignment is full if it assigns values to all variables. An extension of a partial assignment is a full assignment which is consistent with it. Every path in the ADD from the root to

the leaf defines a partial assignment over the internal variables in that path. The ADD function returns equal values for all extensions of a path. For example, the path to 0.95 traversed in the example above, defines a partial assignment to three of the variables but leaves other variables, for example running_c2, unspecified.

Definition 3. *Let $\epsilon(p)$ denote the set of extensions of a path p, that is the set of assignments to all variables in O such that each assignment is consistent with p.*

The path function for an ADD D maps a full assignment $\mathbf{x}$ to the partial assignment defined by the path traced by $\mathbf{x}$ in D and is denoted by $\Phi(D, \mathbf{x})$. The path function is represented as an ordered Binary Decision Diagram (ordered ADD with 0/1 leaves) [24] that returns 1 for all assignments consistent with the partial assignment and 0 otherwise. For an assignment $\mathbf{x}$, ADD D and $\phi = \Phi(D, \mathbf{x})$, let $\epsilon(\phi)$ denote the set of all extensions of ϕ.

We also use the "masking" operator $\oplus_C$ for ADDs, where $A \oplus_C B = (1 - C)A + CB$, for a binary valued ADD (a BDD) C. That is, the result takes the values from B if C is true and otherwise from A. This is similar to the ITE(C,B,A) notation in the BDD literature. This operation can cause merging of paths within the ADD due to reduction, e.g. if A and B agree on many values.

Two additional transformations are useful. The first converts a BDD B to an ADD D by mapping the 0-leaf in B to $-\infty$, denoted by $D = \underline{\mathrm{B}}$. The second, a complementing operation, converts an ADD D to a BDD B by mapping the 0-leaf in D to 1 and all other real-valued leaves to 0, denoted by $B = \overline{D}$.

Basically, ADDs have the same expressive power as branching programs with real-valued outputs, but can be exponentially more compact than Decision Trees. ADDs contain the more widely known Binary Decision Diagrams (BDDs) that have been applied widely in deterministic planning, model checking, motion planning, formal verification of programs and digital circuits in VLSI etc.

For the remainder of this chapter and the next chapter we use ADDs for representing P_i, R and value functions, and BDDs for representing deterministic policies. An interesting implication is that any polynomial sized BDD policies derived by our algorithms (e.g. MDPs whose optimal policy is a symmetric function, threshold function or selector function) can be synthesized into an extremely compact combinational circuit despite the large number of atomic actions [71]. However, families of boolean functions are known

that do not posses the polynomial property [21].

3.3 SPUDD : Prior State of the Art for MDPs with Factored States

The bellman backup operator requires that the value of every state be updated and this is prohibitive with 2^l states over l state variables. Symbolic Dynamic Programming (SDP) [16] uses state aggregation to avoid the cost of enumerating the states. The steps of state aggregation are interleaved with bellman backups on the aggregated states.

In practice, SDP offers the option of calculating optimal value functions for large MDPs. As shown in [66], some large factored state MDPs possess compact optimal value functions in the form of ADDs. Their SPUDD algorithm is an implementation of symbolic VI using ADDs. SPUDD is described in Figure 3.2 where the main loop implements Equation 2.1 over factored states. This algorithm implements the following equations,

$$Q^a_{n+1} = \mathcal{T}_a(V_n) = R^a + \gamma \sum_{X'_1} P^a_1(q_1) \cdots \sum_{X'_l} P^a_l(q_l) \times (V_n)' \ \forall a \in \mathbb{A} \tag{3.1}$$

$$V_{n+1} = \mathcal{T}(V_n) = \max\{Q^{a_1}_{n+1}, Q^{a_2}_{n+1}, \cdots, Q^{a_{2^m}}_{n+1}\} \tag{3.2}$$

In these equations $(V_n)'$ is the value function V_n expressed over next state variables $\mathbf{X}'$. $(V_n)'$ is obtained by renaming each interior node labeled with a current state variable X_i with the label X'_i (n.b. V_n is a function of $\mathbf{X}$ only). Equation 3.1 should be read right to left as follows: each probability ADD $P^a_i(q_i)$ assigns a probability to X'_i from assignments to q_i, introducing the variables q_i into the value ADD as a result of the ADD product. Each $\sum$ marginalization over X'_i removes the variable X'_i from the ADD. One can use the conditional independence of X' variables to push the summation of next state variables into the products. We arrive at the Q-value function for each action in the form of an ADD Q^a for each action a. Equation 3.2 maximizes the value of states over Q^a and produces an ADD equivalent to a bellman backup of V_n.

Thus, SPUDD and other works in the literature of factored state SDP such as APRICODD [137] and MADCAP [129] do not scale well with the number of action variables, because these algorithms rely on enumerating all actions, checking if each violates the action constraints A_C, and if not, computing the value of the action over factored states

Algorithm 3.4.1: SPUDD(V)

Input : ADDs P_i^a, R^a for each $a \in \mathbb{A}$
$V' \leftarrow$ Swap variables X_i in V_n with X_i'
for each action **a** $\in \mathbb{A}$
  if $A_C(a)$ is 1
    $Q^a \leftarrow V'$
    for each X_i'
      $Q^a \leftarrow Q^a \times P_i^a$
      $Q^a \leftarrow \sum_{X_i'} Q^a$
    $Q^a \leftarrow R^a + \gamma Q^a$
  else $Q^a \leftarrow -\infty$
  $V_{n+1} \leftarrow \max(V_{n+1}, Q^a)$
  Update π_{n+1} to a if $(V_{n+1} - Q^a) == 0$
Return (V_{n+1}, π_{n+1})

Figure 3.2: The SPUDD algorithm [66].

Algorithm 3.4.2: FAR(V)

Input : ADDs P_i, R with action variables
$V' \leftarrow$ Swap variables X_i in V_n with X_i'
$Q_{n+1} \leftarrow A_C \times V'$
for each X_i'
  $Q_{n+1} \leftarrow Q_{n+1} \times P_i$
  $Q_{n+1} \leftarrow \sum_{X_i'} Q_{n+1}$
$Q_{n+1} \leftarrow R + \gamma Q_{n+1}$
$V_{n+1} \leftarrow \max_{A_1,\ldots,A_m} Q_{n+1}$
$\pi_{n+1} \leftarrow \arg \max_{A_1,\ldots,A_m} (Q_{n+1})$
Return (V_{n+1}, π_{n+1})

Figure 3.3: Factored Action Regression (FAR).

using ADDs. In the next section we will define backup operators for FA-MDPs that exploit the factorization of the actions. These operators can then be iterated to derive symbolic VI, or more generally symbolic MPI algorithms.

3.4 Backup Operators for Factored Action MDPs

In the following, we use superscript as in $\mathcal{T}^Q$ to denote that actions are not maximized and we have an ADD over state and action variables so that $\mathcal{T} \equiv \max_{A_1,A_2,\ldots,A_m} \mathcal{T}^Q$. Similarly, $\mathcal{T}_\pi^Q$ restricts to a (possibly partial) BDD policy π and does not maximize over the unspecified action choice.

3.4.1 Factored Action Regression (FAR)

$$\mathcal{T}^Q(V) = A_C \times \left[R + \gamma \sum_{X_1'} P_1 \ldots \sum_{X_l'} P_l \times (V)' \right] \tag{3.3}$$

$$= R + \gamma \sum_{X_1'} P_1 \ldots \sum_{X_l'} P_l \times [A_C \times (V)'] \tag{3.4}$$

$$\mathcal{T}(V) = \max_{\mathbf{A}} \mathcal{T}^Q(V) \tag{3.5}$$

Here V is any ADD over state variables $\subseteq \mathbf{X}$ and $(V)'$ swaps each X_i in V with X_i'. Equation 3.3 should be read right to left as follows: each ADD P_i assigns a probability to X_i' from assignments to $q_i \subseteq (\mathbf{X}, \mathbf{A}, \mathbf{X}')$, introducing the variables q_i into the ADD. The $\sum$ marginalization eliminates the variable X_i'. After all primed variables are marginalized and the reward ADD is added to the result, we arrive at the Q-function that maps assignments to subsets of $\{\mathbf{X}, \mathbf{A}\}$ to real values. Finally, we multiply with the action constraints A_C, that "masks" the values of illegal action combinations to $-\infty$. Formally, paths that lead to $-\infty$ in A_C also lead to $-\infty$ in $\mathcal{T}^Q$ and these paths do not contribute to the max over actions in $\mathcal{T}$.

Proposition 1. *FAR as in Equation 3.5 is equivalent to a bellman backup.*

Proof. Recall from our formulation (Assumption 1) that for every state there exists some action that satisfies A_C. So there can be no $-\infty$ in R, V or $(V)'$. Every path p in Equation 3.3 is such that p leads to $-\infty$ if and only if the path p also leads to $-\infty$ in A_C. Equation 3.4 is equivalent to Equation 3.3 by distributive property of ADDs along with the fact that A_C is independent of any primed variables. □

In order to avoid computing any values for illegal actions, the product with A_C is pushed inside the summations. The pseudocode for FAR is shown in Figure 3.3 alongside SPUDD in Figure 3.2. In Figure 3.3, lines 3-7 include representations of V and Q that explicitly refer to action variables, and lines 8 and 9 explicitly marginalize action variables one at a time and in this way compute an exact bellman backup. SPUDD as originally proposed cannot handle constraints between actions, so in our implementation of SPUDD, we check to see if the constraint is satisfied for each action.

FAR is subtly different from SPUDD that leads to large improvements in run time in practice. FAR accounts for action variables explicitly using ADDs. In many domains with factored actions, it is efficient to compute $\mathcal{T}^Q$ as in FAR than each $Q^{\mathbf{a}}$ as in SPUDD. An important distinction is the maximization over actions. FAR uses ADD marginalization of action variables $\max_{\mathbf{A}} = \max_{A_1,\ldots,A_m}$. Thus, the maximization takes into account the structure of the ADD similar to variable elimination, and the position of action variables in the ordering of nodes in the ADD, and can be much more efficient than the max between two ADDs as in SPUDD. In addition to this, FAR computes the expectation over each state variable X_i' only once. This yields additional savings in run time, especially for those state variables X_i' whose CPD do not depend on any action variable (e.g. exogenous state variables).

The explicit use of action variables allows us to take advantage of structure in state-action space. On the other hand, action variables increase the connectivity and treewidth of the bayesian network representation. In terms of FAR, the intermediate diagrams capturing the Q-function depend on all actions simultaneously and can be more complex. Basically, we have traded the exponential time complexity of SPUDD for higher memory usage. This cost, too, can become prohibitive. In the next section we show how one can strike a more refined trade-off between SPUDD and FAR.

3.4.2 Memory Bounded FAR (MBFAR)

SPUDD and FAR are two extremes of handling factored actions in SDP - we either enumerate all actions or none at all, yielding extreme points w.r.t. time and space complexity. Roughly speaking, we expect FAR to be faster unless it exceeds the space available on the computer. In such cases we can obtain a more refined trade-off by controlling the space explicitly.

We develop this idea by analogy to recursive conditioning in Bayesian Networks (BN) [30]. Recursive conditioning is an any-space inference algorithm. In each iteration, a "cutset variable" is chosen and instantiated with every possible value, giving a set of simpler BNs that are in turn passed to an optimal inference algorithm. In our context, the cutset variables are action variables whose instantiation need not form a cutset of the BN. The cutset is chosen dynamically, with the goal that each instantiated BN (corresponding to fixed partial assignment to action variables) leads to a compact bellman backup (as

Algorithm 3.4.3: MBFAR(M, Z)

if $Z = \emptyset$
  $V' \leftarrow V' \times A_C$
else
  $V' \leftarrow$ Restrict action variables in V' according to Z (1)
for each $X'_i \in V'$
  if $size(V') > M$
    (Recusive Conditioning)
    $a_p \leftarrow$ pick action variable
    $Q^{a_p=T}, \pi^{a_p=T} \leftarrow$ MBFAR$(V', Z \cup \{a_p = T\})$
    $Q^{a_p=F}, \pi^{a_p=F} \leftarrow$ MBFAR$(V', Z \cup \{a_p = F\})$
    $Q^Z \leftarrow \max(Q^{a_p=T}, Q^{a_p=F})$
    $\pi^Z \leftarrow \left[\pi^{a_p=T} \times (Q^{a_p=T} \geq Q^{a_p=F})\right] \bigcup \left[\pi^{a_p=F} \times (Q^{a_p=T} < Q^{a_p=F})\right]$
    RETURN(Q^Z, π^Z)
  else (2)
    $V' \leftarrow V' \times Pr^Z(X'_i | X, A - Z)$
    $V' \leftarrow \sum_{X'_i} V'$ (3)
  end
end
$Q^Z \leftarrow \max_{A_1 \ldots A_m} (R + \gamma V')$
$\pi^Z \leftarrow Z \bigcup \arg \max_{A_1, \ldots, A_m} Q^Z$
RETURN(Q^Z, π^Z)

Figure 3.4: Memory Bounded Factored Action Regression (MBFAR) is initialized with $Z = \emptyset$. The maximum size M is compared to the number of nodes in V'. The set Z denotes constraints on action variables.

in FAR) of its (partial) action value function.

We start by computing a backup as in FAR, but when the memory bound is exceeded an action variable is picked and conditioned. This variable is instantiated to the values zero and one, to form two versions of the DBN with simplified CPDs for the corresponding action variable. Similarly, the partially computed FAR backup is also simplified to avoid recomputation. The remaining action and state variables are handled symbolically as in FAR, until the memory bound is exceeded. Finally, the bellman backup is computed by the maximum over the ADDs (as in SPUDD) that represent the value functions of partially instantiated actions.

The pseudo-code for the MBFAR algorithm is shown in Figure 3.4. The algorithm takes as input a partially computed value function V' in the form of an ADD over state, action and next state variables and Z specifying a partially instantiated action. Line 1 restricts action variables in V' to their assignments in Z. Then, as long as the ADDs calculated are small our algorithm behaves exactly as FAR (lines 2-3). If any intermediate stage of symbolic computation results in an ADD that has more nodes than the pre-specified space bound C, the algorithm picks an action variable a_p to instantiate, and calls the function recursively twice by adding the constraints $\{a_p = T\}$ and $\{a_p = F\}$ to Z respectively. It then finds the max of the two ADDs returned by the recursive calls and updates the policy (lines 3.4.3-2).

Importantly, the set of action variables to instantiate are decided dynamically in a recursive manner. The variable picked can be different for different instantiations of previous variables which makes the approach more flexible. In general, different choices of action variables can lead to different sizes of ADD value functions over partially specified actions. As a consequence the total number of actions backed up as ADDs also depends on the choice of the cutset variable. This method of conditioning can also alleviate bad ADD orderings wrt action variables. We used a heuristic static ordering in our implementation. We pick the variable a_p to be the action variable with the highest out-degree in the DBN.

The value functions over partially specified actions, e.g. $Q^{a_p=T}$, are similar to the generalized value functions over subsets of actions introduced by [115]. However, unlike in their work where these subsets are hand-crafted, our method is motivated by the size of the ADD and constructs these subsets incrementally.

MBFAR generalizes FAR and SPUDD in that MBFAR with zero memory is equivalent to SPUDD, and MBFAR with unbounded memory is equivalent to FAR. In principle MBFAR can be extended to use recursive conditioning on state variables as well. To increase efficiency, external memory can be utilized to store the partial value functions similar to memory based VI [40].

Symbolic planners including SPUDD have memory as the common mode of failure. MBFAR overcomes this important limitation of SDP. Empirically, the iteration of MBFAR within VI gives state-of-the-art performance in some FA-MDPs.

Using the FAR and MBFAR operators, near-optimal VI [120] and MPI algorithms can be realized [121] that are efficient for factored actions. We will first discuss the imple-

mentation of MPI using the MBFAR operator in Section 3.5 and note its shortcomings. In Section 3.6, we present a novel policy iteration approach called Opportunistic Policy Iteration with strong theoretical and practical advantages over MPI.

3.5 Factored Action Modified Policy Iteration

In this section, we introduce Factored Action MPI (FA-MPI), a Modified Policy Iteration (MPI) approach to solving FA-MDPs. MPI works by interleaving bellman backups(max over actions) with a small number of policy backups(fixed action per state). Bellman backups are computed by MBFAR. FA-MPI uses a new method for computing policy backups using MBFAR, by treating the policy as a constraint on MBFAR backups similar to the treatment of action constraints A_C in FAR.

The policy is represented as a Binary Decision Diagram (BDD) with state and action variables where a leaf value of 1 denotes any combination of action variables that is the policy action, and a leaf value of $-\infty$ indicates otherwise.

Using this representation, we perform policy backups using $T_\pi^Q(V)$ given in Equation 2 below followed by a max over the actions in the resulting diagram. In this equation, the diagram resulting from the product $\pi \times (V)'$ sets the value of all off-policy state-actions to $-\infty$, before computing any value for them and this ensures correctness of the update as indicated by the next proposition.

$$T_\pi(V) = \max_{\mathbf{A}} \pi \times \mathcal{T}^Q \tag{3.6}$$

$$= \max_{\mathbf{A}} \left[(\pi \times R) + (\gamma\pi) \sum_{X_1'} P^{X_1'} \ldots \sum_{X_l'} P^{X_l'} \times (V)' \right]$$

$$= \max_{\mathbf{A}} \left[R + \gamma \sum_{X_1'} P^{X_1'} \ldots \sum_{X_l'} P^{X_l'} \times (\pi \times (V)') \right] \tag{3.7}$$

First, it should be clear that Equation 3.6 computes policy backups. $\mathcal{T}^Q$ computes an exact backup over all actions. Those state-action combinations that do not agree are with the policy π are "masked" with a value of $-\infty$, and hence don't to contribute to the max. However, this approach would not be much different from SPI, because computing

$\mathcal{T}^Q$ is hard. But we can do better.

Proposition 2. *FA-MPI as in Equation 3.7 computes exact policy backups i.e.* $\max_{\mathbf{A}} \mathcal{T}_\pi^Q = \mathcal{T}_\pi$. *That is,* $\mathcal{T}_\pi^Q(s,a) = -\infty$ *whenever* $\pi(s,a) = -\infty$ *and* $\mathcal{T}_\pi^Q(s,a) = \mathcal{T}_\pi(s) \neq -\infty$ *whenever* $\pi(s,a) = 1$.

The policy π can be pushed inside the summations as they do not depend on state and action variables. The summation will result in $-\infty$ for any action that violates the policy and thus, the reward diagram need not be multiplied with the policy.

One important consequence of using a BDD policy representation is that π can now be a partial policy (more than one action per state). The operator $\mathcal{T}_\pi^Q$ computes the one step backup for the value of all of the actions specified by any partial policy, and the max over actions is a partial policy improvement of π. This agrees with the usual definition of $\mathcal{T}_\pi$ in the literature (Figure 2.1). The arg max can be calculated using diagram operations as $\pi_{n+1} = \overline{\left[\max_{\mathbf{A}} \mathcal{T}_\pi^Q\right] - \mathcal{T}_\pi^Q}$, where $\overline{D}$ is the complementing operator (Section 3.2). Since $[\max_A Q] - Q \geq 0$ everywhere and > 0 on paths that include suboptimal actions this identifies the greedy policy with respect to Q. Below we slightly abuse notation and denote this operation as $\pi_{n+1} = \arg\max_{\mathbf{A}} Q$.

FA-MPI retains the convergence of MPI. The policy must be initialized with the action constraint A_C, so that all policies generated by FA-MPI are consistent with A_C (and so A_C is omitted from Equation 3.7).

Memory Bounded FA-MPI (MB-MPI) is a simple extension that uses MBFAR in place of FAR for the backups in Figure 3.5. MB-MPI is parametrized by k, the number of policy backups, and M, the maximum size (in nodes) of any intermediate value function. MB-MPI generalizes MPI in that MB-MPI(k,0) is the same as SPI(k) [19] and MB-MPI(k,∞) is FA-MPI(k). Also, MB-MPI(0,0) is SPUDD [66] and MB-MPI(0,∞) is FAR [120].

While FA-MPI can lead to improvements over VI (i.e. FAR), like SPI, it can also lead to large space requirements in practice. In this case, the bottleneck is the ADD product $\pi \times (V)'$, which in this case leads to a quadratic growth of the ADD (the worst-case of ADD multiplication), due to multiplying of two ADDs that have disjoint interior nodes. We motivate the next section by viewing π as a constraint on $(V)'$, and FA-MPI strictly enforces this constraint via $\pi \times (V)'$. The strict enforcement of this constraint leads to a size-blow up. The next section uses a more conservative backup that opportunistically

Algorithm 3.5.1: FA-MPI/OPI(k)

$V^0 \leftarrow 0,\ i \leftarrow 0$
$(V_0^{i+1}, \pi^{i+1}) \leftarrow \max_{\mathbf{A}} \mathcal{T}^Q(V^i)$
while $||V_0^{i+1} - V^i|| > \epsilon$
do
- **for** $j \leftarrow 1$ **to** k
  - **do**
    - For algorithm FA-MPI : $V_j^{i+1} \leftarrow \max_{\mathbf{A}} \mathcal{T}^Q_{\pi^{i+1}}(V_{j-1}^{i+1})$
    - For algorithm OPI : $V_j^{i+1} \leftarrow \max_{\mathbf{A}} \hat{\mathcal{T}}^Q_{\pi^{i+1}}(V_{j-1}^{i+1})$
- $V^{i+1} \leftarrow V_k^{i+1}$
- $i \leftarrow i + 1$
- $(V_0^{i+1}, \pi^{i+1}) \leftarrow \max_{\mathbf{A}} \mathcal{T}^Q(V^i)$

return (π^{i+1}).

Figure 3.5: Factored Action MPI and Opportunistic Policy Iteration (OPI).

enforces policy backups while ensuring no growth in the size due to the consideration of the constraint.

3.6 Symbolic Opportunistic Policy Iteration

In this section, we develop a new policy iteration algorithm which addresses the shortcomings of FA-MPI (and MPI). The iteration in FA-MPI(k) converges to the optimal policy (value function) via a sequence of intermediate policies $\pi_1, \pi_2, \ldots, \pi_n$ (value functions). The scalability of this approach crucially depends on the compactness of intermediate value functions as ADDs, which depends on k. In addition, there is an interaction between the choice of k and the convergence rate of FA-MPI. At the level of factored states, the trade-off in FA-MPI corresponds to choosing a set of actions to regress such that the global representation of the policy (value function) is compact. In particular, FA-MPI makes the two extreme choices of regressing one or all actions. Symbolic Opportunistic Policy Iteration(OPI) addresses this issue and provides strong theoretical guarantees.

OPI applies the idea of subsumption in BDDs to the policies $\pi_1, \pi_2, \ldots, \pi_n$ generated

Algorithm 3.6.1: $\mathcal{P}(\mathtt{D}, \pi)$

if $\pi = -\infty$ **return** $(-\infty)$
if $\pi = 1$ **return** (D)
if D is a leaf **return** (D)
$\mathtt{d} \leftarrow$ variable at the root node of $\mathtt{D}$
$\mathtt{c} \leftarrow$ variable at root node of π
if $\mathtt{d} < \mathtt{c}$ in ordering
  then return $(ADD(\mathtt{d}, \mathcal{P}(\mathtt{D}_T, \pi), \mathcal{P}(\mathtt{D}_F, \pi)))$
if $\mathtt{d} = \mathtt{c}$
  then return $(ADD(\mathtt{d}, \mathcal{P}(\mathtt{D}_T, \pi_T), \mathcal{P}(\mathtt{D}_F, \pi_F)))$
if $\mathtt{d} > \mathtt{c}$ in ordering
  then return $(\mathcal{P}(\mathtt{D}, \max(\pi_T, \pi_F)))$

Figure 3.6: Pseudo-code for the pruning operator $\mathcal{P}(D, \pi) \equiv \mathcal{P}_\pi(D)$ for any ADD D and a policy constraint BDD π.

by FA-MPI. A BDD B subsumes another b if $B \Rightarrow b$. That is, the BDD B is more general than b, or a weaker premise. When the BDD represents a policy over state and action variables, the policy Π subsumes another π if its models contain every state-action mapping in the models of π, in addition to possibly some models not contained in π. Consequently, the optimal value function of Π must dominate that of π for every state in the FA-MDP because it is the maximum over a superset of actions. In the next section we introduce a new symbolic operator for finding a subsuming policy of a given policy such that the dominating value function is also more compact.

3.6.1 Pruning Operator

OPI carries over from FA-MPI that the policy is treated as a constraint on usual symbolic bellman backups. As seen in Figure 3.5, the pseudo-code OPI is identical to FA-MPI except that it uses an alternative, more conservative backup operator. This operator enforces the policy constraint *opportunistically*, i.e. only when they do not increase the size of the value function representation. In addition, it is more efficient than the corresponding operator in FA-MPI.

The pruning operator (pseudo-code in Figure 3.6) constrains some parts of the ADD

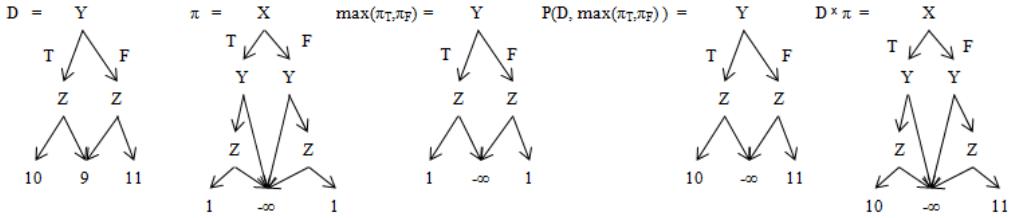

Figure 3.7: An example for pruning. D and π denote the given function and constraint respectively. The result of pruning is no larger than D, as opposed to multiplication. T (true) and F (false) branches are denoted by the left and the right child respectively.

function D with a constraint π, it assigns a value of $-\infty$ to only those paths in D all of whose extensions violate π. The operator does not alter any other paths, thus the result is not identical to $D \times \pi$ in general. The interesting case is when the root variable of π is ordered below ($d > c$ in Figure 3.6) the root of D. Thus, c not appear in D and D is more general than π, so that the only way to violate the constraint is to violate both true and false branches $\max\{\pi_T, \pi_F\}$ (equivalenty, $\exists_c \pi$). Clearly, the result of pruning depends on the ordering of variables in the ADD.

In terms of efficiency, computing $\mathcal{P}(D, \pi)$ take space and time $O(min(|D|, |\pi|))$, whereas computing $D \times \pi$ takes space and time $O(|D||\pi|)$. In addition, the ADD $\mathcal{P}(D, \pi)$ can never be larger than the ADD D. This is illustrated in Figure 3.7. Here the input function D does not contain the root variable X of the constraint, and the max under X is also shown. The result of pruning $\mathcal{P}(D, \pi)$ is smaller than D, whereas the product $D \times \pi$ is larger than D. The evaluation for the assignment $\bar{x}yz$ is 10 according to $\mathcal{P}(D, \pi)$ whereas its evaluation is $-\infty$ according to $D \times \pi$. This is the nature of mistakes made by the pruning operator in exchange for compactness. We can state the following about the result of the pruning operator.

Proposition 3. *Let $G = \mathcal{P}(D, \pi)$ then*
(P1) Every path in G is a sub-path of a path in D.
(P2) If a path p in G does not lead to $-\infty$, then for all extensions $y \in \mathcal{E}(p)$, $G(y) = D(y)$.
(P3) If a path p in G does not lead to $-\infty$, then there is an extension $y \in \mathcal{E}(p)$, s.t. $\pi(y) = 1$.
(P4) If a path p in G does lead to $-\infty$, then for all extensions $y \in \mathcal{E}(p)$ either $\pi(y) = -\infty$ or $D(y) = -\infty$.

Proof. We argue that P1 holds inductively over the recursive structure of the algorithm. Note that whenever the procedure returns it either returns a leaf or it combines a node label in D with a sub-diagram satisfying that all paths are sub-paths of D.

We next show that P2, P3, P4 hold by induction on the number of variables in the union of D and π. For the base case we have zero variables, and the algorithm returns $-\infty$ or the leaf value of D. The claim holds in both cases. For the inductive step consider D, π with $k+1$ variables and consider the cases $d < c,\ d = c,\ d > c$ as in the procedure. In each case, the recursive calls have at most k variables and therefore we can apply the inductive hypothesis on the corresponding diagrams.

Case I: $d < c$**:** Let the outputs of the recursive calls be $G_L = \mathcal{P}(\mathtt{D}_T, \pi)$ and $G_R = \mathcal{P}(\mathtt{D}_F, \pi)$. There are two sub cases to consider wrt to the operation $ADD(\mathtt{d}, G_L, G_R)$. If $G_L = G_R$ then the ADD procedure returns $G = G_L = G_R$. Consider any path p in G leading to a value v which is not $-\infty$. By the inductive assumption, for all extensions y of p we have that $D_T(y) = G(y) = v$ and $D_F(y) = G(y) = v$. Therefore, regardless of the value of d in y we have $D(y) = G(y) = v$, and P2 holds. In addition, by the inductive assumption there exists a y extending p s.t. $\pi(y) = 1$, and P3 holds. Consider next any path p in G leading to $-\infty$. By the inductive assumption, for all extensions y of p we have that $\pi(y) = -\infty$ or $D_T(y) = -\infty$ and that $\pi(y) = -\infty$ or $D_F(y) = -\infty$. Now if $\pi(y) = -\infty$ then P4 holds and if $\pi(y) \neq -\infty$ then both $D_T(y) = -\infty$ and $D_F(y) = -\infty$ and therefore, regardless of the value of d in y we have $D(y) = -\infty$ and P4 holds.

Consider next the sub case where $G_L \neq G_R$ and the ADD procedure returns a diagram G with root label d and children G_L, G_R (the children may share some sub diagrams but this does not affect the argument). Consider any path $p = (d, p')$ that follows the true branch out of the root in G and which leads to a value v which is not $-\infty$. By the inductive assumption, for all extensions y' of p' we have that $D_T(y') = G_L(y') = v$ and in particular this holds for all such y that assign $d = 1$. Therefore, for all extensions y of p where $d = 1$ we have $G(y) = G_L(y) = D_T(y) = D(y)$ where the first and last equalities hold by the structure of D and G. The argument for paths $p = (\neg d, p')$ is symmetric. This shows that P2 holds. To see that P3 holds note that by the inductive assumption there exists a $y \in \mathcal{E}(p')$ s.t. $\pi(y) = 1$, Since π does not depend on the variable d (because we are in the case $d < c$) we can set $d = 1$ in y and still maintain $\pi(y) = 1$. For P4, consider any path $p = (d, p')$ in G leading to $-\infty$. By the inductive assumption, for all extensions y of p' we have that $\pi(y) = -\infty$ or $D_T(y) = -\infty$. Therefore, for all

extensions with $d = 1$ we have $\pi(y) = -\infty$ or $D(y) = D_T(y) = -\infty$.

Case II: $d = c$**:** Let the outputs of the recursive calls be $G_L = \mathcal{P}(D_T, \pi_T)$, $G_F = \mathcal{P}(D_F, \pi_F)$. Again, there are two cases wrt the ADD construction $ADD(d, G_L, G_R)$. If $G_L = G_R$, we return $G = G_L = G_R$. Consider any path p in G that leads to a value $v \neq -\infty$. By the inductive assumption, for all extensions y of p we have $D_T(y) = G(y) = v$ and $D_F(y) = G(y) = v$. Therefore, regardless of the value of d in y we have $D(y) = G(y) = v$ for all y. This shows that P2 holds. To see that P3 holds note that by the inductive assumption there exists a $y \in \mathcal{E}(p)$ s.t. $\pi_T(y) = 1$, Now, since π_T does not depend on the variable d we can set $d = 1$ in y and obtain $\pi(y) = 1$. For P4, consider a path p in G that leads to $-\infty$. By inductive assumption, for all extensions y of p we have that $\pi_T(y) = -\infty$ or $D_T(y) = -\infty$ and that $\pi_F(y) = -\infty$ or $D_F(y) = -\infty$. Now, if $d = 1$ in y then $\pi(y) = \pi_T(y)$ and $D(y) = D_T(y)$ and P4 holds. A similar argument handles the case $d = 0$.

If $G_L \neq G_R$, we return $ADD(d, G_L, G_R)$. Consider any path $p = (d, p')$ that follows the true branch out of d, and leads to a value $v \neq -\infty$. By the inductive assumption, for all extensions y' of p' we have $G_L(y') = D_T(y') = v$ and so, when $d = 1$ in y, we have $G(y) = G_L(y) = D_T(y) = D(y)$ where the first and last equalities hold by the structure of D and G. The argument for paths $p = (\neg d, p')$ is symmetric. This shows that P2 holds. To see that P3 holds note that by the inductive assumption there exists a $y \in \mathcal{E}(p')$ s.t. $\pi_T(y) = 1$, Now, since π_T does not depend on the variable d we can set $d = 1$ in y and obtain $\pi(y) = 1$. For P4, consider a path $p = (d, p')$ in G that leads to $-\infty$. By the inductive assumption, for all extensions y' of p' we have that $\pi_T(y') = -\infty$ or $D_T(y') = -\infty$. Therefore, for $d = 1$, we have that all extensions y of p have $\pi(y) = \pi_T(y) = -\infty$ (since y sets $d = 1$) or $D(y) = D_T(y) = -\infty$.

Case III: $d > c$**:** In this case, the recursive call is $G = \mathcal{P}(D, \max_c \pi)$. For P2, consider a path p that leads to a value $v \neq -\infty$. By the inductive assumption, for all extensions y of p we have $G(y) = D(y)$ as needed. For P3 note that the path p does not include the variable d since d does not appear in either diagram of the recursive call. Now, by the inductive assumption, there is a y extending p s.t. $(\max_c \pi)(y) = 1$. This implies that at least one of π_T, π_F is satisfied by y. Setting $d = 1$ in y if π_T is satisfied and $d = 0$ otherwise, we get that $\pi(y) = 1$ as needed. For P4, consider a path p in G that leads to $-\infty$. By the inductive assumption, for all extensions y of p we have $D(y) = -\infty$ or $\max_c \pi(y) = -\infty$. In the second case, p leads to $-\infty$ in π regardless of the value of c,

and $\pi(y) = -\infty$. We therefore have $D(y) = -\infty$ or $\pi(y) = -\infty$ as needed. □

3.6.2 Effect of Repeated Pruning

In the following we will propose algorithms that apply pruning repeatedly during Bellman or policy backup in order to keep diagrams as small as possible, in particular using updates of the following form:

$$\hat{\mathcal{T}}_C^Q(V) = \mathcal{P}_C \left[\mathcal{P}_C(R) + \gamma \sum_{X_1'} \mathcal{P}_C(P^{X_1'} \dots \mathcal{P}_C(\sum_{X_l'} P^{X_l'} \times (V)') \dots) \right] \tag{3.8}$$

where C represents some constraints. Below we will use this where C is the set of action constraints, and where C is a policy. Note that this computation uses multiplication, addition, and marginalization.

To show correctness of this process we extend the notation from the previous proposition as follows. We say that diagram $\hat{A}$ satisfies P2 with respect to diagram A if for any path p in $\hat{A}$ which does not lead to $-\infty$, we have that for all extensions $y \in \mathcal{E}(p)$, $\hat{A}(y) = A(y)$. We say that diagram $\hat{A}$ satisfies P3 with respect to diagram A (and constraint C) if for any path p in $\hat{A}$ which does not lead to $-\infty$, we have that there exists $y \in \mathcal{E}(p)$, s.t. $C(y)$. We say that diagram $\hat{A}$ satisfies P4 with respect to diagram A (and constraint C) if for any path p in $\hat{A}$ which does lead to $-\infty$, we have that for all extensions $y \in \mathcal{E}(p)$ either $C(y) = -\infty$ or $A(y) = -\infty$. We next show that the properties are inductively maintained by the operations in backups.

Proposition 4. *If $\hat{A}$ satisfies P2,P4 with respect to A and $\hat{B}$ satisfies P2,P4 with respect to B then $\hat{A} \times \hat{B}$ satisfies P2,P4 with respect to $A \times B$.*

Proof. Let p be a path to a leaf valued $v \neq -\infty$ in $\hat{A} \times \hat{B}$ and let y be any extension of p. Then, by the correctness of the ADD operations we have that there exist v_1, v_2 s.t. $\hat{A}(y) = v_1$, $\hat{B}(y) = v_2$, and $v = v_1 * v_2$. Let p_1, p_2 be the corresponding paths in $\hat{A}, \hat{B}$. Since y extends both p_1 and p_2, by P2 in the assumption of the proposition we have that $A(y) = v_1$ and $B(y) = v_2$. Therefore $(A \times B)(y) = v$ and P2 holds.

For P4, if $v = -\infty$ then at least one of v_1, v_2 is $-\infty$. Assume wlog that this holds for v_1. Then by P4 in the assumption of the proposition we have that $C(y) = -\infty$, or

$A(y) = -\infty$ and as a result $C(y) = -\infty$, or $A(y) * B(y) = -\infty$. □

By changing $\times$ to $+$ in the previous proof we get

Proposition 5. *If $\hat{A}$ satisfies P2,P4 with respect to A and $\hat{B}$ satisfies P2,P4 with respect to B then $\hat{A} + \hat{B}$ satisfies P2,P4 with respect to $A + B$.*

The same holds for marginalization.

Proposition 6. *If $\hat{A}$ satisfies P2,P4 with respect to A then $\sum_x \hat{A}$ satisfies P2,P4 with respect to $\sum_x A$.*

Proof. This follows by the previous proposition because for any diagram D, we have $\sum_x D = D_{x=0} + D_{x=1}$. □

It is easy to see that the binary operations and marginalization do not always preserve property P3. However, as we show next P3 is reinforced at any application of the pruning operator.

Proposition 7. *If $\hat{A}$ satisfies P2,P4 with respect to A and $G = \mathcal{P}_C(\hat{A})$ then G satisfies P2,P3,P4 with respect to A.*

Proof. P3 is obtained directly from Proposition 3 and P2, P4 by chaining the properties from Proposition 3 with the assumptions of the current proposition. In particular, let p be a path to a leaf valued $v \neq -\infty$ in G. Then by P3 of Proposition 3 there is an extension y of p s.t. $C(y) = 1$. In addition, by P2 of Proposition 3 for all extensions y of p we have $G(y) = \hat{A}(y)$ and by P2 in the condition of the proposition $G(y) = \hat{A}(y) = A(y)$. The argument for P4 is similar. □

3.6.3 Pruned Backup Operators

Consider the update which prunes action constraints

$$\hat{\mathcal{T}}^{Q}_{A_C}(V) = \mathcal{P}_{A_C}\left[\mathcal{P}_{A_C}(R) + \gamma \sum_{X'_1} \mathcal{P}_{A_C}(P^{X'_1} \ldots \mathcal{P}_{A_C}(\sum_{X'_l} P^{X'_l} \times (V)') \ldots)\right] \tag{3.9}$$

and recall that $\mathcal{T}(V)$ is the Bellman backup as in Equation 3.5.

Proposition 8. $\max_{\mathbf{A}} \hat{\mathcal{T}}^{Q}_{A_C}(V) = \mathcal{T}(V)$

Proof. Applying the propositions inductively, we see that $\hat{\mathcal{T}}^{Q}_{A_C}(V)$ satisfies P2, P3, P4 with respect to $\mathcal{T}^{Q}(V)$. Now, by P4, for any extension y of any path leading to $-\infty$, we have $A_C(y) = -\infty$ (i.e. the action is not legal) or $\mathcal{T}^{Q}(V) = -\infty$ (i.e. the true value is $-\infty$). Therefore, legal actions whose value is not $-\infty$ are not assigned $-\infty$ value. Next note that by P3, for any path not leading to $-\infty$, there is an extension y s.t. $A_C(y) = 1$ (i.e., a legal action). Now the crucial point is that because A_C does not include state variables this holds for all states covered by p. In other words, for any path p and a state covered by this path there is a legal action covered by this path. Finally, by P2, for y not leading to $-\infty$ we have, $[\hat{\mathcal{T}}^{Q}_{A_C}(V)](y) = [\mathcal{T}^{Q}(V)](y)$ so the value on these paths is correct for the legal action. Therefore, $\max_{\mathbf{A}} \hat{\mathcal{T}}^{Q}_{A_C}(V) = \max_{\mathbf{A}}[A_C \times \mathcal{T}^{Q}(V)] = \mathcal{T}(V)$ □

It is interesting to note that if A_C was allowed to include state variables then the proof above does not hold. In this case, every path with value v has some extension y where A_C is satisfied but this might apply only to some of the states covered by the path but not necessarily all the states. If we nonetheless use the update some states could be using a value from an illegal action. This implies that if a planning domain requires state-action constraints then these much be explicitly modeled within the DBN (or enforced by multiplication) and cannot be pruned. On the other hand, if the state-action constraints are imposed by a policy, the effect is to ignore the constraint, or in other words use a non-policy action instead of the policy action when performing policy backup. As we show next, this is still legal, and yields a novel algorithm that partially enforces policy choices to improve the space requirement of the backup.

Consider the update which prunes policy constraints (where we assume that action constraints A_C are implicitly enforced)

$$\hat{\mathcal{T}}^{Q}_{\pi}(V) = \mathcal{P}_{\pi}\left[\mathcal{P}_{\pi}(R) + \gamma \sum_{X'_1} \mathcal{P}_{\pi}(P^{X'_1} \ldots \mathcal{P}_{\pi}(\sum_{X'_l} P^{X'_l} \times (V)') \ldots)\right] \tag{3.10}$$

let $\hat{\mathcal{T}}_{\pi}(V) = \max_{\mathbf{A}} \hat{\mathcal{T}}^{Q}_{\pi}(V)$ and recall that $\mathcal{T}_{\pi}(V)$ is the true policy backup as in Equation 3.6.

Proposition 9. $\mathcal{T}_{\pi}(V) \leq \hat{\mathcal{T}}_{\pi}(V) \leq \mathcal{T}(V)$.

Proof. As in the previous proposition $\hat{\mathcal{T}}_\pi^Q(V)$ satisfies P2, P3, P4 with respect to $\mathcal{T}^Q(V)$. Since we assume that action constraints have been applied, all actions are legal and therefore by P2 the upper bound $\hat{\mathcal{T}}_\pi(V) \leq \mathcal{T}(V)$ holds. Next note that by P4, every pruned path in $\hat{\mathcal{T}}_\pi^Q(V)$ is justified by π and because $\mathcal{T}_\pi^Q(V)$ enforces all the choices of π we have $\mathcal{T}_\pi^Q(V) \leq \hat{\mathcal{T}}_\pi^Q(V)$ and $\mathcal{T}_\pi(V) = \max_{\mathbf{A}} \mathcal{T}_\pi^Q(V) \leq \max_{\mathbf{A}} \hat{\mathcal{T}}_\pi^Q(V) = \hat{\mathcal{T}}_\pi(V)$. □

The same properties hold when in each step we prune both action constraints and the policy in sequence, that is, $\mathcal{P}_\pi(\mathcal{P}_{A_C}(D))$ where D is the diagram from the previous stage. In this case P3 for A_C implies that each action used for the update is legal. That is the update used in the experiments. Intuitively, the theorem can be interpreted using the notion of subsumption in BDDs. A BDD A subsumes another BDD B, denoted by $A \models B$, if every path that leads to a value of 1 in B also leads to a 1 in A. The BDD 1 (0) subsumes all (none) other BDDs. When the BDD represents a policy constraint, $\pi_A \models \pi_B$ implies that for every state, π_A allows for all the actions that π_B does (and possibly more in addition). It should be clear that $\pi_A \models \pi_B \Rightarrow V^{\pi_B} \leq V^{\pi_A} \leq V^*$. The operator $\hat{\mathcal{T}}^\pi(V)$ does exactly this. Given a policy π, $\hat{\mathcal{T}}^\pi(V)$ returns an ADD V such that there exists a policy π' with $\hat{\mathcal{T}}^\pi(V) = \mathcal{T}^{\pi'}(V)$.

3.7 Experiments

In this section, we experimentally evaluate each combination of algorithm and operator in three probabilistic planning problems with factored actions. Two of the three domains we test were part of the probabilistic planning benchmarks of 2011 (IPPC-2011) viz. SysAdmin and Elevator Control. These domains have high branching factors and number of actions which pose significant challenges for both symbolic [121] and sub-symbolic methods [86]. Our experiments highlight the benefits of the symbolic MPI approach over the state of the art symbolic VI approach SPUDD. To the best of our knowledge, this is also the first empirical demonstration of Symbolic MPI (almost 20 years after being first proposed) and its adaptation for factored actions.

3.7.1 Domain Descriptions

The following domains were described using the Relational Dynamic Influence Diagram Language (RDDL) [128]. We ground the relational description to arrive at our proposi-

tional DBN-MDP similar to Figure 2.2. RDDL also allows the specification of constraints which we use to bound the number of parallel actions. The number of parallel actions is the action constraint A_C in our algorithms and determines the size of the action space. The action constraint allows us to test the scaling of our algorithms, and the state-of-the-art symbolic VI algorithm SPUDD, as a function of the action space keeping the state space fixed.

We also use RDDL constraints to specify states that can never occur. State constraints are used to generate an initial state for execution of any policy, from which an invalid state cannot occur as long as the policy obeys action constraints. Our algorithms that use pruning (viz. OPI and MB-OPI) may output a non-trivial value for invalid states, but these values cannot propagate to valid states (Assumption 1). Note that FA-MPI and MB-MPI output a fixed value of $-\infty$ for invalid states.

3.7.1.1 Inventory Control (IC)

The first domain we consider is a simple Inventory Control problem from Operations Research. This domain consists of n independent shops, with two state variables per shop denoting whether the shop is full $empty(shop)$, and whether there is a customer in the shop $person(shop)$. Each shop can be filled by a deterministic action $fill(shop)$ that sets $empty(shop)$ to false, and incurs a cost of -0.35. The total number of shops that can be filled in one time step is restricted using the action constraint which can be interpreted as a fixed number of "trucks" available. The rate of arrival of a customer is distributed independently and identically(IID) for all shops as $Bernoulli(p)$ with $p = 0.05$, and so this domain is the simplest domain we consider. A customer at an empty shop continues to wait with a reward of -1 for each time step until the shop is not empty, at which time the shop is emptied and the person leaves. The reward function is the sum over the rewards at each shop. An instance of IC with n shops and m trucks has a joint state and action space of size 2^{2n} and $\sum_{i=0}^{m} \binom{n}{i}$ respectively.

3.7.1.2 SysAdmin : System Administration Domain

The "SysAdmin" domain was part of the IPPC 2011 benchmark and was introduced in earlier work [57]. The SysAdmin problem has several applications in spatial planning

problems e.g. spreading of disease or fire in a forest, the migration of birds, spreading of invasive plant species [37], fault tolerance in large computer networks etc.

A SysAdmin problem instance consists of a network of n computers connected in a given topology. The topology defines the spatial relations and thus the hardness of the planning problem. Each computer is either running (reward of +1) or failed (reward of 0) so that $|S| = 2^n$, and each computer has an associated deterministic action of rebooting (with a cost of -0.75) so that $|A| = 2^n$. The objective is to maximize the uptime of computers. Unlike the previous domain, the exogenous events viz. the crashing of computers are not independent of one another. A running computer that is not being rebooted is running in the next state with probability p proportional to the number of its running neighbors, where $p = 0.45 + 0.5\left(\frac{1+n_r}{1+n_c}\right)$, n_r is the number of neighboring computers that have not failed and n_c is the number of neighbors. We restrict the number of computers that can be rebooted in one time step using an action constraint. We test this domain on three topologies of increasing difficulty, viz. a star topology, a unidirectional ring and a bidirectional ring.

3.7.1.3 Elevator Control

This domain was also part of the 2011 planning benchmarks. The problem of automatically controlling elevators in a building has long been studied in operations research reinforcement learning [7]. The objective is to control m elevators in a building with n floors. A state is described as follows: for each floor, whether a person is waiting to go up or down (2 bits per floor); for each elevator, whether a person inside the elevator is going up or down (2 bits per elevator), whether the elevator is at each floor (1 bit per elevator per floor), and its current direction (up or down). A person arrives at a floor f, independently of other floors, with a probability $Bernoulli(p_f)$, where p_f is drawn from $Uniform(0.1, 0.3)$ for each floor. Each person gets into an elevator if it is at the same floor and has the same direction (up or down). Ties between multiple elevators that are available at the same floor are broken deterministically using a total ordering among elevators. A person going up exits the elevator at the top floor (a person going down exits at the bottom floor).

In this domain we can describe the dynamics of elevators compactly using a clever encoding of action variables. Each elevator has three actions: $move-up$ or $move-down$

by one floor, or $flip$ its direction. The effects of these actions are deterministic and, more importantly assumed to be independent of the other actions ie. $move - up$ sets the location of the elevator to be set for the floor above the current floor and unset at the current floor. So executing $move - up$ and $move - down$ simultaneously causes an erroneous state due to this independence assumption (see Assumption 1). These invalid actions are prevented from being part of any policy by using the action constraint that only one action is allowed per elevator. This encoding allows for a compact DBN and avoids the use of synchronic arcs between the locations of elevators.

Each person receives a reward of -1 when waiting at a floor and -1.5 if he is in an elevator that is moving in a direction opposite to his destination. There is no penalty if their directions are the same. The reward function is the sum of rewards over each elevator and floor. So the optimal policy will prefer people waiting within the elevator rather than at any floor, as long as the elevator does not move in the opposite direction. This is a complex domain, requiring coordination among elevators, and is generally out of the reach of an exact solution, but we are able to illustrate the difference between the algorithms on a small problem instance.

3.7.2 Experimental Setup

In our experiments the variables in the ADDs are ordered so that $parents(X_i')$ occur above X_i' and the X_i's are ordered by $|parents(X_i')|$. We heuristically chose to do the expectation over state variables in the top-down way, and maximization of action variables in the bottom-up way with respect to the variable ordering. In order to evaluate scaling with respect to the action space we fix the size of the state-space and measure the total CPU time[1] to convergence (Bellman error less than 0.1 with a discount factor of 0.9). The charts denote OPI with k steps of evaluation as OPI(k), and MB-OPI with memory bound M as MB-OPI(k, M)(similarly FA-MPI(k) and MB-MPI(k, M)). In addition, we compare to symbolic value iteration: the well-established baseline for factored states, SPUDD [66], and factored states and actions FAR-VI (or FA-MPI(0)) [120]. Since both are variants of VI we will denote the better of the two as VI in the charts.

We used our own Java implementation of ADDs and symbolic operations between

[1]These experiments were run on a single core of an Intel Core 2 Quad 2.83GHz with 4GB limit on memory usage.

ADDs. The CPU times shown here include the time spent waiting for the automatic garbage collector. An important aspect of this implementation is that it uses a global LRU cache of fixed size for ADD *apply* operations, that stores the results of operations between ADDs. This improves the runtime performance of SPUDD because, even though SPUDD completely ignores action structure, some backups are shared indirectly through the cache, and reduces the difference between SPUDD and FAR-VI. The reader is directed to [120] for a comparison of SPUDD and FAR/MBFAR-VI when a global cache is not used. Those results are not replicated here as our focus is mainly on MPI.

3.7.3 Experimental Validation

The experiments are organized as follows. First, we will compare the effectiveness of $MPI(k)(k > 0)$ for three different values of k. In our experiments, and in theory, OPI is always better than FA-MPI so we compare OPI directly.

Representation compactness : To motivate our MPI approach to these MDPs, we first show that the optimal policy has a much more compact ADD representation over the optimal value function. The ADDs are not able to compactly represent the additive structure of the optimal value functions in these domains, a well-known limitation of ADDs that is circumvented by our MPI approach. To illustrate this, Table 3.1 shows the compression provided by representing the optimal value functions and policies as ADDs versus tables. We observe orders of magnitude more compression for representing policies. The compression ratio for value functions is less impressive and surprisingly close to 1 for the Uniring domain.

Domain	# parallel actions						# parallel actions					
	Compression in V						Compression in π					
	2	3	4	5	6	7	2	3	4	5	6	7
IC(8)	0.06	0.03	0.03	0.02	0.02	0.02	0.28	0.36	0.35	0.20	0.09	0.03
Star(11)	0.67	0.58	0.50	0.40	0.37	0.35	$1.8e^{-4}$	$2.3e^{-4}$	$2.1e^{-4}$	$1.9e^{-4}$	$1.4e^{-4}$	$9.6e^{-5}$
Biring(10)	0.96	0.96	0.95	0.94	0.88	0.80	$1.1e^{-3}$	$1.3e^{-3}$	$1.2e^{-3}$	$1.1e^{-3}$	$9.8e^{-4}$	$7.4e^{-4}$
Uniring(10)	0.99	0.99	0.99	0.99	0.99	0.99	$9.3e^{-4}$	$1e^{-3}$	$9.4e^{-4}$	$8.2e^{-4}$	$5.2e^{-4}$	$2.9e^{-4}$

Table 3.1: Ratio of size of ADD (nodes) to tabular representation of V^* and π^*.

Impact of policy evaluation : We compare OPI(0), OPI(2) and OPI(5) in Figure 3.8. OPI(0) is equivalent to FAR-VI, the state-of-the-art symbolic VI algorithm for factored actions. The charts are shown in Figure 3.8.

First, in an Inventory Control problem with 8 shops (65536 states and 256 actions),

as the number of parallel actions increases, OPI(0) takes increasingly more time for parallel actions 1-4. The time to converge reduces with more parallel actions due to the (a) increased sparsity of the domain dynamics under any policy viz. filling more shops causes fewer shops to be empty. (b) increased structure in the value function viz. all states that have t shops empty have the same value since most of them can be filled in one time step. An increase in the steps of evaluation in OPI(2) and OPI(5) leads to a speedup of upto 3x on the hardest problem.

In the SysAdmin domain, we tested three different topologies. For all the topologies, as the size of the action space increases, OPI(0) (i.e. VI) takes an increasing amount of time, so we stopped the planner after six hours. The bellman error is annotated in the Figure at termination. OPI scales significantly better and does better with more steps of policy evaluation, suggesting that more lookahead is useful in this domain.

In an Elevator Control problem (about 4 million states and 64 actions) OPI(0) converges the fastest, while OPI(2) and OPI(5) have similar convergence rates. This result suggests that lookahead is not crucial in this domain, but it may also be due to the small size of the building in this problem instance.

Impact of pruning : Here we will compare OPI vs. FA-MPI. Although OPI is proven to be no worse than FA-MPI, these experiments demonstrate that OPI significantly outperforms FA-MPI in terms of run-time, whereas FA-MPI blows up in memory in some cases (EML stands for Exceeded Memory Limit in the charts). These charts are shown in Figure 3.9.

In Inventory Control, FA-MPI(5) exceeds the memory limit on five out of the seven problem instances. The bellman error is annotated for these instances showing that FA-MPI becomes worse with more actions. On the other hand, OPI(5) converges in all instances. In the SysAdmin domain, we show the scaling of OPI vs FA-MPI for problems that increase in state and action space. The chart shows the relative time FA-MPI takes more than OPI with increasing number of parallel actions. FA-MPI takes increasingly more time more than OPI. On the largest problem, FA-MPI exceeds the memory-limit, and is at least 150% slower than OPI. In Elevator control, FA-MPI exceeds the memory limit and is at least 250% slower, whereas OPI converges much faster than FA-MPI.

Impact of memory-bounding : Even though memory bounding can mitigate the memory blowup in FA-MPI, it can cause some overhead in time, and can still exceed the limit due to exact policy backups. These experiments show how MB-MPI compares

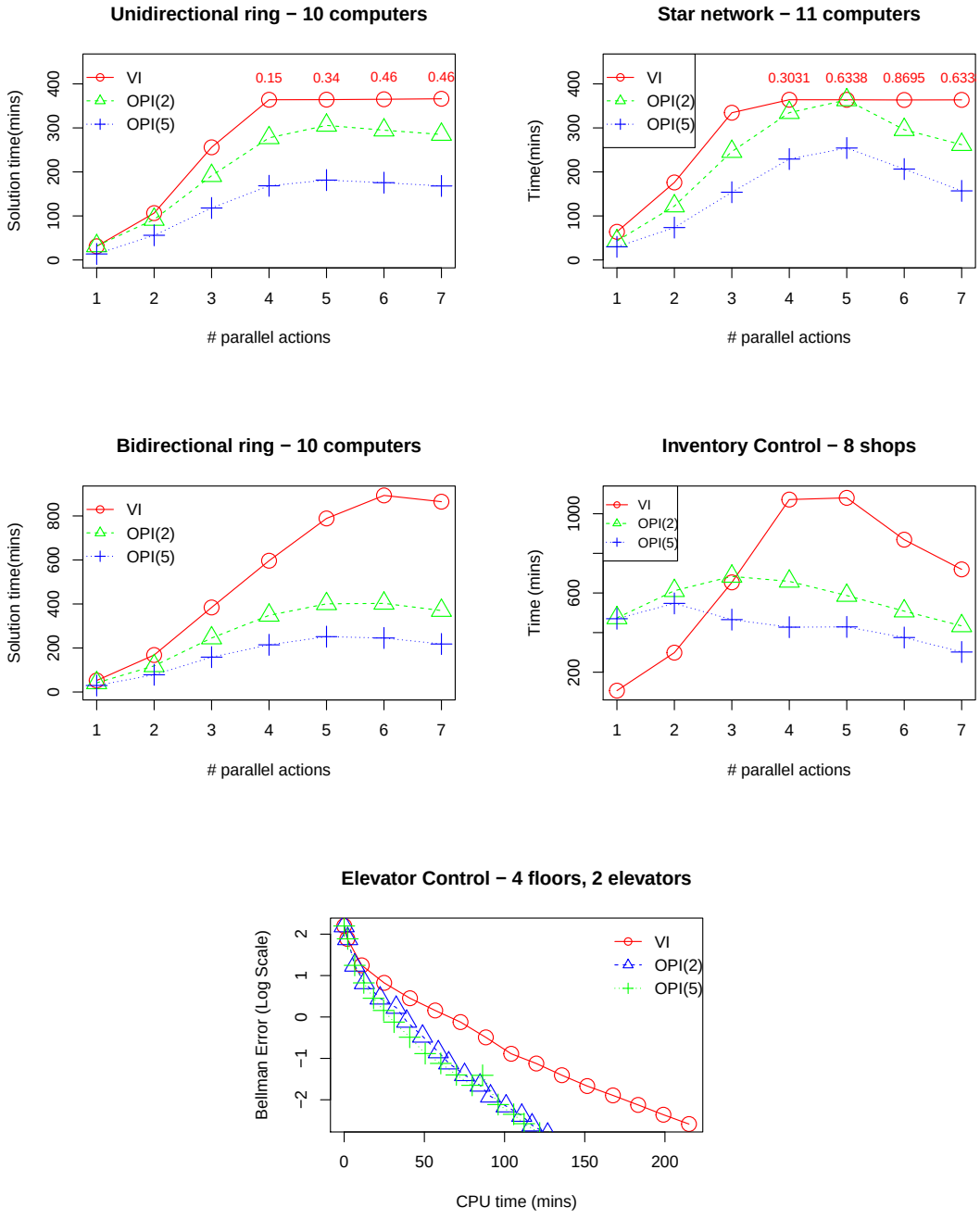

Figure 3.8: Impact of Steps of Policy Evaluation. EML denotes Exceeded Memory Limit.

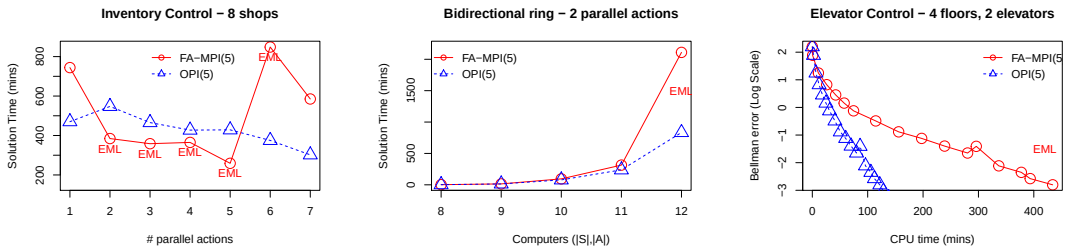

Figure 3.9: Impact of Pruning : OPI vs FA-MPI. EML denotes exceeded memory limit.

with OPI and MB-OPI, and whether the combination of memory bounding and pruning is useful. The charts are shown in Figure 3.10.

In the Inventory Control problem, we already saw that FA-MPI exceeds memory limit in five out of seven instances. MB-MPI mitigates this and reduces the number of EML to two out of seven. On the other hand, OPI can solve all the seven instances and its combination with MB as in MB-OPI provides some speedup. In the SysAdmin domain, the figure shows a comparison between MB-MPI and MB-OPI on a particular problem instance. In this instance, FA-MPI exceeds memory while MB-MPI does not, but is outperformed by MB-OPI. A similar plot is seen in the elevators domain for MB-MPI vs MB-OPI. Thus, the combination of MBFAR and OPI seems to be new best planner for symbolic planning in these domains.

Impact of action variables : Next, we compare our MDP formulation with action variables and the operators FAR against the state-of-the-art symbolic VI algorithm SPUDD. Figure 3.11 show this comparison for the Inventory and Elevator Control domains. As mentioned earlier, since our focus here is mainly on MPI, our implementation uses a global cache that obscures the difference between SPUDD and FAR-VI. See [120] for a detailed comparison of SPUDD, FAR-VI and MBFAR-VI for ADDs and Affine ADDs.

3.8 Discussion

In this chapter we presented symbolic domain-independent near-optimal algorithms for planning in MDPs that have factored states and actions. Our algorithms do not make

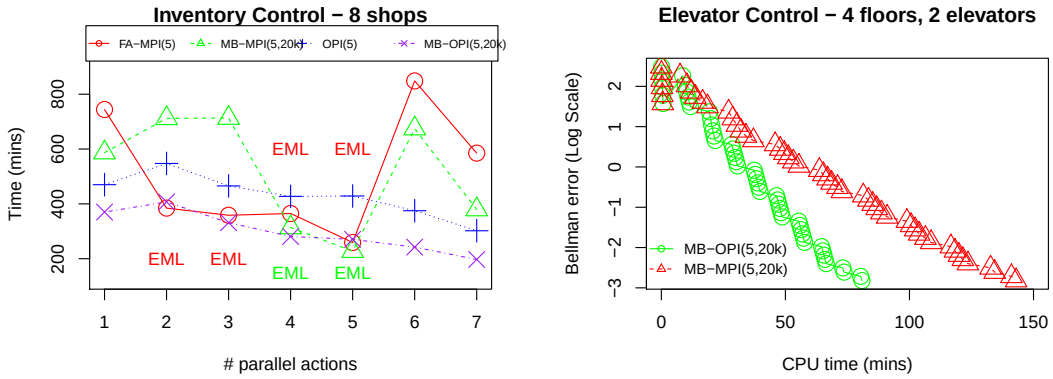

Figure 3.10: Impact of Memory Bounding : MB-MPI vs MB-OPI.

any assumptions on the MDP dynamics or reward and produce a policy over all states. Memory-Bounded OPI (MB-OPI) scales well with the number of actions and outperforms the state-of-the-art symbolic planners. Next we discuss some works in the literature that tackle the planning problem in large action spaces.

There are several approaches addressing factored action MDPs, but unlike our work most of these works do not attempt to compute an exact optimal policy or value function over all states. For example, prior work for goal-based problems with specified initial states [94, 152], does not deal with arbitrary rewards, and does not compute the policy over all states, or guarantee optimality. The work of [98] extends to general rewards but still assumes an initial state and does not compute a policy over the entire space or guarantee optimality. The approach of [57] considers general MDPs and computes policies over the entire space, but is based on function approximation relative to a set of hand-engineered features, which requires additional knowledge and does not guarantee optimality.

Our work in this chapter is related to work on model-minimization for factored action spaces [82]. Like our work, that approach leverages decision diagrams in order to compactly represent policies and value functions, but in a very different way. The approach pre-processes the MDP description in order to compute a "homogeneous" partition of the state-action space that makes all distinctions necessary for representing the value function of any policy. These partitions are then provided to an explicit MDP solver

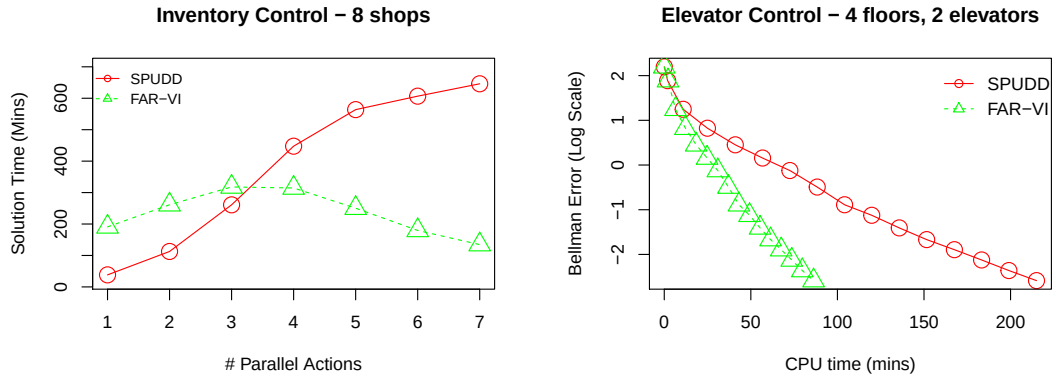

Figure 3.11: Impact of action variables.

to find a policy. In contrast, SDP interleaves partitioning with steps of planning (i.e., regression) and thus only needs to make distinctions necessary for the value functions encountered during planning. For this reason, the model-minimization approach, by its very nature, is less efficient than SDP in general [53].

Another common way to deal with FA-MDPs in practice is to convert it to a factored-state MDP (such as [80]). This involves treating the action variables as part of the state description, here called "pseudo state variables". If there are k action variables, each concrete action is split into k pseudo-actions, wherein one pseudo-state variable is set or unset. At the end of k pseudo-actions, we can extract a joint action from the pseudo-state variables and execute it. This method is unsatisfactory for several reasons. Firstly, it increases the size of the state space by a multiplicative factor of 2^k, an undesirable change because many planners have a polynomial dependence on the size of the state space. Secondly, it increases the horizon of the decision making problem by a multiplicative factor of k, undesirable for planners whose quality depends exponentially on the horizon (e.g., Monte-Carlo methods). Action variables are different by their very nature, and need different solution strategies.

Several promising directions are available for future work in factored action spaces. There is very little research addressing factored actions, and even the best planning algorithms using Monte-Carlo Tree Search fail in the face of large action spaces with a large number of successor states for each action. Our algorithms are able to successfully

tackle these problems, but the scalability for an arbitrary problem is not clear.

Existing techniques that address scalability of symbolic planners can readily be applied within our algorithms. For example, Affine ADDs offer a more compact representation over ADDs and may be more scalable. Earlier work [120] explored AADDs in the context of VI in factored action spaces. Similarly, the simple APRICODD approximation [137] for ADDs can be readily applied to MB-OPI to get approximate MPI. An important part of scalability of classical planners is the use of relaxations of the domain description, and there is no such work on relaxation for factored state and action MDPs. Coming up with principled relaxations and corresponding symbolic algorithms is also future work.

Chapter 4: Memory-Efficient Anytime Planning using Symbolic Dynamic Progamming

4.1 Introduction

The success of online planning in Markov Decision Processes (MDPs) depends crucially on the extent to which the information gathered from search is generalized to unseen states. In the absence of generalization and heuristic guidance, the planner must explore the entire reachable state space. In factored MDPs, the size of the state and action spaces is exponential in the number of state and action variables, causing algorithms that are polynomial in the number of states and/or actions to be impractical.

The current state-of-the-art online algorithms based on e.g. , Real-Time Dynamic Programming (RTDP) [8] and UCT [84], search in terms of atomic or "flat" states. They are unable to take advantage of the factored structure present in the MDP descriptions which allows strong generalization among states [18]. For example, these planners are not able to identify irrelevant state variables (state features) in factored MDPs, e.g. in the presence of exogenous events the value of a state depends only on the most important request.

In contrast, symbolic decision theoretic planners, such as SPUDD [66], do take advantage of the factored structure of transitions and rewards among state and action variables. These algorithms interleave Dynamic Programming (DP) [13] updates with steps of model minimization [53] in a selected representation such as Algebraic Decision Diagrams (ADD) [5]. As seen in the previous chapter, the factored MDP model is compiled into the representation ADDs as preprocessing and a (near optimal) value function is symbolically deduced within this representation, e.g. SPUDD a symbolic VI algorithm for factored state MDPs, later extended to factored state and action MDPs[120]. These offline planners sometimes scale to large MDPs, but depend on compactly representing the optimal value function of the entire MDP [66]. Due to this requirement, these algorithms exceed practical memory and time limits in many problems of interest. When they do work it is because the optimal values generalize over many states leading to the

compact ADD value function.

Symbolic Real-Time Dynamic Programming (sRTDP) [44, 46] aims to combine the benefits of the symbolic methods and online planning by incorporating symbolic state generalization into the computation of the online planner. This effectively imposes state constraints capturing reachability from the current world state into the symbolic computation. However, sRTDP is a general framework, and its performance is sensitive to a prespecified definition of generalized states that maps flat states to generalized states. Existing definitions in prior work lead to algorithms that cannot tradeoff the coarseness of the generalization with the memory consumption. Due to this, despite the aim for generalization the resulting planner is often inferior to the corresponding RTDP algorithm working in the flat state space, and in some cases simply blows up in memory without failing gracefully to a flat search method.

Our contribution is the introduction of new symbolic generalization operators that guarantee a more moderate use of space and time, while providing non-trivial generalization. Using these operators, we present symbolic online planning algorithms that combine forward search from an initial state with backwards generalized DP updates. The symbolic updates are applied to *generalized* states. The first algorithm, Path Dynamic Programming (PDP) (Section 4.3.1), samples fixed-length trajectories by acting greedily and refines *one path* in an ADD for each visited state. It uses either an operator based on value invariance (PDP-V) or one based on policy invariance (PDP-π). Both operators yield anytime algorithms that guarantee convergence to the optimal value and action for the current world state, while maintaining bounded growth in the size of the symbolic representation.

In spite of the slow growth of the value function representation, intermediate computations in PDP leading to that representation can potentially exceed memory limits. This motivates our second operator that performs a more careful control of space in its generalization, in that it takes a more incremental approach to ADD path refinement. The resulting planning algorithm, Pruning Path Dynamic Programming (pPDP) (Section 4.4), applies the pruning procedure of [121] to control the size of intermediate results. The algorithm is convergent and provides generalization only when it does not increase space requirements compared to flat state search. Thus, it is guaranteed not be worse than flat state space search. It is the first symbolic algorithm to yield a sound generalization while guaranteeing not to use more memory than flat RTDP.

We empirically demonstrate (Section 4.5) the performance of PDP and pPDP on three benchmark domains from the recent International Probabilistic Planning Competitions (IPPC), where the proposed algorithms show significantly better anytime performance than previous previous symbolic and sub-symbolic methods. We illustrate some compact generalized states found by our approach to show the context-specific nature of these domains and the presence of irrelevant state variables that our approach is able to exploit.

4.2 Symbolic Online Planning

In order to facilitate the presentation of online symbolic methods we next consider an update that explicitly controls which states are updated. Let X be a BDD representing some set of states, and let $\underline{X}$ be the corresponding constraint ADD mapping states in X to 1, and states not in X to $-\infty$. The operator $\mathcal{B}^*(V, X)$ performs an exact update (a Bellman backup) for the values of states in X and copies the values from V for other states using $\oplus$. This operator can be implemented via the ADD expression:

$$\mathcal{B}^*(V, X) \stackrel{\Delta}{=} V \oplus_X [\max_A (R + \gamma E_{X'_1} \dots E_{X'_l}(\underline{X} \times V'))] \tag{4.1}$$

where multiplication by $\underline{X}$ is not necessary for correctness but it helps control space. The product of V' with $\underline{X}$ fixes the value of states that are not in the set X to $-\infty$. Therefore, the sum and product operations also result in $-\infty$ without increasing the size of ADDs for these states. The constraint $\underline{X}$ can be pushed inside the summations due to the distributive property of ADD operations (Chapter 3). Note that sRTDP uses an operator equivalent to $\mathcal{B}^*(V, X)$ via a more memory intensive ADD expression.

We can now explain more general algorithms. Let X denote a set of states or "a generalized state" and s denote an atomic state or "flat state". FAR (Equation 3.5) uses the backup of Equation 4.1 with $X = 1$, which means it updates the values of all states.

Real-Time Dynamic Programming (RTDP) [8] is an online planning algorithm that only updates the values of reachable states. RTDP works by simulating trajectories from a starting state and setting $X = s$ for each encountered flat state s. While each update is time and space-efficient, convergence can take a long time in factored MDPs.

sRTDP [44] generalizes RTDP updates, uses an update similar to Equation 4.1 by

setting X to an arbitrary set of states. The set X is defined by an equivalence relation over states (e.g. the bisimulation relationship [53]), which is in practice, heuristically chosen to trade off the efficiency of the update with the benefit of generalization. An unwise choice of X in Equation 4.1 can lead to unreasonable space (or time) requirements. Despite its goal of generalization, the performance of sRTDP can be inferior to RTDP in the online setting where both space and time are limited.

Next we describe our formulations of generalized states X that lead to efficient updates both in time and space. Convergence of RTDP (and sRTDP) can be retained if X includes state s. Efficiency comparable to RTDP can be achieved if the generalized value functions and policies can be captured with about the same amount of memory. We give equivalence relationships that are more restricted than bisimulation [92], and lead to efficient symbolic online algorithms.

4.3 Path Dynamic Programming

Our algorithms are instantiations of Trial-Based Real Time Dynamic Programming (RTDP) [8, 81] with a particular generalized backup function and a fixed trial length. They have two parameters : a lookahead integer $H > 0$ that is the length of trajectories and a real valued ε that controls approximation error in the values of states.

In contrast to most online planners which use a tabular representation, we maintain one value ADD $V^d, d \in [0, H-1]$ per level of the lookahead tree. We chose this over a global ADD (as in sRTDP) because it allows representing non-stationary policies and value functions compactly, as well as allowing different levels of approximation per level, e.g., for increasingly coarse representations of the future . In addition, to simplify the presentation, we explicitly maintain a policy BDD π^d, for each level d.

Our algorithms start from an initial state and sample a trajectory $\langle s_0, a_0, \ldots, a_{H-2}, s_{H-1} \rangle$ by following the greedy policy $\pi^0, \ldots, \pi^{H-1}$. Then, the ADDs are updated in the backward direction: V^{H-1} is updated from the ADD 0, V^{H-1} is used to update V^{H-2} and so on till V^0, which includes s_0 but may be more general.

The general pseudocode for all the algorithms is shown in Figure 4.1. They have an update of the form $V = V \oplus_M \max_{\mathbf{A}} Q$. The algorithm requires three properties: (A) M is a *path* over state variables (hence the name Path Dynamic Programming), (B) Q is an ADD over state and action variables with updated values for a super-set of the states

Algorithm 4.3.1: (ADDs $V^0, \ldots, V^{H-1}$, $\pi^0, \ldots, \pi^{H-1}$)

Initialize each $V^i \leftarrow (H - i + 1)R_{\max}$, $\pi^i \leftarrow$ NoOp.
Sample trajectory $\langle s_0, a_0, \ldots, a_{L-1}, s_L \rangle$ using π.
for $i \leftarrow L - 1$ **downto** 0

do
- **if** PDP-V then $(Q, M) \leftarrow$ Equations 4.3, 4.4
- **if** PDP-π then $(Q, M) \leftarrow$ Equations 4.5, 4.7
- **if** pPDP then $(Q, M) \leftarrow$ Equations 4.8, 4.10
- $V^i \leftarrow V^i \oplus_M \max_{\mathbf{A}} Q$
- $\pi^i \leftarrow \pi^i \oplus_M \arg\max_{\mathbf{A}} Q$

if beyond time or trajectory budget
then return $\pi^0(s_0)$

Figure 4.1: Pseudocode for Path Dynamic Programming (PDP). $A \oplus_X B = (1 - X)A + XB$.

in M. (C) The current state s_i is included in path M.

Proposition 10. *Any instance of the PDP algorithm (Figure 4.1) satisfying properties B and C converges to the optimal value (and action) for* s_0.

Proof (sketch): The proof directly follows from the convergence of Trial-Based RTDP [8]. First, it can be shown that PDP maintains the invariants $V_i \geq V_i^*$ for all i. Second, it uses greedy action selection to sample trajectories and each trajectory always includes the state s_0. Hence if each update is equivalent to DP update on some states, the value and policy at s_0 converge to their optimal values. □

4.3.1 PDP-V

The main idea for PDP is to restrict the update to one path in the ADD, instead of one state in RTDP, and PDP-V uses one path in the value ADD. However, this requires a

careful control of the set M as shown below.

$$\mathcal{B}(V, s) = V \oplus_M \max_{\mathbf{A}} Q \tag{4.2}$$

$$Q = R + \sum_{X'_1} P_1 \times \ldots \sum_{X'_l} P_l \times (\underline{\Phi(V, s)} \times V') \tag{4.3}$$

$$M = \Phi(\max_{\mathbf{A}} Q, s) \wedge \Phi(V, s) \tag{4.4}$$

For a given state s and value ADD V, the values of all states that are extensions of the current path $\Phi(V, s)$ are updated in the ADD Q (Equation 4.3). The ADD $\max_{\mathbf{A}} Q$ has updated values for the path extensions of $\Phi(V, s)$ and $-\infty$ otherwise. But using this update with $M = \Phi(V, s)$ might lose compactness if many of the extensions of $\Phi(V, s)$ have different values. Additionally, the new path $\Phi(\max_{\mathbf{A}} Q, s)$ can be shorter than $\Phi(V, s)$ whereas only the extensions $\Phi(V, s)$ have correctly updated values. Sound generalization and compactness are both achieved by restricting the update to the path refinement of $\Phi(V, s)$ by setting M to be the intersection of the set of states that share the same path as state s before and after the update. This is the main difference between PDP and sRTDP. It guarantees that the updated V has the same number of leaves as the flat state update, an important guarantee for a symbolic method.

Proposition 11. *Let V be an ADD, s a state, $W = \mathcal{B}(V, s)$ and M the path according to Equation 4.4.*
(a) For all states $q \in \epsilon(M)$, $W[q] = \mathcal{B}^*(V, q)[q]$.
(b) W grows by at most one leaf node over V.

Proof (sketch): (a) follows because ADD Q is a sound update for states in $\Phi(V, s)$ (because the constraint $\Phi(V, s)$ can be pushed inside the summation (Chapter 3). The BDD M represents a subset of states that satisfy $\Phi(V, s)$ due to Equation 4.4[1]. (b) is true because the path M leads to a leaf in $\max_{\mathbf{A}} Q$. For paths in $1 - M$ the values are copied from V and do not add leaves to W. □

The first part of Proposition 11 guarantees the convergence of PDP-V according to Proposition 10. In practice, it is observed that the paths in symbolic VI often remain unchanged between consecutive iterations while the values have not converged. PDP-V

[1]Note that the proposition does not hold for the path $\Phi(W, s)$ (rather than M) due to the $\oplus$ operator which might merge an updated path with a path that was not updated.

updates these efficiently and succinctly, gaining a speedup proportional to the number of states in the path. However, in order to find the mask M in PDP-V we have to calculate updated values for all extensions of $\Phi(V, s)$ in Equation 4.3, and in some cases this preparatory step exceeds memory limits. Section 4.4 gives an algorithm that does not have this disadvantage.

4.3.2 PDP-π

PDP-π similarly restricts the update to one path. However, is appeals to the notion of policy irrelevance [76, 92, 67], that captures states having the same optimal action. Recall that PDP maintains a policy representation in addition to the value ADDs. PDP-π updates states that share a path in π before and after a DP update to the policy. The memory efficiency of path refinement is retained with respect to the policy representation.

PDP-π starts with a trivial policy (e.g. NoOp) and refines the policy for generalized states visited by trajectories. In this way, PDP-π behaves more like a policy search method. It is well known that in some cases paths in the policy BDD remain unchanged during iterations of symbolic VI even though the values keep changing. PDP-π captures these succinctly (as shown empirically in Section 4.5).

Let $\pi(s)$ be the policy action, i.e., a complete assignment to action variables for state s according to BDD π, $\pi(s) = \arg\max_{\mathbf{A}} \pi_{\downarrow s}$. In case of a tie, some action variables are set to false (including the case when they are unspecified by $\pi_{\downarrow s}$). Let $\Phi_\pi(s)$ be the path over state variables according to the greedy action in π, $\Phi_\pi(s) = \Phi(\pi, s \wedge \pi(s))$. The update is similar to PDP-V except it uses Φ_π and arg max instead.

$$Q = (R + \sum_{X'_1} P_1 \times \ldots \sum_{X'_l} P_l \times (\underline{\Phi_\pi(s)} \times V')) \tag{4.5}$$

$$\mu = \arg\max_{\mathbf{A}} Q \tag{4.6}$$

$$M = \Phi_\mu(s) \wedge \Phi_\pi(s) \tag{4.7}$$

The ADD Q contains updated values for the states in $\Phi_\pi(s)$ rather than $\Phi(V, s)$ as in PDP-V. The mask M uses $\mu = \arg\max_{\mathbf{A}} Q$ to denote the greedy policy BDD extracted from Q. Finally, the policy BDD is updated as $\pi = \pi \oplus_M \arg\max_{\mathbf{A}} Q$. Clearly, the property in Proposition 11 (a) holds here as well and therefore by Proposition 10 the

algorithm PDP-π converges.

4.4 Pruning Path Dynamic Programming

The idea in pPDP is to repeatedly prune the intermediate ADDs of Equation 4.3 so that the ADD Q has space complexity no larger than the DP update of a flat state. We use the pruning operator proposed in Chapter 3 to control the size of the ADD. Briefly, the pruning operator denoted by $\mathcal{P}(D, C)$ for an ADD D and a constraint C represented as a BDD returns an ADD which is no larger than D. The result of pruning removes some of the paths from D that violate the constraint C but not all.

Lemma 1. *[121]. Let $G = \mathcal{P}(D, \pi)$ then*
(1) Every path in G is a sub-path of a path in D.
(2) If a path p in G does not lead to $-\infty$, then for all extensions $y \in \epsilon(p), G(y) = D(y)$.
(3) If a path p in G does lead to $-\infty$, then for all extensions $y \in \epsilon(p)$ either $\pi(y) = -\infty$ or $D(y) = -\infty$.

Part (1) gives a strong memory guarantee that G is no larger than D. Pruning accomplishes this by leaving some paths in G unchanged if only some (not all) of their extensions violate the constraint. pPDP uses the flat state $C = s$ as the constraint. Let $\mathcal{P}_s(D)$ denote $\mathcal{P}(D, s)$ for any ADD D and a flat state s.

$$Q = \mathcal{P}_s(R + \mathcal{P}_s(\sum_{X'_1} P_1 \times \ldots \mathcal{P}_s(\sum_{X'_l} P_l \times V'))) \tag{4.8}$$

As the expectation is computed over X'_i, state and action variables are introduced into the paths of V'. The paths that do not cover the current state are pruned and point to $-\infty$. Hence it is efficient to compute the ADD Q in memory.

Proposition 12. *(1) Q contains $O(2^m)$ paths over state and action variables that do not lead to $-\infty$.*
(2) Every path p that does not lead to $-\infty$ is a DP update for the Q-value of all states and actions in p.

Proof (Sketch) : (1) is due to using the state s as the constraint. Any path that has assignments to state variables differing from s is pruned and leads to $-\infty$. The paths

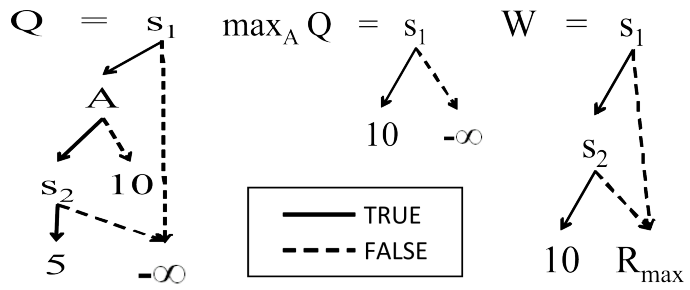

Figure 4.2: Example illustrating the mask M in pPDP. V is set to $R_{\max}$ initially. Q is computed using Equation 4.8 for the state $s_1 = 1, s_2 = 1$. The ADD $\max_{\mathbf{A}} Q$ gives an incorrect value for $s_1 = 1, s_2 = 0$, compared to W, the update using PDP-V (Equation 4.2).

that do not lead to $-\infty$ have different assignments to action variables for a maximum of $O(2^m)$ paths. (2) is due to part two of Lemma 1 because the pruning operator does not alter the paths that are consistent with state s. $\square$

The pruning operator removes portions of the diagram in subtle ways and therefore we have to be careful in choosing the mask M. Consider using the path $M = \Phi(\max_{\mathbf{A}} Q, s)$ which at first appears to be a natural choice. Unfortunately, this path does not give sound generalization because of the maximization.

To illustrate this, consider the hypothetical diagram Q shown on the left of Figure 4.2 where the state s assigns $s_1 = 1$ and $s_2 = 1$ and paths disagreeing with this assignment have been replaced with $-\infty$. The diagram $\max_{\mathbf{A}} Q$ is shown on the right and gives a value of 10 for state $s_1 = 1$ and $s_2 = 0$. This is incorrect since the true value of Q from the path $s_1 = 1$, $a = 1$, $s_2 = 0$ can be larger than 10. In $\max_{\mathbf{A}} Q$ the true value of the states on this path is ignored and assumed to be $-\infty$, and therefore the value calculated for the partial assignment $s_1 = 1$ is not correct for all extensions.

To guarantee correctness we require that all the values in the sub-diagram below the node to be different than $-\infty$. Therefore, states whose values are valid in $\max_{\mathbf{A}} Q$ are those where for all actions $\mathbf{A}$, $(Q \neq -\infty)$, denoted by the BDD $\forall_{\mathbf{A}}(Q \neq -\infty)$. In this example, $(Q \neq -\infty)$ when $\{s_1 = 1, A = 0\} \vee \{s_1 = 1, A = 1, s_2 = 1\}$, and quantification yields the mask $M = \{s_1 = 1, s_2 = 1\}$. Note that the BDDs $(Q \neq -\infty)$ and $\forall_{\mathbf{A}}(Q \neq -\infty)$ cannot be zero because all actions are updated in state s. Therefore, in the worst case, the mask M is equal to the state s and the step degenerates to a flat

RTDP update. Formally, the operator used in pPDP is

$$\hat{\mathcal{B}}(V, s) = V \oplus_M \mathcal{P}_s(\max_{\mathbf{A}} Q) \tag{4.9}$$

$$M = \Phi(\forall_{\mathbf{A}}(Q \neq -\infty), s) \tag{4.10}$$

Proposition 13. *Given an ADD V and state s, let $W = \hat{\mathcal{B}}(V, s)$ as in Equation 4.9, and let M be the path from 4.10. Then, $\forall q \in \epsilon(M)$, $W[q] = \mathcal{B}^*(V, q)[q]$.*

The proof follows from the soundness of pruning (Lemma 1) and the fact that all path extensions of M lead to a value not equal to $-\infty$ in Q. Therefore, by Proposition 10 pPDP converges as well. In cases where PDP exceeds memory limits pPDP can capture some of the sound generalizations, precisely those that can be captured without increasing the size of intermediate Q ADD. The only overhead in pPDP is the time required for the pruning operations which is negligible.

4.5 Experiments

We now evaluate the empirical impact of our proposed generalization operators within the family of RTDP-syle algorithms. To do this we compare our algorithms PDP-V, PDP-π, and pPDP to the following baselines: 1) **RTDP(Table)** is a simple table-based implementation of RTDP with state values initialized to $R_{\max}$. 2) **RTDP(ADD)** is like RTDP(Table) (i.e. no state generalization), except that each state backup is done symbolically using the FAR operator. This can be more efficient for factored actions compared to enumerating actions. 3) **sRTDP** [44], where our implementation uses FAR for updates in order to exploit factored actions. 4) **LR^{2}TDP** [86] is an extension of RTDP(Table) to solve finite horizon MDPs using iterative deepening and labeling, which was successful in recent planning competitions. 5) **FAR** [120] as described above. FAR is limited to 500 minutes of offline planning and then the resulting policy is executed online. This algorithm is only applicable to some of the small problem instances in our experiments and is included to give an optimal baseline value when possible.

All planners were implemented in a common framework, except for LR^{2}TDP, for which we used the publicly available code. For PDP-V and pPDP, we initialized each V^i with $R_{\max}$ (scaled by i). For PDP-π, we initialized π^i to the NoOp policy. Planning domains and problems are specified in the Relational Dynamic Influence Diagram

Language (RDDL) [128], which we convert to an ADD-based representation. The ADD variable ordering puts $parents(X_i')$ above X_i', where the X_i's are ordered (ascending) by the number of parents that are action variables. Note that the parents include current state variables and action variables so that this defines an ordering over all variables. In all experiments, our symbolic operators allow an approximation error of $\varepsilon = 0.1$ with the upper bound merging strategy [137].

Our experiments below are on 5 problem instances of varying sizes from three domains of the 2011 and 2014 International Probabilistic Planning Competitions (IPPCs). A memory limit of 4G is imposed to restrict the size of the ADDs, beyond which the planner can no longer proceed which we denote as "EML"(Exceeded Memory Limit). A planner is evaluated on a problem by running 30 trials of horizon 40 and averaging the total reward across the trials. We report the averages and 95% confidence intervals for each problem. Planners use a specified time limit per decision and we give results for different time limits. The value functions and policies are reinitialized after each decision.

Academic Advising Problem: The Academic Advising domain [56], from IPPC 2014, is a stochastic cost minimization problem that models the process of selecting the courses for a student in order to complete degree requirements, where the courses have complex prerequisite structure. The state space encodes which courses have been taken and whether they were passed or not. The actions at each step correspond to selecting one or more courses to take next. We consider two variants of the domain, a non-concurrent variant, which only allows a single course to be selected at each decision point, and a concurrent version, which allows multiple courses to be selected. The dynamics encode the probability that a course is passed if taken. Missing requirements and retaking of courses are penalized.

Figure 4.3[2] shows the performance vs. time for the different planners on IPPC 2014 problem instances for the non-concurrent and concurrent variants. All algorithms use a planning lookahead horizon of 16 steps. In the non-concurrent variant and shortest time limit (top left panel), we see that sRTDP fails to scale beyond the two smallest problems, and that FAR is able to solve these two problems as well. In larger instances both of these methods EML. In contrast, PDP-V, PDP-π and pPDP are able to give good performance across problem sizes. Moreover, for the smallest instances where FAR

[2]Charts best viewed in color.

is able to compute an optimal policy, these algorithms yield near optimal performance. This result demonstrates the importance of using update operators that attempt to trade off generalization and memory usage.

The flat search methods RTDP(Table), RTDP(ADD), and LR^{2}TDP do not perform well. For the largest three instances, these methods have a return no better than that of a NoOp policy. This shows the importance of generalization across states in order to achieve good performance in reasonable time.

Comparing performance across time limits (increasing time from left to right) we see the following. The flat search methods are not able to improve by much as the time limit is increased. PDP-V, PDP-π and pPDP also do not improve significantly with more time. Importantly they are able to avoid EML as more trajectories are sampled with larger time limits. PDP-V shows the most improvement on the largest instance as time increases. This shows that, in this domain, the use of generalization by our methods is the dominating factor in improving performance, and is even more effective than increasing the time limit.

Figure 4.3 (bottom) shows results for the same problem instances, but with concurrency (of 5 for the first instance and 2 for others). Here, both sRTDP and FAR (not shown) exceed memory limits even for the smallest instances. The flat search methods degrade quickly as the problem instances become larger. Our proposed methods (with one exception) outperform the competitors, especially for the larger instances. The exception is PDP-π on the largest instance, which results in EML after 18 seconds of planning due to the size of the intermediate ADDs in Equation 4.5. If we increase the time further (not shown here), PDP-V also does exceed memory limits. On the-other-hand pPDP does not result in EML due to its guarantees on bounding the diagram size (including intermediate diagrams), possibly at the expense of less aggressive generalization.

SysAdmin Problem : SysAdmin [57] models a computer network with n computers. Computers can fail with some probability, which requires a reboot action to correct. Neighbors of a failed computer have a higher probability of failing. The reward is based on the number of running computers with a cost associated with a reboot action. Unlike the academic advising domain, the number of reachable states in this domain is practically the entire state space. To allow sRTDP to produce non-EML results we consider networks of 10 computers connected in a star network. Following [120], we consider problems that vary the maximum number of computers that can be rebooted per decision

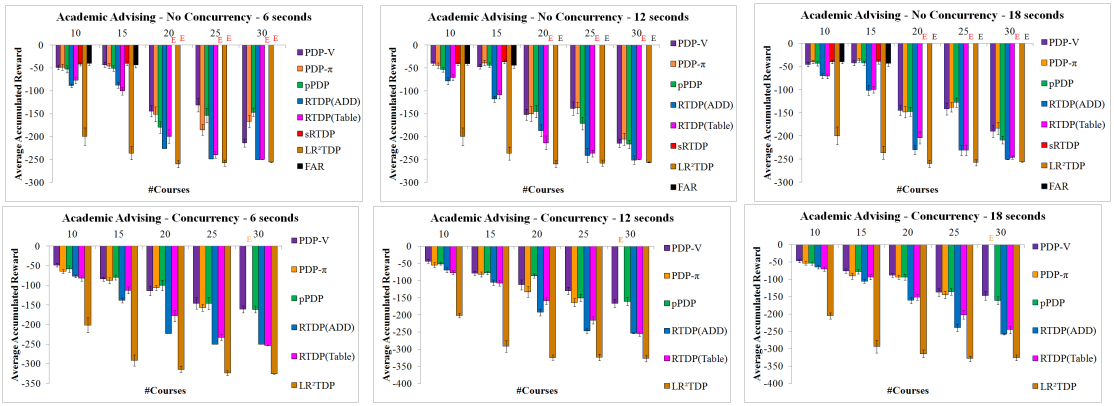

Figure 4.3: Academic Advising Problem : Performance vs. Time for time limits 6, 12 and 18 seconds per decision without (with) concurrency in the top (bottom) panels respectively. Error bars show 95% confidence intervals. The size of the state space is 2^{2x}, x is the number of courses shown on the x-axis. The algorithms that exceed memory limits (EML) are annotated as E.

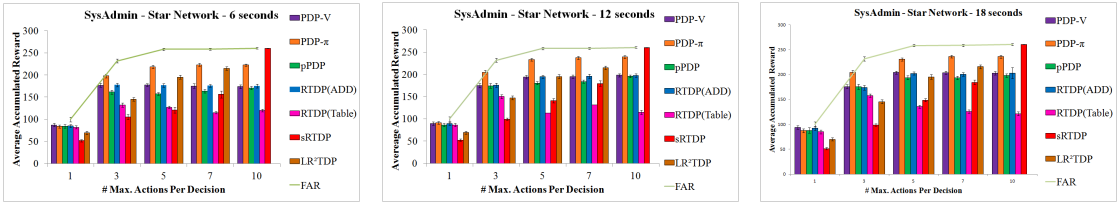

Figure 4.4: Performance vs. Time for 6, 12 and 18 seconds per decision in the SysAdmin Problem vs. concurrent actions : The state space is 2^{10} and action space is $O(2^x)$, x is the maximum number of parallel actions. Error bars show 95% confidence intervals.

$(1, 3, 5, 7,$ or $10)$, which gives a progressively growing factored action space.

Figure 4.4 gives results for three different time-limit settings. Due to the highly random nature of the problem, we used a short lookahead of four steps for all algorithms. The curve above the bar graphs shows the performance of the optimal policy found by FAR.

sRTDP exhibits interesting behavior in this domain. It performs worse than using no state generalization (i.e. RTDP(ADD)) in the first four instances and then optimally for the largest instance. The increased complexity of the sRTDP backup causes poor performance in the smaller action spaces—sRTDP samples fewer trajectories than our

algorithms. On the largest instance as the value ADD becomes more compact (more states have similar values), sRTDP is able to exploit generalization. This shows the difficulty of predicting apriori how much space and time the sRTDP generalization operator may require. In this domain, PDP-V and pPDP scale similarly to RTDP(ADD), because the reward function involves counting the number of computers, which makes the path formula $\Phi(R, s)$ the same as s. This means that these algorithms achieve very little state generalization in this domain. However, they do outperform RTDP(Table) due to the use of the factored FAR backup compared to a backup based on action enumeration.

In this domain, PDP-π clearly outperforms PDP-V and pPDP. In this case, policy irrelevance is able to capture abstract states more succinctly. For example, in the first instance, any state in which the computer at the center of the network is down has the same path formula : $\neg$`running`_$c1 \Rightarrow$ `reboot`_$c1$. Note that the values of these states are not equal and depend on the status of other computers. As parallelism increases, nodes near the center get added to these rules regardless of the status of nodes farther away. Clearly, in this domain, generalization based on policy irrelevance is more appropriate than value irrelevance. Finally we see that as the time limit increases there is a small improvement in performance for most algorithms. It is clear that the increase in performance due to larger time limits is not as significant as using the appropriate generalization mechanism, in this case policy irrelevance.

Crossing Traffic Problem: This IPPC 2014 domain models the arcade game Frogger, where the agent moves in a 2-D grid to cross a road to reach a goal location, while avoiding right-to-left moving cars that enter the road randomly from the rightmost column. The reward is -1 for each move and -40 for collision.

The boolean encoding of this domain has $|S| = O(2^{2(n^2)})$ for an $n \times n$ grid, with two bits per cell for the presence of the agent and car respectively. However, depending on the current location of the agent, many of these bits can be ignored for predicting the optimal value and action.

Figure 4.5 shows the results for different time limits and problem instances involving 3x3, 4x4, and 5x5 grids. There is larger variance in this domain, compared to the others, due to collisions. We see that sRTDP performs well in the first three instances and is able to improve with more time per decision. However, in instances 4 and 5 sRTDP exceeds memory limits when given more time. PDP-V and PDP-π scale to these instances and

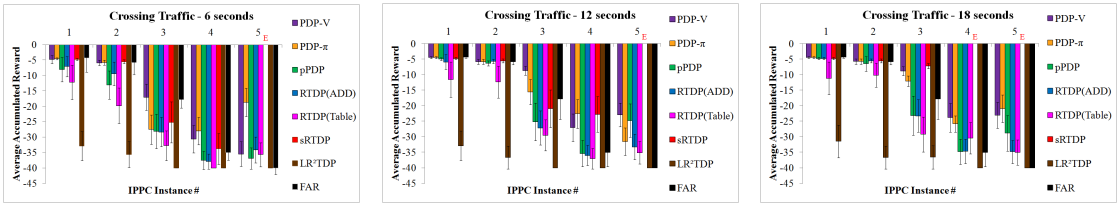

Figure 4.5: Performance vs. Time for 6, 12 and 18 seconds per decision in the Crossing Traffic Problem : The odd numbered instances have grids of size $3 \times 3, 4 \times 4$, and 5×5 and arrival probability of 30%. The even numbered instances have the same grid sizes but with arrival probability of 60%. Error bars show 95% confidence intervals.

times without EML. PDP-π performs better than PDP-V initially but PDP-V is able to improve more than PDP-π with more time per decision. We see that these algorithms outperform RTDP(ADD), showing that generalization is clearly useful in this domain. Again we see that generalizing appropriately is the dominating factor toward performance compared to increasing the time limit. pPDP never performs worse than the flat search methods RTDP(Table) and RTDP(ADD), and outperforms them in some cases. Its less aggressive generalization, however, leads to overall worse performance compared to our other algorithms.

Large instances : The preprocessing of translating RDDL to propositional logic does not scale for the large instances from the IPPC. For the purpose of showing the scaling of our algorithms, we refactored the RDDL domain - by decomposing the "robot-at(x,y)" propositions into two independent propositions "robot-at(x)" and "robot-at(y)" because the actions' effects are independent along x and y dimensions. Figure 4.6 shows the performance of PDP, PDP-π, pPDP and sRTDP for grids of sizes 6×6 and 7×7.

The algorithms are given much more time per decision to demonstrate the comparative scaling. The charts show the percentage out of 30 trials that the agent reached the goal. We see that in the 6x6 grid (left panel) PDP-V outperforms sRTDP by a large amount. sRTDP is able to scale with time without EML and slowly converges to optimal performance. By comparison, sRTDP does not perform well in the 7 by 7 problem (right panel) whereas PDP-V is able to make progress. PDP-π performs worse (better) than sRTDP in the former (latter) instance. The flat search methods are not able to make any progress in this problem due to the large stochastic branching factor. pPDP is able to improve on the flat search in the first instance but falls back to the flat method in the

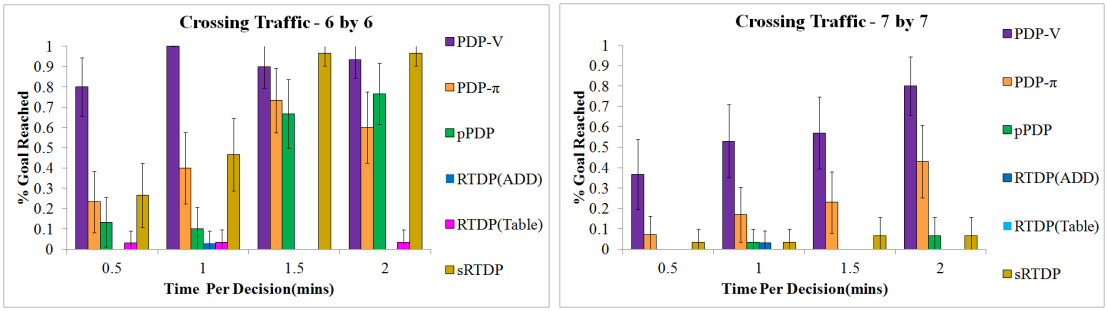

Figure 4.6: Crossing Traffic (large instances) : Performance vs. Time with 30, 60, 90 and 120 seconds per decision. Error bars show 95% confidence intervals.

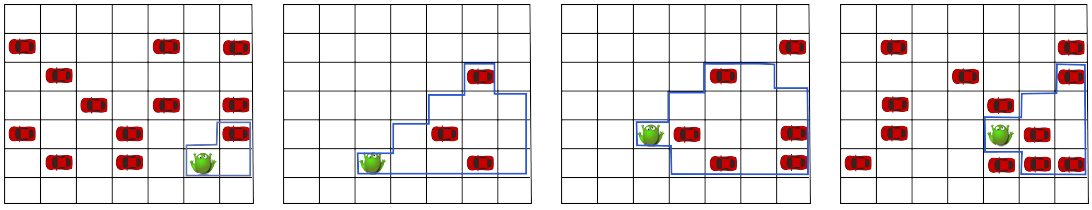

Figure 4.7: Some generalized states discovered by PDP-V in the Crossing Traffic problem. The grid denotes a flat state s_0 and the tiles in blue denote the generalized state $\Phi(V_0, s_0)$. The presence of cars outside of these tiles is irrelevant to value prediction.

larger problem.

Finally, we present a generalized state found by PDP-V to illustrate the effectiveness of generalization in these problems. Figure 4.7 shows an instance of the Crossing Traffic problem. All the cells in the grid, including the location of the agent and each car, constitute a flat state. The cells within blue denote the cells that appear in the path formula $\Phi(V_0, s_0)$ after running PDP-V from s_0. We see that PDP-V is able to ignore the assignments to many cells that are irrelevant for optimal online planning.

4.6 Discussion

We presented the first fully symbolic anytime planning algorithms for factored MDPs that generalize simulated experience soundly and efficiently. There were several observations from our experimental results. First, in all domains, we saw that using the appropriate form of generalization was the dominating factor towards good performance

compared to increasing the time-limit for algorithms without state generalization. Second, we saw that the most appropriate form of generalization can differ across domains and sometimes problem instances within a domain. This suggests that it is fruitful to investigate mechanisms for tuning or selecting among generalization methods. Third, pPDP never exceeded memory limits, while other generalization approaches did occasionally. Further, previous versions of sRTDP very frequently exceeded memory limits. This suggests that pPDP is perhaps the safest choice for generalization, especially for large problems and short time limits.

Our experiments on the academic advising problem are remarkable in that with the help of generalization, our approach is able to reach the goal state even in the largest problem instance which consists 2^{60} flat states. In the recent IPPC, every domain-independent planner was unable to reach the goal leading to concerns about the correctness of the domain description and problem specification. We showed that the bottlenecks of large flat state space and concurrent actions [56] are the bottlenecks that are successfully alleviated by our approach.

The most successful anytime planners in these competitions work by generating domain-independent heuristics using flat MDP relaxations of the factored MDP, e.g. using determinization [87], domain analysis [80] along with a gamut of heuristics for focussing the search near the root node under tight time constraints. Although these techniques improve the empirical anytime performance on the benchmarks, they are heuristic in the sense that they do not provide explicit generalization in a sound manner.

This work is orthogonal to research on improving the anytime performance of RTDP algorithms via labelling [15], smart sampling [104, 145] and heuristics [86, 81]. All of these can benefit from using our new operators as generalization is expected to benefit the rate of convergence. Finally, this work can benefit prior work on symbolic online planning for POMDPs using heuristic search [135].

This work is also related to the problem of state abstraction in MDPs that is the problem of finding equivalence relations between states. Indeed, it has been noted in [92] that the optimal SDP algorithms that use Algebraic Decision Diagrams covers the well-known bisimilarity relationship [36, 53] - an optimal notion of equivalence between MDP states. Our generalization operators in Section 4.3.2 implement a restricted version of value and policy irrelevance abstractions of [92] for the purpose of asychronous DP updates. A promising direction for future work is to incorporate bisimilarity metrics [47,

49, 48] into the online algorithm similar to the work of [127, 67] for incrementally refining the state abstraction using simulated trajectories. In contrast to [127, 67], symbolic methods are unique in that they allow abstract states to be merged and split efficiently in an online fashion.

In our experiments we used replanning at every state to output one action. Our approach paves the way for speedup learning for the purpose of generalizing the online search across different initial states i.e. intra-problem generalization. For the case of learning from policies, our approach outputs a set of BDDs that represent the (partial) policy at each time step. The crucial improvement over traditional learning approaches is that the examples, that are paths of the BDD, are generalized and succinct. One potential is merging the BDDs across different runs (different initial states). It is clear that an exact merging of these BDD policies (union over BDDs) would not be scalable, but some alternatives are available. In the work of [117], pruning functions are learned that are partial policies subscribing some set of actions including the optimal action. That is, it learns a policy that subsumes the actions selected by the anytime planner – a well-studied problem in BDDs and logical functions in general. For example, one might use the affine approximation of BDDs [65, 155], or use Horn approximations [133, 42] or any class of boolean functions with known closure operators for BDDs [131] from the knowledge compilation [31, 41, 43]. The DAGGER trick [126] could be employed into the learning algorithm.

Chapter 5: Online Action Selection using Hindsight Optimization

5.1 Introduction and Related Work

Many real-world decision-theoretic planning problems are naturally modeled using concurrent, hybrid (discrete and continuous) state and action (HSA) MDPs. Examples include reservoir control under rainfall uncertainty [124] and the unit commitment problem of power generation subject to demand uncertainty [114]. Existing approaches to solving expressive HSA-MDPs largely fall into two categories: dynamic programming for special restricted classes of HSA-MDPs, and approximate optimization of restricted value function or policy representations. Unfortunately, each category has critical limitations discussed next.

For the case of exact or bounded approximate solutions, numerous (symbolic) dynamic programming approaches have been proposed for restricted classes of HSA-MDPs ranging from the univariate continuous state setting for time-dependent MDPs [20] to piecewise value representations [45, 91, 130, 154] and phase-type transition approximations [97]. Later work introduced real-time dynamic programming extensions yielding more compact value functions [107, 143]. Sample average approximations (SAA) [83] of dynamic programming can be used for value approximation in an initial state [106]. Unfortunately, all these dynamic programming approaches for HSA-MDPs are either too restrictive or computationally intractable for the state-action dependent stochastic, 100-dimensional hybrid state, 50-dimensional hybrid action, and moderate horizon domains we experiment with in this chapter.

A different line of research directly optimizes a restricted class of value functions or policies. For value approximation, such methods are exemplified by Hybrid Approximate Linear Programming (HALP) [89], which sought to approximate expressive HSA-MDP value functions via a weighted basis function representation. On the policy approximation side, approaches like Pegasus [113] sought to optimize restricted parameterized policy classes subject to trajectories sampled in an SAA setting. All of these methods assume *a priori* knowledge of a good value or policy representation, which is typically

difficult to have in advance.

In this work, we explore a qualitatively different approach to solving a wide class of HSA-MDPs that does not require domain-specific assumptions on the value function or policy representation. Our approach is based on developing the framework of hindsight optimization (HOP) [25, 26] for HSA-MDPs. HOP provides an upper bound on the finite-horizon action values in the current state, which can be used for action selection. But the challenge is to compute this bound efficiently.

In particular our contributions are as follows: (i) We develop a generic linear space and time compilation of an expressive fragment of the RDDL [128] HSA-MDP representation to a mixed integer linear program (MILP). This compilation is augmented with action constraints to yield different algorithmic variations. (ii) Our contribution is the hindsight optimization variant. This generalizes previous work on HOP [72] to handle both continuous random variables and state-action dependent stochasticity. (iii) We develop a second variant based on straight line plans which is complementary in that it provides a lower bound on action values. (iv) Empirical results show that HSA-HOP scales to HSA-MDPs with moderate horizons and high-dimensional state and action spaces and generally outperforms baselines that are capable of scaling to such large hybrid MDPs. In a concluding case study in Chapter 6, we cast the real-time dispatch optimization problem faced by the Corvallis Fire Department as an HSA-MDP with factored actions. We show that our domain-independent planner significantly improves upon the responsiveness of the baseline that dispatches the nearest responders.

5.2 Hybrid State and Action MDPs

A **discrete-time** MDP is a tuple $(\mathbb{S}, \mathbb{A}, T, R)$ where $\mathbb{S}$ is a state set, $\mathbb{A}$ is an action set, $T : \mathbb{S} \times \mathbb{A} \times \mathbb{S} \to [0, 1]$ denotes the stochastic transition function for time $t + 1$ such that $T(s^t, a^t, s^{t+1}) = P(s^{t+1}|s^t, a^t)$, and $R : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$ denotes the state-action reward function. In this chapter we focus on finite horizon planning for a specified horizon h with the objective of maximizing the expected total reward accumulated over h steps.

Factored state and action MDPs [18, 120] extend the basic setting by specifying the state and action spaces as products of discrete variables. Hybrid State and Action MDP (HSA-MDP) [89, 130] keep the factored structure but provide a significant extension by allowing for both continuous and discrete state and action variables. The state space $\mathbb{S}$

and action space $\mathbb{A}$ are represented by finite sets of state variables $\mathbf{X} = \{X_1, \ldots, X_l\}$ and action variables $\mathbf{A} = \{A_1, \ldots, A_m\}$, where both $\mathbf{X}$ and $\mathbf{A}$ can contain both continuous and discrete random variables.

The transition function of the MDP is factored over the state variables, ie. $P(s'|s, a)$ is represented as a product of conditional probability distributions $P_i(q_i) = P(X'_i|q_i)$ for $i = 1, \ldots, l$. Each $P_i(q_i)$ is a function of a (typically small) subset $q_i \subseteq \{\mathbf{X}, \mathbf{A}, \mathbf{X}'\}$ where $\mathbf{X}'$ are the next state variables and the set of dependencies is acyclic. At any time t and state variable $X_i \in \mathbf{X}$, $X_i^{(t+1)} \sim P_i(q_i^{(t)})$.

5.2.1 RDDL Representation of Reward and Dynamics

Following recent work on HSA-MDPs [130, 154, 143], we use the description language RDDL [128] to specify HSA-MDPs. RDDL allows for relational templates to specify random variables which are instantiated via a set of objects. As a preprocessing step, we ground the templates with a concrete set of objects and expand relational quantifiers over their finite domains to obtain a propositional HSA-MDP as defined above. We note that the RDDL simply provides a convenient interface and our approach is not restricted to this language.

The RDDL source code specifies the factored transition and reward functions through a sequence of assignment and sampling statements that define intermediate variables and next state variables, using algebraic expressions in these definitions. The crucial point for a well-defined RDDL model is that the dependence among variables arising from the sequence of statements is acyclic.

HSA-MDPs with Piecewise Linear (PWL) dynamics have received significant attention due to their simplicity and expressivity [45, 91, 107, 154]. Whereas exact Dynamic Programming (DP) approaches for HSA-MDPs with PWL rewards and piecewise constant transition probabilities have been explored [45, 154], these cannot handle probability distributions whose parameters are continuous functions of state and action variables. In this case the optimal value function need not be PWL as required by previous work, making exact DP impossible. In contrast, our definition allows for a general form of stochasticity which we define next as the **stochasticPWL** class.

Definition 4. An expression for state variable X (or reward variable R) is a **stochasticPWL** expression if it is built recursively using the following 3 cases:

(a) A *deterministic* PWL expression containing (1) scalars, current state, current action, defined intermediate variables and next state variables, (2) boolean operations $\wedge, \vee, \neg, \geq$, (3) linear combination with constant coefficients, and (4) multiplication with a boolean predicate. The syntactic structure of each of these cases is shown in Table 5.1.

(b) If X a discrete random variable with support $\{1, \ldots, n\}$, its probability mass function is parametrized as $(E_1, \ldots, E_n)$ where each of $E_1, \ldots, E_n$ is a stochasticPWL expression.

(c) If X is a continuous random variable in the location-scale family of distributions [109], it is parameterized with a PWL transformation $X = E_1 + E_2 Z$, where E_1 (aka location) and E_2 (aka scale) are stochasticPWL expressions, and Z is a random variable for the standardized form of the distribution with known Cumulative Distribution Function (CDF).

The location-scale family includes many distributions of practical use such as the uniform, gaussian, exponential, logistic, beta, gamma distributions [109]. For example, the following RDDL expressions are valid according to Definition 4, where x^t is a state variable, a^t is an action variable, and z^t is an intermediate variable used in calculating r^t the reward.

$$x^{t+1} = \max(1, \min(0, 0.2x^t + 0.7a^t)) \tag{5.1}$$

$$z^{t+1} = \text{Normal}(x^t, a^t), a^t > 0 \tag{5.2}$$

$$r^{t+1} = \text{if } (x^t < z^{t+1}) \text{ then } x^{t+1} \text{ else } 1 - x^{t+1} \tag{5.3}$$

These illustrate the recursive structure of expressions, the limitation to PWL, and state-action dependent parameters of random variables. We emphasize that both state variables and action variables can be either discrete or continuous.

While PWL is a practical restriction, PWL functions [79] are an arbitrarily good approximation for higher order transition functions [39]. As will be clear below, the restriction to PWL is due to translating the RDDL code into mixed integer *linear programs*

Deterministic PWL	Condition	MILP Constraints
$E \to k$	k is a constant	$v_E = k$
$E_b \to$ true		$v_{E^b} = 1.0$
$E_b \to$ false		$v_{E^b} = 0.0$
$E \to p$	p is a state or action variable	$v_E = v_p$
$E \to \wedge_{i=1}^{n} E_b^i \equiv \forall_i E_b^i$	E_b^i is a boolean expression	$nv_E \leq \sum_{i=1}^{n} v_{E_b^i} \leq (n-1) + v_E$
$E \to \vee_{i=1}^{n} E_b^i \equiv \exists_i E_b^i$		$v_E \leq \sum_{i=1}^{n} v_{E_b^i} \leq nv_E$
$E \to \neg E_b$		$v_E = 1 - v_{E_b}$
		$v_E = 0$ or 1. Each $v_{E_b^i} = 0$ or 1
$E \to kE_1$	k is a constant	$v_E = kv_{E_1}$
$E \to E_1\ op\ E_2$	$op = +$ or $-$	$v_E = v_{E_1}$ op v_{E_2}
$E \to E_b E_1$	E_b is a boolean expression	Same as $E \to$ if E_b then E_1 else 0
$E_b \to E_1 \geq E_2$		$-M(1 - v_{E_b}) \leq v_{E_1} - v_{E_2} \leq Mv_{E_b}$
		$v_{E_b} = 0$ or 1, M is large e.g. $M = 10^6$
$E \to$ if E_b then E_1 else E_2	E_b is a boolean expression	$v_{E_1} - M(1 - v_{E_b}) \leq v_E \leq v_{E_1} + M(1 - v_{E_b})$
		$v_{E_2} - Mv_{E_b} \leq v_E \leq v_{E_2} + Mv_{E_b}$
		$v_{E_b} = 0$ or 1, M is large e.g. $M = 10^6$

Table 5.1: The syntax of Deterministic PWL statements and their MILP encoding. In each case E_1 and E_2 belong to the same language as E without cycles. The function represented by the expression E is equivalent to the free variable v_E in MILP.

(MILP) and using MILP solvers. By using more powerful solvers one can expand this approach.

5.3 Hindsight Optimization (HOP) for HSA-MDPs

HOP [25, 26] is a computationally attractive heuristic for online action selection. HOP approximates the optimal value function by optimally solving different realizations of the MDP called *futures* or *determinizations* and aggregating their values. While it is easy to construct domains where the HOP heuristic fails, previous work has shown that it performs well in many benchmark probabilistic planning problems [151]. Recent work has shown how to apply HOP in discrete factored MDPs through a translation to Integer Linear programs [72]. In this chapter, we show how these ideas can be extended in two respects in order to handle HSA-MDPs: First, we allow the more general state-action dependent stochasticity of Definition 4 (previous work restricted stochasticity to a small number of state-independent cases). Second, we allow for continuous and discrete variables.

The notion of a **random future** is central to the idea of HOP. Given a fixed policy, the MDP model induces a distribution over length-h trajectories. Viewing the choice of policy, and the random choices of the MDP as separate processes, we can make the random choices in advance (e.g., fixing the seed for the random number generator). This selection which, conditioned on any policy, produces a random trajectory for the policy is known as a random future.

HOP requires sampling random futures and once the random choices are fixed, a future represents a deterministic planning problem. The optimal value of any state in the MDP is $V_h^*(s^0) = \max_\pi E_f[\sum_{t=0}^h R(s_f^t, \pi_f^t)]$, which is the maximum expected value over random futures f of length h. In contrast, the hindsight value $V_h^{hop}(s^0) = E_f[\max_{a_f^0,\ldots,a_f^h} \sum_{t=0}^h R(s_f^t, a_f^t)]$ is the expected value of the optimal values of each future, where the inner maximization optimizes a *plan* (instead of a policy) for each future.

Observe that the maximizing values of actions a_f^t are future-dependent, i.e., "in hindsight" assuming a particular outcome of s_f^{t+1}. Due to swapping expectation and maximization and Jensen's inequality the function V^{hop} is an upper bound on V^* [105].

Action selection in HOP uses one-step lookahead using V^{hop} so that the outcome of the first action is not assumed. For each action a, the next states $s_1 \ldots s_F$ are sampled

and their V^{hop} value is used. The HOP algorithm picks $\arg\max_{\mathbf{a}} Q_h^{hop}(s_0, a)$ with

$$Q_h^{hop}(s_0, a) \approx R(s_0, a) + \frac{1}{F} \sum_{f=1}^{F} V_{h-1}^{hop}(s_f) \; . \tag{5.4}$$

5.3.1 Reduction of Deterministic HSA-MDPs to MILP

In this section we show how to translate a deterministic HSA-MDP, for example as generated by the determinization process of the next section. After determinization each P_i and R will be Deterministic PWL as given by Definition 1(a). The optimal plan is the solution to the following Mixed Integer Linear Program (MILP):

$$\max \sum_{t=0}^{h} R(\mathbf{X}^t, \mathbf{A}^t) \tag{5.5}$$

$$\text{s.t. } X_i^t = P_i(q_i^{t-1}), \qquad i = 1, \ldots, l; t = 1, \ldots, h; \tag{5.6}$$

where $\mathbf{X}^0 = s_0$ is a given initial state and $\mathbf{X}^t = (X_1^t, \ldots, X_l^t)$ and $\mathbf{A}^t = (A_1^t, \ldots, A_m^t)$ are the optimization variables for each $t = 0, \ldots, h-1$.

The translation of deterministic $P_i()$ to a set of MILP constraints is done recursively using the syntax in Definition 1(a). We formalize the syntax in the form of a recursive grammar in Table 5.1. The translation to MILP constraints is given in the third column and is standard for most cases. The encoding of if-then-else expressions requires the use of a large constant (i.e., "big-M trick") that can be chosen generically. Let the size of an expression be characterized by the size of the abstract syntax tree that produces the expression.

Proposition 14. *(Linear-time Reduction) Given a deterministic HSA-MDP specified in the language of Definition 1(a), the MILP in Equations 5.5-5.6 is such that (1) the compilation is produced in linear-time w.r.t. the number of constraints, the size of each PWL expression, the number of state variables and the planning horizon, and (2) an optimal solution of the MILP is an optimal plan for the deterministic HSA-MDP.*

5.3.2 Determinization of HSA-MDPs

Next we consider the determinization of HSA-MDPs by determinizing stochasticPWL expressions. The key question is how to sample a random future when the parameters of the random variables are not completely known at compile time, and are dependent on the states within a trajectory. We propose to encode random futures in a MILP using inverse transform sampling. Intuitively, we first sample a $u \sim \text{Uniform}(0, 1)$ and encode the quantile of order u as a set of MILP constraints. More precisely for any $u \in [0, 1]$,

1. If X is a discrete variable, its cumulative distribution function is $(E_1, E_1+E_2, \ldots, \sum_{i=1}^{n} E_i)$ and is PWL. The u-quantile is encoded in the PWL constraints: (1) $S_j = \sum_{i=1}^{j} E_j$ for $j = 1, \ldots, n$, (2) $x = \sum_{i=1}^{n} i \times [(S_i \geq u) \wedge (S_{i-1} < u)]$, and (3) $x \in \{1, 2, \ldots, n\}$.

2. If X is a continuous variable as in Definition 4, its quantile function is $F_X^{-1}(u; E_1, E_2) = E_1 + E_2 F_Z^{-1}(u)$ and is PWL. The u-quantile is encoded in the MILP constraint $x = E_1 + E_2 \times F_Z^{-1}(u)$, where $F_Z^{-1}(u)$ is a constant.

Proposition 15. *(Correctness for single variable sampling) Let X_i be a state variable whose transition function is given by a stochasticPWL expression as in Definition 4. Then, the MILP variable x produced by the above procedure encodes a random future for X_i given its parents q_i.*

Note that u and $F_Z^{-1}(u)$ are constants that are known at compile time. Yet the corresponding deterministic expressions for X yield a sample from the state dependent distribution which is not known at compile time. Consider for example the variables in Equation 5.2 and a pre-determined $u = 0.23$. The constraint $z^{t+1} = x^t + 0.23a^t$ produced by the above procedure encodes the 0.23-quantile outcome for every value of x^t and a^t.

The overall determinization algorithm applies the above procedure recursively on the syntactic structure of each X_i^{t+1}, using the corresponding deterministic or probabilistic MILP translation. By using Proposition 15 inductively over the acyclic structure of the dependencies in the RDDL code:

Proposition 16. *(Correctness of future generation) The MILP constructed using the overall determinization procedure is a random future from the distribution induced by the RDDL source code.*

5.3.3 HOP for HSA-MDPs

We next describe the overall MILP using multiple sampled determinizations. For a given HSA-MDP we produce copies of the state and action variables, annotated with superscripts f, t, where f is the future index, t the time step, and the optimization variables are $X_i^{f,t}$ and $A_j^{f,t}$. We generate F futures using the determinization procedure above. The objective function of the MILP is the h-step accumulated reward averaged over F futures. This objective and the first set of constraints is used by all of our algorithms:

$$\max \frac{1}{F} \sum_{f=1}^{F} \sum_{t=0}^{h} R(\mathbf{X}^{f,t}, \mathbf{A}^{f,t}) \tag{5.7}$$

$$\text{s.t. } X_i^{f,t} = \text{ Determinization of } P_i(q_i^{f,t-1}) \tag{5.8}$$

$$i = 1, \ldots, l; f = 1, \ldots, F; t = 1, \ldots, h \tag{5.9}$$

$$\mathbf{X}^{f,0} = s_0, f = 1, \ldots, F \tag{5.10}$$

The **HSA-HOP** algorithm uses one additional set of constraints restricting the action variables at time step $t = 0$ to be identical across futures:

$$A_j^{f,0} = A_j^{0,0}, j = 1, \ldots, m; f = 1, \ldots, F \tag{5.11}$$

Observe that the MILP in Equations 5.7-5.10 solves all futures independently. When the constraint in Equation 5.11 is added, the MILP implements a one-step lookahead where the solutions are coupled by requiring the first action to be the same.

Proposition 17. *(Equivalence to HOP)* **[72]** *The MILP in Equations 5.7-5.11 identifies the same solutions as the explicit HOP construction in Equation 5.4.*

Proof. The proof is due to the first term of the objective function being identical across futures for any fixed s_0 and any feasible solution to $A_j^{0,0}$, resulting in Equation 5.4. □

This construction identifies the HOP solutions in factored spaces without state and action enumeration and without the additional enumeration or continuous maximization implicit when using Equation 5.4 for action selection.

5.3.4 Algorithmic Baselines and Variations

As a baseline we use the simple idea that determinizes the problem using the most likely outcome determinization (for discrete variables) and the expected outcome determinization (for continuous variables). We use this baseline in our experiments denoting it as **Mean**. This idea is not new but the challenge is to encode it without enumeration of state-action dependent parameters of random variables. This can be done using Definition 4 with the MILP encoding: (1) If X is a discrete stochasticPWL variable, its most likely determinization $\max(E_1, \ldots, E_n)$ is PWL and equivalent to a MILP constraint (Table 5.1). (2) If X is a continuous state variable, its expected outcome determinization $E(X) = E_1 + E_2 E(Z)$ is PWL because $E(Z)$ is known.

We also consider two alternative formulations to HSA-HOP. The first is based on the idea of a **straight line plan** also known as an open loop policy or conformant plan. In this case we commit to a sequence of future actions regardless of the probabilistic outcomes of earlier actions. We can achieve this in the MILP formulation by replacing the constraint in Equation 5.11 with

$$A_j^{f,t}{=}A_j^{0,t}, j{=}1, \ldots, m; f{=}1, \ldots, F; t{=}0, \ldots, h-1 \tag{5.12}$$

The straight line value converges to the optimal value of the best open loop policy as the number of futures increases. Since in this case we are limiting the set of policies, the value of the optimal straight line plan is a lower bound on the optimal value of any state. The gap between the optimal HOP and straight line value can be used in various ways, e.g., to guide the generation of futures, to detect convergence, and to calculate approximation guarantees. Although this formulation commits to an entire plan, in our evaluation at each step we only use the first action from that plan, exactly like the other algorithms, and then replan for the next step.

The second variant is **Consensus**: determinizations are sampled exactly as in HSA-HOP, but solved independently of other determinizations. An action is selected by majority vote (ties broken randomly) among the $A_j^{0,f}$ across the futures. This trades off the monolithic MILP of HSA-HOP with several independent MILPs (one for each future) and aggregates their solutions heuristically.

5.4 Experimental Evaluation

As previously mentioned, we use the description language RDDL [128] to specify the domain dynamics. In all experiments we use the Gurobi optimizer [61] for optimizing the MILPs. The implementation used in our experiments is available as a domain independent tool for RDDL to MILP translation of HOP. We compare our algorithms **HOP** and **Straight Line** to the baselines of **Mean** and **Consensus**. When applicable we show the performance of **Noop**, **Random** and hand-coded policies.

The different algorithms are evaluated in an online replanning mode, i.e., planning is repeated at every world state and one action is output. The average accumulated reward over a horizon of 20 steps is measured (averaged over 30 trials) and a 95% confidence interval is shown. Each evaluation has three experimental parameters : (1) Time limit t per decision in minutes, (2) Lookahead L, the length of sampled futures and, (3) Number of sampled futures F per decision. We evaluated the algorithms by setting a reasonable value for t keeping in mind the runtime, then increasing L and F for best performance, until the MILP solver throws a memory error caused by an excessively large MILP. We use the best feasible solution found for any MILP after t minutes.

5.4.1 Domains

Power Generation [114, 2]: This domain concerns the unit commitment problem for a set of independent power plants represented as a Factored MDP. Each plant i has a current reserve of $\texttt{stock}_i$ ($\texttt{stock}_i \geq 0$) and observes fluctuations in $\texttt{temperature}_i$ with a uniform distribution as $\texttt{temperature}_i^{t+1} \sim \texttt{Uniform}(\mu_i - \delta_i, \mu_i + \delta_i)$. The fluctuations in temperature create a $\texttt{demand}_i$ for heating as well as cooling $\texttt{demand}_i^{t+1} = \mu_i|\texttt{temperature}_i^{t+1} - \mu_i|$. In this simple example, we assume μ_i and δ_i are constant, so this domain has state-independent continuous stochasticity. The demand is always positive with a V-shape centered at zero when $\text{temperature}_i = \mu_i$. The objective is to optimize $\texttt{order}_i$, the real-valued units of power to be generated at plant i within a prespecified budget $\sum_i \texttt{order}_i \leq B$ (B constant), produced at a cost of \$0.5 per unit and consumed at \$1 per unit $\texttt{reward}^{t+1} = \sum_i[\texttt{min}(\texttt{demand}_i^{t+1}, \texttt{stock}_i^t) - 0.5\texttt{order}_i^t]$. Unconsumed power is carried over as $\texttt{stock}_i^{t+1} = \texttt{if}\ \ (\texttt{stock}_i^t < \texttt{demand}_i^{t+1})\ \ \texttt{then}\ \ (\texttt{order}_i^t)\ \ \texttt{else}\ \ (\texttt{stock}_i^t - \texttt{demand}_i^{t+1} + \texttt{order}_i^t)$. This domain consists of a hybrid state space and

continuous action space.

The mean temperature is μ_i and the demand for the mean temperature is zero. Thus, the **Noop** policy is optimal with respect to the **Mean** determinization. The **maximum demand** is $\mu_i \delta_i$ per time step and **expected demand** is $\frac{\mu_i \delta_i}{2}$[1].

Reservoirs problem [124]: This problem consists of hybrid states with state-dependent noise in the transitions. The problem consists of a set of reservoirs connected by a set of 2-ended bidirectional pipes. Although any topology can be encoded, we demonstrate a linear topology of reservoirs and say that each reservoir i is downstream of reservoir $i-1$ and connected by pipe i. An MDP state is a list of positive current water levels $\texttt{rlevel}_i$ and $\texttt{rain}_i$ for each reservoir i. An MDP action is a list of real-valued flows $\texttt{flow}_i$, one for each pipe i connecting reservoirs $i-1$ and i. The sign of $\texttt{flow}_i$ determines direction of flow (positive for downstream). The rain level is stochastic as a zero truncated Gaussian distribution $\texttt{rain}_i^{t+1} = \texttt{max}[0, \texttt{Normal}(\texttt{rain}_i^t, \sigma)]$, and the flow is deterministic as $\texttt{rlevel}_i^{t+1} = \texttt{rlevel}_i^t + \texttt{rain}_i^t - \texttt{flow}_i^t + (\texttt{flow}_{i-1}^t > 0)\texttt{flow}_{i-1}^t + (\texttt{flow}_{i+1}^t < 0)\texttt{flow}_{i+1}^t$.

Each reservoir has a preset minimum α_i and maximum β_i water level. The objective is to minimize overflow and underflow of the reservoirs outside the prespecified limits, encoded by the reward function $\texttt{reward}^{t+1} = -5 \sum_i |\texttt{max}(0, \alpha_i - \texttt{rlevel}_i^t, \texttt{rlevel}_i^t - \beta_i)| - \sum_i |(\texttt{flow}_i^t < 0)\texttt{flow}_i^t|$.

The second term penalizes upstream flow (checked via the sign of $\texttt{flow}_i$) by the magnitude of the flow. PWL constraints are included in the RDDL to enforce (1) total outflow does not exceed current water level and (2) non-negativity of rain and water levels in each reservoir. In addition, each reservoir has a maximum capacity.

Icetrack : We illustrate a stochastic version of the classical Racetrack problem [8]. In contrast to the previous domains which have an optimization flavor, Icetrack is goal based and requires a large lookahead. Icetrack consists of real valued states of the form (x, y, v_x, v_y) where (x, y) is the real-valued position of a car on ice, and (v_x, v_y) are its velocities in the x and y directions respectively. The control inputs are the accelerations (a_x, a_y) in the two directions. The actions are susceptible to failure with a fixed probability θ with $v_x = \texttt{ if Bernoulli}(1-\theta) \texttt{ then } v_x + a_x \texttt{ else } v_x$. The transition for v_y is analogous. So this domain demonstrates state-independent discrete noise. We specify a goal state and measure the negative of the number of steps to reach

[1] $E[\text{demand}_i] = \frac{\mu_i}{2\delta_i}[\int_{x=\mu_i-\delta_i}^{\mu_i}(\mu_i - x_i)dx + \int_{x=\mu_i}^{\mu_i+\delta_i}(x_i - \mu_i)dx]$

it. Any collision makes the car unable to move for the rest of the episode. Icetrack shows two limitations of our PWL modeling restriction viz. we cannot use polar form (v, θ) as the state updates would require non-PWL functions, and in our discrete time model, collisions between (x^t, y^t) and (x^{t+1}, y^{t+1}) are given a bilinear function. So we restrict the track to be rectilinear (axis-parallel walls) to allow PWL collision detection (omitted due to space constraints).

5.4.2 Instances

In the Power Generation domain, we varied the number of plants between 10 and 50. All the plants start with 0 stock, the demand constants are set $\mu_i = 10$ and each δ_i is a fixed integer sampled uniformly between $\delta_i = 3$ and $\delta_i = 8$. In the Reservoirs problem, we varied the number of reservoirs between 10 and 50 but only show the three largest instances due to space constraints. We set $\alpha_i = 1000$, $\beta_i = 8000$ and capacity to 40000 for all reservoirs. All reservoirs are empty in the initial world state and $\sigma = 1024$. In the Icetrack problem, we tested with a 1 cell thick track along the edges of a $10X10$ grid, initial position set to $(5, 1)$ and goal $(5, 9)$. The acceleration is restricted $a_x, a_y \in [-4, 4]$, and the slippage θ is varied from 0% to 20%.

5.4.3 Results

The results are shown in Figure 5.1 for Power Generation, Table 5.2 for Reservoirs and Figure 5.2 for Icetrack. Overall, we see that HSA-HOP (denoted as HOP) performs the best across our three evaluation domains and Straight Line performs equally well in two of them.

Sampled vs. static determinization: In Power Generation, Mean is equivalent to Noop and achieves a total reward of zero, whereas HOP and Straight Line achieve much higher rewards. In the Reservoirs problem (Table 5.2), we see that the performance of Mean degrades as the number of reservoirs increases in comparison with HOP and Straight Line. Further, Mean has a higher variance and wider confidence interval across episodes whereas HOP appears more stable. Finally in Icetrack, determinization assumes that the actions always succeed. Even though the assumption is true in the first instance (Figure 5.2), we see that the performance of Mean is suboptimal, whereas using multiple

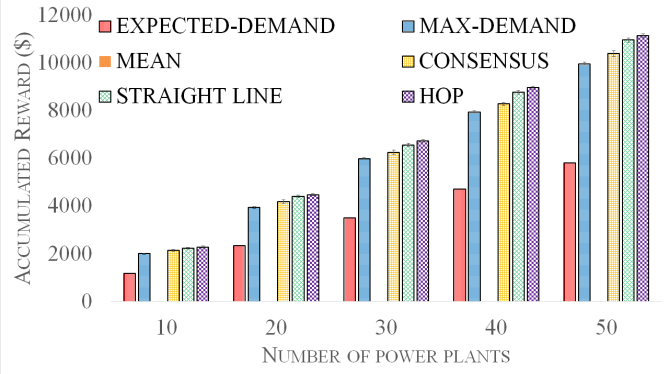

Figure 5.1: Results in the Power Generation problem with settings $L = 4, t = 0.5$ (mins), $F = 5$.

futures (as in HOP) leads to optimality. We found that this suboptimality is caused by numerical instabilities in the output of Gurobi across runs. As the noise increases in Figure 5.2, the performance of Mean is never better than HOP, because HOP accounts for the failure of actions.

Action Selection: The algorithms HOP, Straight Line and Consensus differ only in their action selection via the constraints they impose on action variables. Our HOP algorithm performs the best in this regard, and performance of Straight Line and Consensus vary. Our Straight Line algorithm performs surprisingly well in Power Generation and Reservoirs, and the small gap from the performance of HOP suggests the existence of high quality open-loop policies. In Icetrack, the strong action constraints of Straight Line, and the large lookahead of the domain, make the MILP significantly harder and leads to poor performance. The poor performance of Consensus is not surprising when continuous stochasticity is presented. We found that the average consensus among root actions went from as low as 20% in Power Generation, to 55% in Reservoirs, to 95% in the discrete noise case of Icetrack. Overall, HOP strikes a balance between optimality and hardness of the MILP, and gives the best performance over other action selection methods.

Reward ($\times 10^5$)	# Reservoirs		
Algorithms	30	40	50
HOP	**-2.4** (0.22)	**-1.51** (**0.02**)	**-2.27** (0.01)
Mean	−2.42(0.22)	−1.57(0.15)	−2.32(0.21)
Straight Line	−2.51(0.22)	−1.56(0.01)	−2.29(0.01)
Consensus	−2.78(0.21)	−1.78(0.05)	−2.61(0.04)
NoOp	−2.76(0.25)	−3.71(0.14)	−4.378(0.13)
Random	−2.76(0.25)	−6.41(0.17)	−7.94(0.19)

Table 5.2: Average Accumulated Reward ($\times 10^5$) and 95% Confidence Intervals with increasing reservoirs (columns), setting $L = 4$ lookahead steps, $t = 2$ (mins) and $F = 5$ futures per decision.

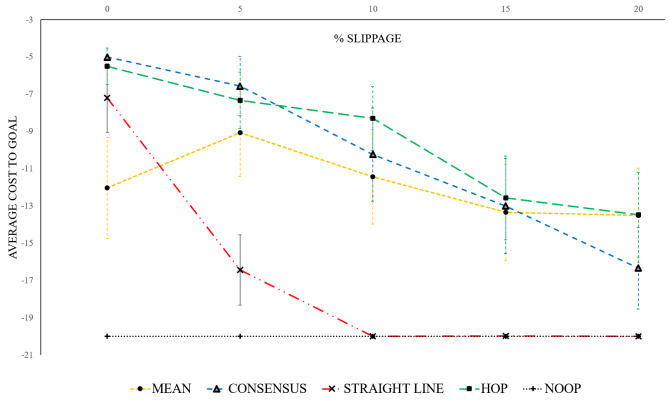

Figure 5.2: Results in Icetrack with settings $L = 20$, $t = 1$(mins) and $F = 5$.

5.5 Literature on HSA-MDPs

HSA-MDPs have received some attention in the literature (see Table 5.6). A naive approach is first discretizing the HSA-MDP [110] and using a discrete MDP solver. But this approach can lead to poor performance in practice [116] and is not considered here. Existing approaches to solving expressive HSA-MDPs largely fall into two categories: dynamic programming (DP) for special restricted classes of HSA-MDPs, and approximate optimization of restricted value or policy representations. Unfortunately, each category has critical limitations discussed next.

For the case of exact or bounded approximate solutions, numerous (symbolic) dynamic programming approaches have been proposed for restricted classes of HSA-MDPs ranging from seminal univariate continuous state settings for time-dependent MDPs [20]

to piecewise value representations [45, 91, 130, 154] and phase-type transition approximations [97] for multivariate state and action HSA-MDPs.

The obstacle for DP-based algorithms is the integration over continuous state variables, thus prior work has focussed on restrictions that lead to closed form solutions. When the number of outcomes of a state variable is finite, the integration is only a summation as in [107]. In addition, if the reward function is also PWL, then the optimal value function is also PWL [45]. Equivalently, when the transitions are deterministic expressions, the integration is a substitution operation, thus closed form analytical VI is possible in deterministic HSA-MDPs [130, 154, 143]. Despite the restricted scope of the MDPs, the number of pieces required to represent the optimal PWL value function grows rapidly with the planning horizon. The limitations of determinism and restricted scalability are overcome in this work.

Some prior work addresses the scalability issue via approximation, by interleaving steps of DP with value function approximation. The value function is approximated as a PWC function in [91], as a Piecewise Gamma (PW Gamma) in [97], and as a PWL function in [144]. An analytical DP update is possible in the PW Gamma case [97], whereas the complexity of the value function is the motivation in [91, 144]. Our approach is significantly different in terms of the quantity being approximated (Section 5.3) and allows strictly more general probability distributions.

Sample average approximations (SAA) [83] of dynamic programming can be used for value approximation in an initial state [106], but this is only tractable in restricted settings. Like our HOP approach, the SAA solution samples multiple deterministic futures, but, unlike HOP, SAA does not allow for independent solutions of those futures. As a result SAA is more demanding computationally, but converges to the optimal policy under some assumptions.

A different vein of research has chosen to directly optimize a restricted class of value functions or policies. On the value approximation side, such methods are exemplified by Hybrid Approximate Linear Programming (HALP) [89], which sought to approximate expressive HSA-MDP value functions via a weighted basis function representation. The key idea in HALP [89] is the use of conjugate families to approximate the transitions and value functions respectively, so that the expectation over continuous state variables has a closed form solution. The resulting LP has an infinite constraint space, so HALP uses discretization, an additional limitation for scalability and performance, in addition to the

drawback of designing basis functions to approximate optimal value functions. On the policy approximation side, approaches like Pegasus [113] sought to optimize restricted parameterized policy classes subject to trajectories sampled in an SAA setting. All of these methods assume *a priori* knowledge of a good value or policy representation, which is typically difficult to have in advance.

Finite horizon planning for MDPs is related to multi-stage stochastic programming (SP) [55]. SP solvers use algebraic modelling languages, such as SAMPL [139], the stochastic extension to the popular AMPL modelling language, thus similar to our representation of HSA-MDPs using RDDL expressions. SP solvers (e.g. [106]) typically assume that the number of possible state-action outcomes is finite [35]. Another typical assumption is that the stochasticity is independent of the actions (aka exogenous). We tackle both issues of infinite state-action outcomes and state-action dependent stochasticity.

5.6 Discussion

We explore a qualitatively different approach to solving a wide class of HSA-MDPs that does not require domain-specific assumptions on the value function or policy representation. Our online approach is based on developing the framework of HOP for HSA-MDPs that (i) introduces a linear space and time compilation of an expressive fragment of the RDDL [128] HSA-MDP representation to a mixed integer linear program (MILP), (ii) contributes a novel hindsight optimization (HOP) compilation generalizing previous work [72] to both continuous stochastic variables and state-action dependent stochasticity, and (iii) provides both upper and lower bound (stochastic) approximations of the optimal value differing on how the policy space is constrained in the MILP encoding.

The expressive class of HSA-MDPs consists of location-scale probability distributions for the transitions of each state variable. The importance of location-scale distributions is that they allow a compile-time encoding of the sampled next states at some time step in the future, even though the parameters of the corresponding density function involves state and action variables that are unknown at compile time. In our instantiation of this approach, the MILP solver then maximizes over the unknown policy over multiple sampled futures. This instantiation assumes that the parameters are piecewise linear functions of state and action variables, or location-scale distributions over them. To

the best of our knowledge, this is the first characterization of a subset of HSA-MDPs, and corresponding fragment of the RDDL planning domain description language, that have a linear time compilation to an MILP of linear size. The general approach is more broadly applicable by using a more complex solver that matches the function space of the parameters, e.g. using an MIQP solver when a quadratic function is the appropriate reward function.

Our reduction approach for optimizing futures is motivated by a history of good scalability, e.g. planning by reduction to satisfiability (SAT) [78], to maxSAT for cost optimal planning [125, 32], stochastic satisfiability for probabilistic planning [95], stochastic CSPs for factored POMDPs [69], and weighted model counting [38], and more recently, reduction to Integer Linear Programs (ILP) for discrete factored action MDPs [72]. HSA-HOP contributes a novel and compact MILP encoding for HSA-MDPs to this line of reasearch that handles the additional concerns mentioned above. of factored actions.

The key idea in [72] is reduction to Integer Linear Programming (ILP) for factored MDPs with boolean state and action variables. It is clear that hybrid state-action variables can be handled with a simple extension to Mixed ILP (MILP). The key question that we answer is the encoding of state-action dependent stochasticity for continuous variables, whereas [72] assumes discrete state variables with state independent noise ie. PWC functions. In the tradition of reduction based approaches, we show that our MILP encoding is compact.

Our main algorithm HSA-HOP significantly improves on the scalability of previous domain-independent approaches by leveraging state-of-the-art MILP solvers, scaling to MDPs with 100 continuous state dimensions and 50 continuous action dimensions. The alternative algorithm using straight line plans provides a complementary heuristic based on a lower bound approximation. This is found to be competitive in some cases but overall less robust (cf. Icetrack) than our novel contribution of HSA-HOP, which shows uniformly strong performance over diverse domains.

Future work is needed in the algorithm space because HSA-HOP is an unsound and heuristic algorithm by nature of the HOP approximation because it allows identical states to have different future-dependent optimal actions. Future work must alleviate this and formulate sound (e.g. RTDP) and sample-efficient algorithms (e.g. Sparse Sampling) for HSA-MDPs. The challenge is to retain the scalability of the HOP approach that is able to scale to truly massive HSA-MDPs. However, for the time being HSA-HOP serves as

	Transitions	Reward	Factored States	Actions
Dynamic Programming (DP)				
[20]	Discrete	PWL	Hybrid	Discrete
[107]	Discrete	-	Hybrid	Discrete
Symbolic DP (SDP)				
[45]	Rectilinear PWC	Rectilinear PWL	Continuous	Discrete
[130]	Deterministic	-	Hybrid	Discrete
[154]	Determinstic PWL	PWL	Hybrid	Continuous
[143]	Deterministic PWP	PWP	Hybrid	Hybrid
Approximate DP				
[91]	PWL	PWC	Hybrid	Discrete
[97]	Exponential	PW Gamma	Discrete	Discrete
[144]	Deterministic PWL	PWL	Hybrid	Continuous
Approximate Linear Programming (ALP)				
[64]	Beta distributions	Polynomial Bases	Continuous	Discrete
[58]	Beta distribution	Polynomial Bases	Hybrid	Hybrid
[88]	Exponential family	Exponential family	Hybrid	Hybrid
[89]	Beta distribution	PWL	Hybrid	Hybrid
Hindsight Optimization (HOP)				
[72]	Rectilinear PWC	Rectilinear PWC	Discrete	Discrete
This work	Stochastic PWL (Definition 4)	PWL	Hybrid	Hybrid

Table 5.3: Summary of related work to hybrid state-action MDPs (PWC=Piecewise constant, PWP=Polynomial, PWL=Linear). Our domain-independent work generalizes to the fully hybrid setting with a more general model of state-action dependent stochasticity (Definition 4).

a baseline for these MDPs.

Another potential direction for future work is the learning of models within this general subset of factored state and action MDPs. Recent advances in learning bayesian networks and learning of multi-layered neural networks can be tapped. In fact, the commonly used convolutional neural network with rectified linear activations, if used to model the one step dynamics, is exactly equivalent to the deterministic PWL class. While HSA-HOP can take these expressive models as input in principle to derive model predictive controllers, the neural models are significantly larger in terms of the number of neurons compared vs the number of state and action variables.

Chapter 6: Application to Response Optimization of Fire and Emergency Services in Corvallis

6.1 Introduction and Literature Review

The emergency response problem has been well studied over the past five decades in the planning and operations research communites, for example see the comprehensive surveys of [54, 136, 1, 11, 3, 27]. The critical goal is to improve the performance of existing emergency response services, by providing decision support tools that are based on automated planning and decision making. The performance is typically measured by the precious minutes elapsed between the reported time of the emergency and the time of the arrival of responders. This domain has attracted the attention of many cities in order to counteract the effects of urban sprawl (leading to larger service regions) alongside limited budgets (leading to few available responders). Some examples include Melbourne [100], London [102], Utrecht [75], Vienna [132], and in this work the city of Corvallis following the earlier work of [14]. Our domain-independent online planning approach coupled with a parametrized (relational) domain description provides a powerful tool that can easily be modified to fit the needs of some different service region.

Broadly, prior research in this domain is traditionally categorized into *static* and *dynamic* approaches. Static approaches concern one-shot (time-invariant) decisions, exemplified by ambulance location and coverage problems studied at length in operations research [22, 70, 153, 52]. These concern the optimal location of responders under some fixed dispatch policy that assigns responders to any given emergency.

The most commonly used dispatch policy is to dispatch the nearest responders in terms of travel-time [73, see Table 2]. Many static approaches as well as simulation models developed for this domain [134] heuristically use this policy as the basis of their study. It is well-known that the dispatch policy has significant impact on the response time, and in turn on the patient survivability in the EMS setting [149] because of the highly random and time-dependent nature of the demand. For example, [63, 33, 93] show that none of the commonly used fixed dispatch policies dominate the performance over

heterogenous types of emergencies. Consequently, we also find that the performance of HSA-HOP dominates the aforementioned baseline on real-world emergency sequences.

Dynamic approaches can potentially account for real-time concerns such as the road network, weather conditions and real-time traffic information [4]. Although dynamic approaches typically optimize the dispatching policy while assuming a fixed location for responders' stations, they can potentially include the relocation decisions considered in static approaches as well. The modification is similar to [153], where the additional MDP actions of relocation are available at a coarser time scale than the dispatching actions.

In this context, two-stage algorithms have been used to jointly address the location and dispatching problems as well. The work of [112] use a two-stage stochastic program to minimize the number of relocations, whereas [14] use a two-phase hill-climbing search that alternates regionalized policy optimization with relocation of reponders under a fixed dispatch policy. We follow [14] in many aspects but restrict to the latter part of dispatch optimization with fixed base stations for responders. However, in contrast to both [112] and [14], we allow a novel form of re-deployement at every time step that allows responders to be redirected from the location of a prior emergency to a new one.

We show that the emergency domain is naturally formulated as a Hybrid State and Action Markov Decision Process (HSA-MDP, [89]). The state space consists of both continuous and discrete random variables that encode the status of all responders and details of the current emergency. Our dynamic approach is online replanining using HSA-HOP that uses the HSA-MDP model to output the action for the current world state. Online action selection offers a scalable alternative to global dispatch policy optimization and removes the need for human effort in engineeringn a policy space. It allows quick and near-optimal responses for any given emergency taking into account the subsequent emergencies that may arise in the near future.

The first challenge in the straightforward application of MDP solvers is the hybrid state space, causing algorithms based on Dynamic Programming to be intractable in general. The second challenge of the online formulation is the factored action space that is the cross product of assignments to individual responders. As shown in this thesis, existing MDP solvers scale poorly with the number of actions in general, that is exponential in the number of responders in this domain. The domain-independent HSA-HOP algorithm [123] is the current state-of-the-art for hybrid and factored state and action MDPs with state-action dependent stochastic transitions and rewards. We show

that HSA-HOP is able to handle the multidimensional hybrid state and action space of this domain, due to its compact and factored MILP encoding of multiple futures spanning multiple time steps.

We leverage an expressive continuous space and time model, whereas prior MDP based approaches to this domain must resort to discretization of the continuous locations (e.g. [62, 73, 27, 74]) and time [74, 112]. Due to the computational obstacle of the continuous state space formulation, the basic approach is to approximate the service region using a fine-grained graph [74, 27, 93]) or as a grid (as in [132, 112]), and use existing solvers based on exact (e.g. Value Iteration in [74]) or approximate dynamic programming (ADP) (e.g. [101, 132, 14]) respectively.

The ADP approach optimizes the dispatching policy within some restricted space of policies (or value functions) spanned by prespecified basis functions. For example, the works of [14, 132] consider "regionalized" policies based on some discretization of the continuous grid and the use an engineered approximation space. The work of [101] learn a type of re-deployement policy, that maps a free ambulance (that has completed its assignment) to its next standby (or base) location.

These algorithms are based on approximating the value function in terms of prespecified basis functions. In follow up work, [99] showed a potential pitfall of the approach in simple example MDPs where ADP fails to find the optimal dispatch policy despite using sufficiently expressive basis functions. The authors propose a direct policy search method in order to circumvent this issue, but the depedence on hand-coded basis functions still exists.

The sequence of papers [140, 142, 141] refer to these policies as *compliance tables* and give several heuristic algorithms that use a finite graph to approximate the region and time. In contrast to these, we note that our action space allows re-deployement but does not impose that the nearest responder must respond. Secondly, their focus is on minimizing the number of re-deployements and not the raw response time. The table is computed using only one-step of lookahead over all possible next-states in [9, 140, 10].

Following recent work on HSA-MDPs [130, 154, 143], HSA-HOP takes as input the reward and dynamics of the MDP specified in the RDDL domain description language [128]. We argue that a high-level domain description language such as RDDL is suitable for developing decision support tools for the emergency domain for the following reasons. (1) RDDL allows for relational templates to specify random variables which are instan-

tiated via a set of objects. This parametrized encoding provides a unified framework for much of the prior research in this domain. For example, we test the performance with varying number of responders under varying objectives by viewing them as different instantiations of the relational RDDL domain description. (2) RDDL provides a concise and interpretable encoding of the probability distributions of discrete and continuous random variables. We demonstrate some basic machine learning models learned in the form of RDDL expressions from real-world emergency data. (3) RDDL allows general state-action constraints to be specified that are considered necessary by prior research from the standpoint of overall system architecture, e.g. enforcing load balancing over responders, upper limit on the response time, tiered priority depending on the nature of the emergency etc.

In the context of the vast literature in this domain, our first contribution is an RDDL encoding of the emergency domain (annotated in `typefont`) that can be handled by HSA-HOP. We first describe the hybrid state space that captures the effects of dispatching actions on the future availability of a responder. In order for HSA-HOP to be applicable, the RDDL encoding must satisfy the requirements of Definition 4, in order for which we must make piecewise linearity assumptions on the travel time and service time of a responder.

In order to complete our domain encoding, we need to specify an RDDL model for the arrival of emergencies. In our second contribution, we show that HSA-HOP can overcome this with a minor modification, that allows randomly sampling futures (sequence of 9-1-1 calls) directly from a corpus of real world emergencies. Historical data was used in the prior work of [112], but their underlying algorithm optimizes the assignments in multiple futures independently of one another. HSA-HOP incorporates a one-step lookahead by coupling the futures via the first state-action pair, and thus algorithmically differs from their approach.

As our third contribution, we illustrate the supervised learning of a hybrid probabilistic model. Our modelling assumptions are common to many prior works on the emergency domain from the operations research community. Rather, our illustration tests two potential limitations of HSA-HOP namely (1) the expressivity of Definition 4 with respect to basic probabilistic models, and (2) the effect of overfitting during model learning on the computational complexity of the planning algorithm. Section 6.3 contains experiments that illustrate the performance of our approach in comparison to the

dispatch-closest policy.

6.2 HSA-MDP Formulation

The state space of our formulation is based on keeping track of the current time. In our formulation, each state presents a new emergency and the action consists of the set of responders to dispatch. We do not allow deferring the response to the future, therefore we do not maintain a history of previous emergencies. The forward flow of time is determined by the arrival of emergencies. The state space consists of :

- Emergency state : $(x, y, t, \texttt{code})$: denoting the x, y coordinates in miles, the current time and the nature of the emergency. The continuous variable $t > 0$ is a floating-point representation of time. The nature of the emergency, denoted by `code`, is a discrete random variable that can take one of 15 distinct values. Different codes represent the severity of the emergency and the number of responders required to fully service the call.

- Responder state : $(x(r), y(r), t_{\text{in}}(r), t_{\text{out}}(r), t_{\text{home}}(r))$ for each responder r, where (x, y) are the coordinate of the responder's last known location and t_{in} is the absolute time of arrival at the scene of the emergency last serviced by responder r. Similarly, $t_{\texttt{out}}$ and $t_{\texttt{home}}$ are the expected time of completion of its previous assignment and time of return to its home base respectively.

Action Space : In our simplified presentation, each responder can fill exactly one of four *roles* namely Engine, Ambulance, Ladder and Command. Each MDP action is an assignment to boolean action variables of the form `dispatch(responder,role)`. The size of the action space is $O(2^n)$ for n responders, the exponential size of the action space is caused by the factored action space. In practice, the action space is even larger because responding units are formed on the fly using available personnel and ground vehicles.

When a unit is dispatched, the responder travels from $(x(r), y(r))$ to (x, y) at a constant speed. Upon arrival, the responder spends a constant amount of time at the scene that is determined by `code` (denoted by `stay(code)`). Note that the model for the

service time can be easily extended (cf. scale family) to the more commonly used exponentially distributed service times. Depending on `code`, the responder must additionally proceed to one of two special locations (hospital or transfer) before the assignment is considered complete (denoted by $\texttt{dest}_x$ and $\texttt{dest}_y$).

After a mandatory waiting period (denoted by `wait(code)`) at the end point of the assignment, it returns to its appointed (fixed) home base (denoted by $\texttt{home}_x$ and $\texttt{home}_y$). We allow a novel type of redeployement than prior work, specifically we allow a responder to be directly dispatched to a new emergency during this waiting period.

We follow the Code3Sim simulator [90] in assuming that `wait(code)` and `stay(code)` are constants for any given `code`. As mentioned above, these timings are typically modelled using exponential distributions that are well within the scope of our planning algorithm. The update equations for the responders' state variables are specified as a PWL function of the travel time.

Travel Time : Following [73], we assume that the travel time of a responder is a deterministic function of the distance between any two points. Following [90], we use the manhattan distance ($d = |x_1 - x_2| + |y_1 - y_2|$ between any two points $(x_1, y_1), (x_2, y_2)$) because it naturally fits the PWL requirement of Definition 4. This assumption may be valid in many urban regions. Similar to the service times, our assumption that the travel time is deterministic can be easily extended via Definition 4. For example, the travel time based on the manhattan distance may represent the mean of a normal distribution whose variance may reflect the effect of the time of the day.

In prior work, [90] also approximated the speed of responders based on the road network and the availability of highways in Corvallis. The mapping is a piecewise constant function with 10 pieces, e.g. for distances under 5 miles, we use a constant travel speed of 38 mph where for distances between 5 and 7 miles we use a speed of 42 mph [1]. Combining these leads to a PWL model of the travel time as shown in the following equations.

[1]The mapping is (5,7,12,15,17.5,20,25,28.5,41,70,500 miles) to (38,42,40,48,56,52,50,54,49,70,70 mph, respectively).

1.

$$
\begin{aligned}
d(r) &= |x(r) - x| + |y(r) - y| \\
\text{Travel Time : } t(r) &= \frac{d(r)}{38}(d(r) \leq 5) + \frac{d(r)}{42}(d(r) > 5 \wedge d(r) \leq 7) + \ldots \\
t'_{in}(r) &= \texttt{if } (\exists_{s:\texttt{role}}\texttt{dispatch}(r, s)) \\
&\qquad \texttt{then } t + t(r) \texttt{ else } t_{in}(r)
\end{aligned}
$$

2.

$$
\begin{aligned}
d_2(r) &= |\texttt{dest}_x\texttt{(x,code)} - x| + |\texttt{dest}_y\texttt{(y,code)} - y| \\
t_2(r) &= \text{//Similar to } t(r) \text{ but using } d_2(r) \\
t'_{out}(r) &= \texttt{if } (\exists_{s:\texttt{role}}\texttt{dispatch}(r, s)) \\
&\qquad \texttt{then } t'_{in}(r) + \texttt{stay(code)} + t_2(r) \texttt{ else } t_{out}(r)
\end{aligned}
$$

3.

$$
\begin{aligned}
d_3(r) &= |\texttt{home}_x\texttt{(r)-dest}_x\texttt{(x,code)}| + |\texttt{home}_y\texttt{(r)-dest}_y\texttt{(y,code)}| \\
t_3(r) &= \text{//Similar to } t(r) \text{ but using } d_3(r) \\
t'_{home}(r) &= \texttt{if } (\exists_{s:\texttt{role}}\texttt{dispatch}(r, s)) \\
&\qquad \texttt{then } t'_{out}(r) + \texttt{wait(code)} + t_3(r) \texttt{ else } t_{home}(r)
\end{aligned}
$$

Depending on the nature of the emergency, the function $\texttt{dest}_x\texttt{(x,code)}$ ($\texttt{dest}_y\texttt{(y,code)}$, respectively) returns the x-coordinate (y-coordinate, respectively) of the hospital, transfer location or simply returns x (y, respectively). The transition for $x(r)$ and $y(r)$ are

analogous.

$$x'(r) = \texttt{if } (\exists_{s:\texttt{role}} \texttt{dispatch}(r, s))$$
$$\texttt{then dest}_x\texttt{(x,code)}$$
$$\texttt{else if } (t > t_{home}(r))$$
$$\texttt{then home}_x(r)$$
$$\texttt{else } x(r)$$

Response Time : The performance of responders is evaluated in terms of *first response* and *full response*. The first response δ_1 is the time elapsed between the reporting of the emergency and the arrival of the first responder.

The full response is measured with respect to predefined requirements, specified as a mapping from `code` to the number of required responders of each role and denoted as `required`. For example, a Structure Fire requires a total of four responders viz. two engines and one ladder and command each, whereas a Code-1 Medical emergency requires only one ambulance. The specific constants we use are based on the Code3Sim simulator [90].

We define the intermediate boolean variable `overwhelm` to check if the requirement of full response is not met. In the event of an overwhelm, the full response time is set to the maximum possible response time Δ. The full response δ_2 is then defined as the supremum of the time of arrival of all responders. Since the travel times of individual responders satisfies Definition 4, so does the minimum and maximum over them.

1.

$$\delta_1 = \min_r[\texttt{if } (\exists_{s:\texttt{role}} \texttt{dispatch}(r, s))$$
$$\texttt{then } t(r) \texttt{ else } \Delta]$$

2.

$$\texttt{overwhelm} = \exists_{s:\texttt{role}}(\texttt{required}(\texttt{code}, s) > \sum_{r} \texttt{dispatch}(r, s))$$

$$\delta_2 = \texttt{if (overwhelm) then } \Delta$$
$$\texttt{else } \max_{r}[(\exists_{s:role}\texttt{dispatch}(r, s)) \times t(r)]$$

Emergency state : It is far more challenging to specify the dynamics for the exogenous part of the state space. Recall that the HSA-HOP algorithm works by sampling random futures i.e. a relaxation of the MDP such that the sequence of states with fixed initial state is a function of the policy alone. As observed by prior works, note the property that the factored transition of the emergency state is conditionally independent of the transition of the responder's state.

We sample futures from historical data that consists of the logs of emergencies across days. Given an emergency state with time t, we identify the closest emergency for each day that is the first emergency that occurs after time ($t\%24$) on that day. This results in a number of candidates equal to the number of days in the dataset. We select one of these at random and return $(x', y', \texttt{code}', t')$ such that t' occurs after the original time t. In Section 6.3, we sampled futures from a dataset of 30 days collected in the month of January 2011, that corresponds to a sample space of $O(30^h)$ outcomes for length h futures. This approach allows incorporating real world data without the need for modelling effort expended by the human. On the other hand, the sampled futures do not generalize or present novel events beyond those in the dataset (Section 6.4).

This corresponds to a point estimate of the interarrival gap $\delta(t)$ and we set $t' = t + \delta(t)$, assuming that δ is IID for different days. This discrete-time MDP formulation simulates only the arrival of emergencies. A slightly more complex model defined as $t' = t + \min(\delta(t), \min_r t_{out}(r), \min_r t_{home}(r))$ simulates the change in status of responders. This more complex model can allow responses to be delayed but it is still within the scope of deterministic PWL (satisfies Definition 4).

Constraints : The following RDDL constraints enfores the availability of resources and forward flow of time. These deterministic expressions satisfy Definition 4 and are

translated to MILP constraints for each time step as well as each future.

$$\forall_{r:\texttt{responder}} \forall_{s:\texttt{role}} [\texttt{dispatch}(r,s) \Rightarrow$$
$$[(t \geq t_{out}(r)) \wedge (t \leq t_{out}(r) + \texttt{wait(code)})] \vee (t \geq t_{home}(r))]$$
$$\forall_{r:\texttt{responder}} \sum_{s:\texttt{role}} \texttt{dispatch}(r,s) \leq 1$$
$$\forall_{r:\texttt{responder}} 0 \geq t_{in}(r) \geq t_{out}(r) \geq t_{free}(r)$$

The first constraint is an action precondition to ensure that the responder is either at the scene after completion of the previous assignment (tarry) or idling at the base station. The second constraint ensures that each responder is assigned at most one role and is also an action precondition. The third one is an integrity constraint that ensures the validity of the timers for each responder.

6.3 Experiments

Baseline : As mentioned earlier, there are no domain-independent MDP algorithms that scale to hybrid state and action spaces of high dimensions. Our reference baseline will be the dispatch-closest policy (denoted closest for short) that is used extensively in the literature. In Corvallis, the current operations and the simulator software used to study the performance is based on the dispatch-closest policy.

The policy works as follows : it iterates through the roles of Engine, Ambulance, Ladder, Command, and for each role dispatches the k-closest available responders, where k is the minimum of the number of available responders and the required number (specified by $\texttt{required}(\texttt{code}, \texttt{role})$) of responders. The closest responders are identified using the travel time. Intuitively, it is easy to see that the dispatch-closest policy is not optimal when the density of emergencies is non-uniform (in space) and the locations of responders also changes over time. In addition, this policy does not optimize the first response, which often requires a responder to be dispatched immediately even if their type is non-essential for the nature of the current emergency.

Setup : Our experimental setup is similar to Section 5.4. The evaluation is in online replanning mode, i.e., planning is repeated at every world state and one action is output. Each evaluation has three experimental parameters : (1) Time limit t per decision in

minutes, (2) Lookahead h, the length of sampled futures and, (3) Number of sampled futures F per decision.

In practice, the time limit must be small. In Corvallis, the dispatch must be within two minutes of the received call. We note that our implementation of the MILP translation is not optimized for speed. We found that for small values of h and F, Gurobi solves the resulting MILPs optimally for many states in under two minutes. HSA-HOP is not able handle large values of h, and to a lesser extent large values of F. For a fixed value of t, this is seen via the degradation of the accumulated reward with increasing h or F. We attribute this issue to the monolithic MILP solved by HSA-HOP that simultaneously solves for all futures, whereas their solutions are only loosely coupled via the action at the root state. While Gurobi and many modern MILP solvers may automatically identify and decompose the MILPs, we think that a significantly more scalable implementation is possible via the dual decomposition method of [150].

6.3.1 Multiple Objectives

Regarding the reward function of the HSA-MDP, we minimize the average response time of responders. We follow [14] in exploring linear combinations of δ_1 and δ_2 as $\texttt{reward} = \theta\delta_1 + (1-\theta)\delta_2$, for $0 \leq \theta \leq 1$. These two objectives basically oppose each other because they are coupled via resource constraints. Focussing on the full response δ_2 keeps fewer vehicles in reserve, fewer available to serve future calls causing a higher value of δ_1 in the future. On the other hand, focussing on δ_1 (i.e. sending only one responder and keeping rest as reserve) leads to a poor value of δ_2 per state.

Figure 6.1 shows the performance of HSA-HOP and our baseline evaluated on a fixed test set (240 calls between Jan 1-15, 2011). The training data for HSA-HOP also consists of 240 calls (between Jan 15-31, 2011). HSA-HOP randomly samples futures from the training data, so we average its performance over 30 trials and show the 95% confidence interval (from a Normal distribution).

The figure shows that as θ increases the average first response time of HSA-HOP is significantly smaller (by about 1 min) than the baseline. We see that the first response of HSA-HOP dominates the baseline for all values except $\theta = 0$. On the other hand, we see that the full responpse time HSA-HOP is slightly higher than the baseline.

In order to get a clearer understanding, the right panel of Figure 6.1 shows the fraction

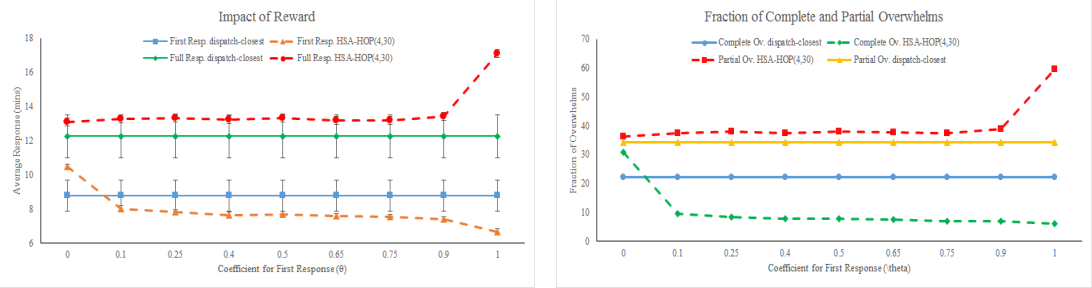

Figure 6.1: Performance of HSA-HOP with five responders under different reward functions of the form $\theta\delta_1 + (1-\theta)\delta_2$. Training data for HSA-HOP is randomly sampled from 240 calls in Jan 15-31, 2011. Testing data is a fixed sequence of 240 emergencies between Jan 1-15, 2011.

of calls left unanswered. The figure denotes *complete overwhelms* whenever $\delta_1 = \Delta$ (this implies $\delta_2 = \Delta$) i.e. no responders dispatched, in contrast to a *partial overhwelm* when $\delta_2 = \Delta$. We observe that the number of unanswered calls is significantly smaller than the baseline, from about 22% of calls to less than 10% with HSA-HOP. The fraction of partial overwhelms is only slightly higher than the baseline. This is acceptable because partial overwhelms are less severe by nature.

For this illustration, we used a lookahead $h = 4$ and $F = 30$ futures. We set $t = 10$ minutes but we found that the average optimization time was less than two minutes (about 1.6 minutes) averaged across states. We used five responders for this experiment, namely a Command, Engine, Ladder and two Ambulances. The next experiment varies the number of responders with a fixed reward function.

6.3.2 Fleet Size

In this experiment we set the reward function to be the average of the first response and the full response namely $\texttt{reward} = 0.5\delta_1 + 0.5\delta_2$. We show the relative performance of HSA-HOP and our baseline in terms of this reward function as well as the first and full response times. The training and test datasets are the same as the previous experiments.

In contrast to the previous experiment, here we test the scalability of HSA-HOP on problem instances of increasing state and action space. We show the details of the

# Responders	# State vars.	# Action vars.
4	38	16
5	43	20
6	48	24
7	53	28
8	58	32
9	63	36

Table 6.1: Size of problem instances vs. number of responders. The number of legal actions is exponential in the number of responders. The action space is boolean and the state space is hybrid.

problem instances in Table 6.1. We start with four responders namely one each of Command, Ambulance, Engine and Ladder. Each subsequent instance adds one responder in cyclic fashion i.e. add an Ambulance for the fifth instance, then an Engine for the sixth instance and so on. We do not add additional Command responders. One must note the high dimensional state and action spaces.

The left panel of Figure 6.2 shows the scalability of HSA-HOP and the effect of the fleet size on the response time. As expected, both the first and full response times decrease as more units are recruited. We observe that HSA-HOP has a clear advantage over the baseline in terms of first response time. For example, we see that HSA-HOP with five units has lower response time than the baseline using seven units. On the largest instance, HSA-HOP has a first response of 5.4 minutes, whereas the baseline policy has a first response of 6.2 minutes. The chart shows that the gain from planning with HSA-HOP is larger when few responders are available.

This lower first response time seems to come at the cost of slightly higher full response times. Intuitively, it makes sense to keep at least one Ambulance in reserve accounting for future emergencies that may arise before any busy responders become available again. As shown in the figure, the full response time of HSA-HOP is slightly higher than the baseline.

The right panel of Figure 6.2 shows the significance of the observed gains by showing the fraction of unanswered calls. The fraction decreases with more responders as one would expect. However, HSA-HOP drops only about 2% of calls on the largest problem instance, whereas the greedy baseline drops 7.5% of calls. This difference is exacerbated when fewer units are available, with HSA-HOP dropping about 8% with five units,

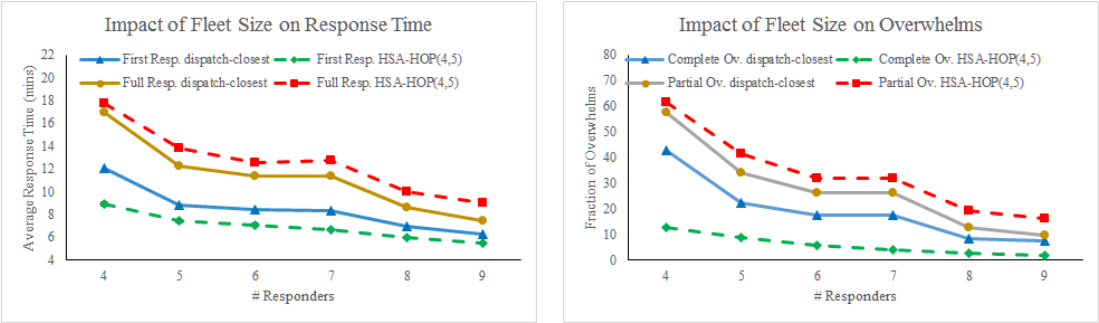

Figure 6.2: Performance of HSA-HOP under different number of responders. Training data for HSA-HOP is randomly sampled from 240 calls in Jan 15-31, 2011. Testing data is a fixed sequence of 240 emergencies between Jan 1-15, 2011.

whereas the baseline drops 22.5% of calls. Planning as in HSA-HOP is able to use additional responders effectively to improve the performance.

Table 6.2 shows the average reward and real time performance of HSA-HOP with varying parameters. In each case, the average reward of the baseline policy is dominated by HSA-HOP with some parametrization. In the smallest instance, we see that increasing the number of futures results in better performance with a narrower standard deviation when $h = 2$ as well as $h = 4$. This is the pattern we expect to see when the MILPs are optimally solved for each state. But, as the problem size increases we see that increasing the lookahead can often decrease the performance when the number of futures is fixed. Correspondingly, we see an order of magnitude increase in the translation and optimization time. HSA-HOP shows promise of scaling to these massive HSA-MDPs while providing some improvement over the baseline, but the scalability of the current implementation needs to be improved further.

6.4 Learning an RDDL Model

The experiments in the previous section show that the performance of the dispatcher can be improved using online planning as in HSA-HOP. We showed that with a minor modification to HSA-HOP, one can incorporate real-world data and circumvent the model specification for emergencies. Whereas this approach increases the applicability of HSA-

Responders	h	F	Avg. Reward (Std. Err.)	Opt. Time (mins)	Trans. Time (mins)	MILP Vars.	MILP Constraints
closest			-804.874				
4	2	5	-567.606 (13.811)	0.006	0.006	11996	24836
		50	-564.037 (7.325)	0.137	0.138	119137	247537
	4	5	-601.396 (14.752)	0.027	0.027	25047	52277
		50	-563.287 (13.878)	1.197	1.234	249531	521831
closest			-460.246				
5	2	5	-356.865 (12.269)	0.009	0.01	14093	29238
	4	5	-414.204 (20.283)	0.058	0.06	29531	61756
		30	-379.95 (14.201)	1.849	1.786	176658	370008
closest			-364.725				
6	2	5	-285.501 (12.867)	0.025	0.018	16298	33828
		50	-270.976 (8.256)	2.723	2.618	162147	337447
	4	5	-317.655 (18.427)	0.209	0.187	34214	71594
		50	-331.109 (17.969)	4.747	4.615	341132	714932
closest			-185.935				
8	2	5	-146.015 (7.547)	1.216	1.181	20505	42645
		30	-146.579 (5.79)	6.142	5.756	122543	255383
	4	5	-193.427 (15.118)	4.417	4.228	43172	90542
closest			-154.586				
9	2	5	-127.71 (7.15)	2.169	2.089	22604	47049
		30	-128.966 (7.998)	6.746	6.602	135141	281811
	4	5	-162.699 (15.303)	5.401	5.178	47663	100028
	4	30	-302.514 (28.385)	2.71	2.667	285415	599605

Table 6.2: Average reward of the dispatch-closest policy and HSA-HOP with standard deviation over 30 repetitions in paranthesis. We vary the number of responders, lookahead (h) i.e. the number of actions to lookahead and the number of futures (F). Training data : 240 calls between January 15-31, 2011. Testing data : 240 calls between January 1-15,2011. The timing information is averaged over the test states with a per state time limit of 10 minutes imposed on the MILP solver.

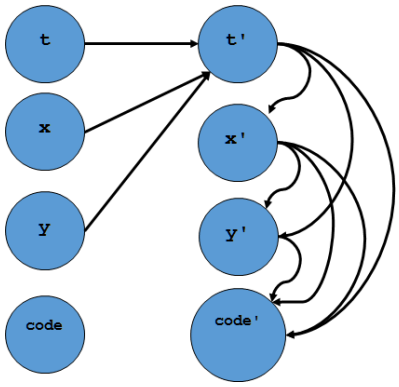

Figure 6.3: DBN representation of the model learned for the emergency domain.

HOP and avoids issues with model misspecification, the approach may be considered a bit adhoc. One possible issue is the inability to include novel emergencies in the planning algorithm when limited training data is available. In contrast to the previous section, we used five years of data consisting of 35122 calls for learning.

Our motivation is to evaluate HSA-HOP in comparison to the previous section. Firstly, we test the impact of generalization on its performance compared to the previous experiment. Secondly, we test the impact of overfitting on scalability in terms of translation and optimization time. The complexity of the learned model has an adverse effect on the planning time. In particular, overfitting in the PWL model space leads to a larger number of pieces, whereas the planning time grows with the number of pieces.

Model learning as in supervised machine learning is the standard way of dealing with this issue. For our purposes, the learned model must be in the form of RDDL expressions, and in particular must satisfy the requirements of Definition 4. Accordingly, we illustrate some fundamental models used in supervised learning for the purpose of modelling the emergency state. The distributional assumptions of our model for the forecasting of emergencies follows many prior works in operations research, in the context of which online action selection using such expressive models is a novel approach.

The target distribution is $P(x', y', t', \texttt{code}'|s, a)$ where s, a are the current state and action respectively. We assume that this conditional distribution only depends on one part of the current state as in $P(x', y', t', \texttt{code}'|s, a) = P(x', y', t', \texttt{code}'|x, y, t)$. Furthermore, we factorize this as shown in Figure 6.3.

$$P(x', y', t', \texttt{code}'|x, y, t) = P(t'|x, y, t) \times P(x'|t') \times P(y'|x', t') \times P(\texttt{code}'|x', y', t')$$

As alluded above, we assumed the existence of an interarrival distribution $\delta(x, y, t)$ that affects t' through t. We assume that (x', y') is independent of (x, y), but depends on t via t'. Although the target distribution can be factorized in different ways, we found that `code` is a strong function of (x, y, t). For example, when the call originates from a hospital, the nature code is always 'Transport' (a patient).

1. **Interarrival Time** $P(t'-t|t)$: The natural choice for this positive valued random variable is the exponential distribution. The underlying assumption is that the emergencies occur IID and without memory and the number of calls form a poisson process as a result. Figure 6.4 shows the histogram of Gap (interarrival time). Our initial guess seems accurate by the shape of the empirical distribution. We observe that the mode is not zero, motivating the use of the more general Gamma distribution. The standard exploratory analysis for the gamma regression model is shown in Figure 6.5, showing the presence of a linear relationship between the inverse of the gap and Time. In addition, the deviations from the linear trend appears to be non-constant and multiplicative.

   In order to satisfy Defintion 4, we must assume that the rate parameter is a PWL function of the time of the day. The model is of the form $\delta = t'-t \sim \texttt{Gamma}(\frac{1}{\phi}, \theta^T\Phi)$ where Φ are basis functions dependent on time. In the HSA-MDP case, $\Phi(t)$ may be continuous (e.g. t itself) or discrete (e.g. $\mathbf{1}(t > 5)$). A constant dispersion parameter ϕ is estimated from the data and so the shape parameter of the underlying distribution is assumed to be independent of time. Thus, the mean and variance of δ are $E[\delta] = \frac{1}{\phi}\frac{1}{\theta^T\Phi}$ and $V[\delta] = \frac{1}{\phi}\frac{1}{(\theta^T\Phi)^2}$.

   We use the approach of Multivariate Adaptive Regression Splines [50] to estimate the parameters as well as to find good basis functions. This is similar to decision tree regression consisting of two phases namely, the forward phase of building a fixed depth tree wrt the training data, and the backward phase of pruning wrt a separate validation set. The leaf nodes of the tree are Gamma distributions with the canonical link function whose parameters are estimated using the theory of

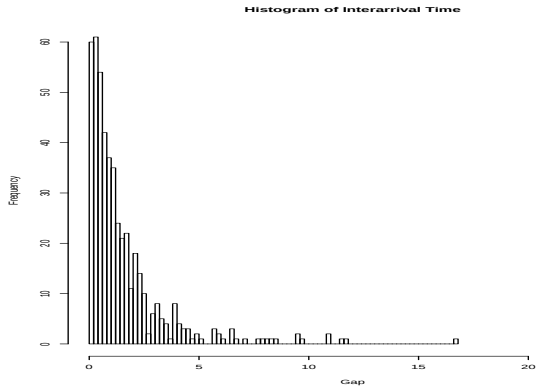

Figure 6.4: Histogram of observed interarrival time (x-axis in minutes).

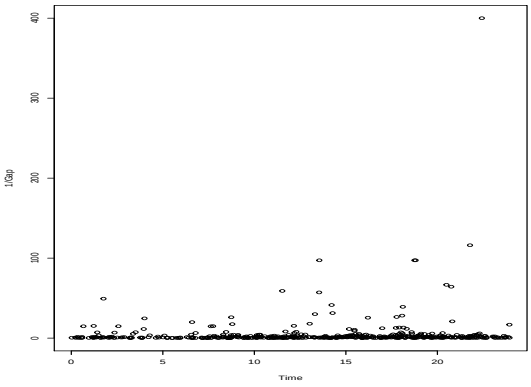

Figure 6.5: Linear relationship between the inverse of the interarrival gap and the time of the day.

Generalized Linear Models (GLM) [103]. We used an existing statistical package `earth` [108] which is based on the `mars` package. The parameters of the learner are the maximum number of basis functions and a threshold value for pruning. For the purpose of illustration, we used the default settings for `earth` but using "exhaustive" pruning, we arrived upon the model given below.

Figure 6.6 illustrates the model fit on the observed training data and testing data. The figures show a kernel density plot of the number of calls as a function of time of the day. The curve denoted "observed" corresponds to the training and testing data of the true t' in the respective figures. The curve denoted "predicted" is the kernel density plot of the t' predicted using the previous time t as $t' = t + \delta$.

```
GLM g: null.deviance 23362.88 (17560 dof)   deviance 21888.09 (17554
    dof)   iters 7

Call:
glm(formula = yarg ~ ., family = family, data = bx.data.frame,
    weights = weights, na.action = na.action, control = control,
    model = TRUE, method = "glm.fit", x = TRUE, y = TRUE, contrasts =
        NULL,
    trace = (trace >= 2))

Deviance Residuals:
```

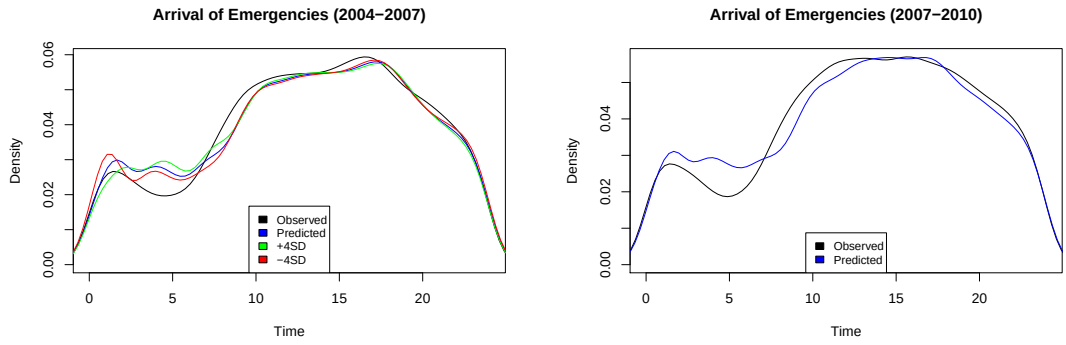

Figure 6.6: Kernel density plot of the piece-gamma fit for the interarrival gap (minutes) as a function of the time of the day. The left and right panels show the fit for the training and testing datasets respectively. The curve denoted "observed" corresponds to the training (left) and testing (right) data of the true t' in the respective figures. The curve denoted "predicted" is the kernel density plot of the t' predicted using the previous time t as $t' = t + \delta$.

```
    Min       1Q   Median       3Q      Max
-3.7612  -1.0973  -0.3869   0.3356   4.9446

Coefficients:
                Estimate Std. Error z value Pr(>|z|)
(Intercept)     0.335299   0.009796  34.228  < 2e-16 ***
`h(t-16.7833)` -0.058277   0.004481 -13.006  < 2e-16 ***
`h(t-9.01667)` -0.070491   0.004898 -14.392  < 2e-16 ***
`h(y-65.7576)`  0.070939   0.016222   4.373 1.23e-05 ***
`h(t-3.6)`      0.070466   0.002866  24.585  < 2e-16 ***
`h(y-64.5834)` -0.130102   0.035653  -3.649 0.000263 ***
`h(y-64.0341)`  0.071717   0.025193   2.847 0.004417 **
---
Signif. codes:  0    ***    0.001    **    0.01    *    0.05    .
    0.1         1

(Dispersion parameter for Gamma family taken to be 1.070864)

    Null deviance: 23363  on 17560  degrees of freedom
Residual deviance: 21888  on 17554  degrees of freedom
AIC: 54671
```

```
Number of Fisher Scoring iterations: 7
```

Note that the hinge functions learnt by `mars` of the form $h(ax+b) = \max[0, ax+b]$ are PWL and allowed by Definition 4. In order to sample futures, one only needs to know that the Gamma distribtion is a scale family. Since the dispersion parameter is close to one, we used an exponential distribution because of its closed form CDF. A sample is drawn as $-\log(u) * (\theta^T \Phi)$ where $-\log(u)$ is the inverse CDF of the standard exponential distribution and $u \sim U(0, 1)$.

2. **Location** $P(x', y'|t')$: The two-dimensional location of emergencies follows a distribution similar to the density of population in different neighborhoods and the major roads of Corvallis. The distribution of emergencies consists of distinct clusters of calls separated by regions of low or no call density. For example, the West Hills Assisted Living center in the southwest part of town is a region with a high frequency of emergencies separate from the other modes near the center of the city as well as another cluster seen in the south part of town.

   We use a gaussian mixture model as the continuous state space model. One significant obstacle is that the current language specification of RDDL does not allow any multivariate state variable. This prevents us in modelling the locations as a mixture of bivariate gaussian distributions. The only workaround is to factorize $P(x', y'|t')$ using the marginal distribution of x' ie. $P(x'|t')$ and the conditional distribution of y' conditioned on x' ie. $P(y'|x', t')$. The former is modeled as a univaritate gaussian mixture model (GMM) whose mean is assumed to be a linear function of time. The latter is also modeled as a GMM whose mean is aassumed to be linear function of x and time. These assumptions are necessary to satisfy Definition 4.

   The GMM can be encoded using the PWL machinery as below. Let $\theta_i \in [0, 1]$ denote the mixture probability, $\mu_i(t)$ denote the mean and, $\sigma_i > 0$ denote the constant variance of the i^{th} component of the GMM with n components. As before, let $u \sim U(0, 1)$, then a random future is sampled by first sampling the discrete random variable corresponding to the component index, followed by sampling from the corresponding gaussian distribution. We note the necessity for discrete and

Components	LogLik	BIC	training.error	testing.error
1	-42826.10	85691.30	2.856958	2.836998
2	-35214.71	70507.60	2.429330	2.395086
3	-34417.74	68952.75	2.385163	2.319646
4	-34250.94	68658.25	2.376528	2.360736
5	-33891.41	67978.29	2.350192	2.343996

Table 6.3: Training and Testing error (in miles) vs. the number of GMM components for x locations.

continuous random variables. The piecewise functions covered by Definition 4 easily permit the GMM model.

$$S_j = \sum_{j=1}^{i} \theta_j; S_0 = 0$$
$$x = \sum_{j=1}^{n} (u \leq S_j \wedge u > S_{j-1}) x_j$$
$$x_j \sim N(\mu_j(t), \sigma_j) \text{ using Proposition 15}$$

Note that the model does not have knowledge of the geography near Corvallis. The model may predict emergencies near rivers or sparsely populated regions. However, for the purpose of planning of dispatch actions, the model is sufficient if the predictions have a similar manhattan distance as observed emergencies. As mentioned above, it is possible to encode a more complex model including the real road network.

Table 6.3 (Table 6.4) shows the model fit with varying number of components for the x component (y component, respectively). We found that increasing the number of components to five reduces both the training and testing error, as well as the BIC. The error is measured in terms of euclidean distance between the observed and predicted (x, y) locations. The parameters of the GMM were optimized using the existing `mixtools` package [12] that uses the EM algorithm. Increasing the number of components would lead to better fits to the training data.

Components	LogLik	BIC	training.error	testing.error
1	-46277.17	92603.22	3.456872	3.502303
2	-36035.41	72168.56	2.756012	2.857903
3	-35479.95	71106.49	2.775615	2.867790
4	-35579.51	71354.49	2.741688	2.834325
5	-33621.00	67486.34	2.862767	2.884020

Table 6.4: Training and Testing error (in miles) vs. the number of GMM components for y locations.

However, we found that HSA-HOP is not able to handle more than three components, in terms of the size of the MILP produced under a given memory limit, as well as their translation and optimization timings. We were able to evaluate HSA-HOP with the following GMM with three components, where λ denotes the mixing proportions, σ denote the standard deviation, β_1 is the intercept, β_2 is the coefficient corresponding to t and β_3 corresponds to x. Figure 6.7 illustrates the model fit for x and y locations.

```
number of iterations= 239
summary of regmixEM object:
            comp 1       comp 2      comp 3
lambda 3.29666e-02  4.02628e-01 5.64406e-01
sigma  5.98178e+00  2.34516e+00 6.87869e-01
beta1  1.40558e+03  1.41578e+03 1.41693e+03
beta2  1.31019e-02 -2.18833e-03 4.53499e-03
loglik at estimate:  -34417.74
```

```
number of iterations= 420
summary of regmixEM object:
             comp 1       comp 2       comp 3
lambda    0.1321839  0.000420624  8.67395e-01
sigma     8.0110470  0.009319536  1.15126e+00
beta1  -429.0252604 91.940029476 -8.96243e+02
beta2    -0.0229183  0.006408035 -2.55224e-03
beta3     0.3461729 -0.018789849  6.78642e-01
loglik at estimate:  -36031.5
```

---

3. **Nature Code** : The nature code is a discrete random variable. Nature codes can be broadly categorized into medical causes, such as Code-1, Code-3, Code3Trauma,

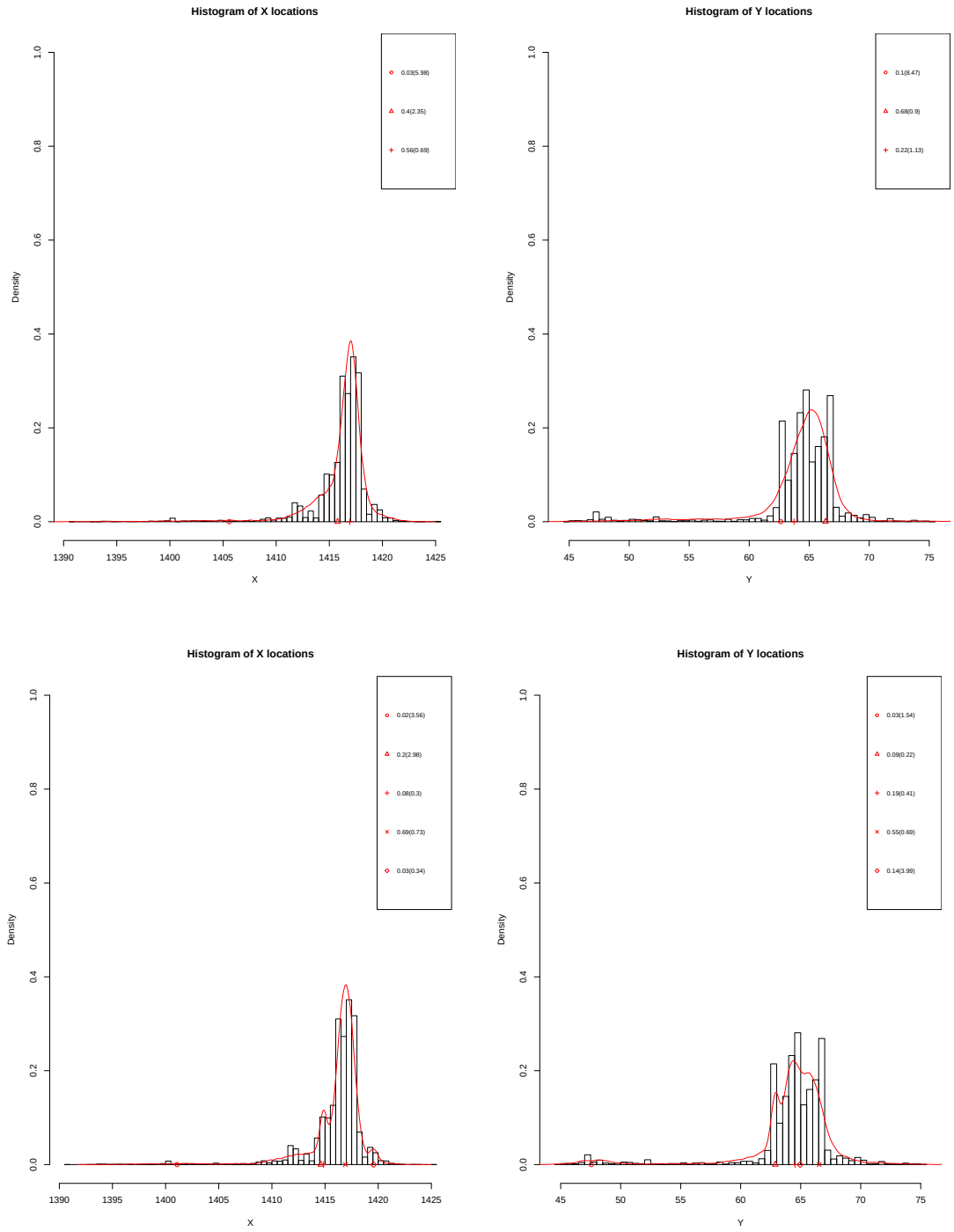

Figure 6.7: Kernel Density plot of the x (left) and y (right) locations predicted by the Gaussian Mixture Model with three (top) and five (bottom) components. The histogram represents the true and observed locations.

EMS, and fire related causes, such as Hazmat, MVA, StructureFire, VehicleFire, WildlandFire and so on, for a total of 16 nature codes. The basic model for such a random variable is multinomial regression, which works by modelling the log-odds ratio as a linear model.

$$\pi_x = \log(\frac{p_x}{p_{ref}}) = \alpha_1 x + \alpha_2 y + \alpha_3 t$$

where p_x is the PMF for outcome x, and ref denotes an arbitrary category that is chosen as the reference. We used the existing `multinom` package to estimate the coefficients. The linearity assumption made above is justified below.

The criterion of Definition 4 for discrete random variables requires the PMF to be a linear function, whereas the PMF of the multinomial model is the softmax function which is an exponential function of the parameters. Intuitively, we can use the monotonicity of the transformation to sample directly in the log-odds space in order for Definition 4 to hold. For example, the RDDL expression $\max_x \pi_x$ is also PWL if each log-odds ratio π_x is PWL. But, this expression only returns the most likely outcome which is not the same as the softmax.

The correct way to sample from the PMF is to compare the log-odds ratios to random samples from the Gumbel distribution [60]. In our model, we used a separate binary variable $\mathbf{1}(x)$ for the indicator that `code`$=x$ then the following RDDL expression sets exactly one $\mathbf{1}(x)$ to `true`.

$$\mathbf{1}(x) = \forall_{y:cause;y \neq x}(\pi_x - \pi_y \geq \eta_x - \eta_y)$$

where, $\pi_x - \pi_y$ captures the log-odds of outcome x vs. outcome y, and is PWL whenever π_i is PWL for all i.

The random variables η_i are IID distributed as a standard Gumbel distribution. The difference of the random variables $\eta_x - \eta_y$ then follows a standard logistic distribution [60]. Thus, the inner expression is true with probability equal to the lower-tail of the logistic distribution $P(\eta_x - \eta_y \leq \pi_x - \pi_y)$. The outer universal quantification ensures that only outcome is set and corresponds to an argmax. Finally, the known inverse CDF for the Gumbel distribution is $-\log(-\log(u))$ for

	h	F	δ_1 (mins)	Compl. Ov.	δ_2 (mins)	Partial Ov.	Opt. Time	Trans. Time
closest			7.11 (1.898)	12.5	9.20 (2.721)	20		
HSA-HOP	3	3	6.15 (0.251)	4.21	10.80 (0.534)	27.5	0.02	0.02
HSA-HOP*	3	3	5.79 (0.256)	2.8	11.13 (0.528)	27.92	6.727	0.125
HSA-HOP*	3	5	5.99 (0.261)	3.17	11.39 (0.525)	28.42	4.76	0.464
HSA-HOP*	3	10	7.38 (0.337)	12.42	12.73 (0.549)	35.5	3.46	3.765
HSA-HOP*	4	3	6.84 (0.315)	8.42	12.15 (0.539)	32.25	8.96	0.144
HSA-HOP*	5	3	8.57 (0.385)	20.17	13.94 (0.559)	41.17	8.947	0.364

Table 6.5: Impact of generalization on one problem instance consisting of five responders evaluated against the first 40 emergencies in January 2011. A 95% confidence interval is shown for the average response times δ_1 and δ_2. The columns denoting partial and complete overwhelms are the fraction out of 40 calls (%). All times are shown in minutes.

$u \sim U(0, 1)$.

Impact of Generalization : Two versions of HSA-HOP are shown in Table 6.5. The algorithm denoted HSA-HOP is the version presented in the earlier experiments that samples futures from the dataset. The version denoted HSA-HOP* is HSA-HOP that samples futures from the learned model presented above. We immediately see the impact of generalization on the first response time and the fraction of unanswered calls. HSA-HOP* shows that the first response can be reduced even further, while at the same time reducing the fraction of complete overwhelms of HSA-HOP.

The improved performance is achieved at the cost of significantly higher optimization time. HSA-HOP* uses a significantly larger problem description consisting of the learned RDDL model for the emergencies, in addition to the action model used by HSA-HOP, that leads to larger MILPs solved at each state. These timings give context on our earlier remark about planning with a higher number of components in our mixture model. Finally, similar to HSA-HOP, we see that increasing the lookahead and number of futures causes problems in terms of overall performance that is exacerbated by the significantly larger problem description in HSA-HOP*.

Chapter 7: Concluding Remarks

It is well known that the planning problem in MDPs is very hard in theory, and it is challenging to design algorithms whose performance scales well with the size of the MDP. From a practitioner's point of view, an algorithm is more likely to be useful when a compact description of the problem can be given as input, and it is used to efficiently select actions from a large and potentially infinite action space.

Our contributions take a step in this direction inspired by many real-world applications with natural factored action spaces, and further motivated by the recent international planning competition benchmarks where factored action MDPs are increasingly seen. In these benchmarks, existing MDP and factored MDP planners show inferior performance when the number of factored actions is increased. Across our algorithms, the recurring message is the use of expressive and compact symbolic representations for computing action values in a factored manner, which in turn lead to significantly better scalability and improved performance than treating each of the exponentially many actions as atomic.

The most successful anytime planners in planning competitions work by generating domain-independent heuristics using flat MDP relaxations of the factored MDP, e.g. using determinization [87], domain analysis [80] along with a gamut of heuristics for focussing the search under tight time constraints. Although these techniques improve the empirical anytime performance on the benchmarks, they are heuristic in the sense that they do not provide explicit generalization in a sound manner. Symbolic Real-Time Dynamic Programming (sRTDP) [44, 46] aims to combine the benefits of the symbolic methods and online planning by incorporating symbolic state generalization into the computation of the online planner. However, sRTDP is a general framework, and its performance is sensitive to the definition of generalized states. Existing definitions in prior work lead to algorithms that exceed memory limits in many cases. Despite the aim for generalization, the resulting planner is often inferior to the corresponding algorithms working in the flat state space (e.g. RTDP). We presented the first fully symbolic anytime planning algorithms for factored MDPs that generalize simulated experience soundly

and efficiently. We observed that using the appropriate form of generalization was the dominating factor towards good performance compared to increasing the time-limit for algorithms without state generalization. We also saw that the most appropriate form of generalization can differ across domains and sometimes problem instances within a domain. This suggests that it is fruitful to investigate mechanisms for tuning or selecting among generalization methods.

This work is related to the problem of state abstraction in MDPs that is the problem of finding equivalence relations between states. Our generalization operators in Chapter 4 are able to discover restricted versions of value and policy irrelevance abstractions of [92], and exploits them in asychronous DP updates. A promising direction for future work is to incorporate bisimilarity metrics [47, 48, 48] into the online algorithm similar to the work of [127, 67]. In contrast to [127, 67], symbolic methods are unique in that they allow abstract states to be merged and split efficiently in an online fashion.

In our experiments we used replanning at every state to output one action. Our approach paves the way for speedup learning for the purpose of generalizing the online search across different initial states i.e. intra-problem generalization. The crucial difference over traditional learning approaches is that the symbolic examples are generalized and succinct. It is clear that an exact merging of these BDD policies (union over BDDs) would not be scalable, but some alternatives are available. Following the work of [117], pruning functions can be learned that are partial policies subscribing some set of actions including the optimal action. This is a well-studied problem in BDDs and one might use the affine BDD approximation [65, 155], Horn approximations [133, 42] or any class of boolean functions with known closure operators for BDDs [131].

Approaches based on Value Function Approximation [138] have been extended to handle factored state spaces [85, 59] and factored actions [57, 58, 89], but their performance crucially depends on domain specific basis functions that must be engineered extensively in practice. The work of [57] motivated a factored action space for modelling fully cooperative multiagent problems with centralized controls, and give approximate algorithms using factored basis functions within this framework. These methods are generalized to the hybrid setting by Hybrid Approximate Linear Programming (HALP) [89], which seek to approximate value functions via a weighted basis function representation. On the policy approximation side, approaches like Pegasus [113] seek to optimize restricted parameterized policy classes subject to sampled trajectories. All of these methods assume

a priori knowledge of a good value or policy representation, which is typically difficult to have in advance.

Further afield, Monte Carlo Tree Search (MCTS) [23] is a popular framework for action selection in MDPs. In contrast to the above approaches, as well as our contributions, MCTS is a different approach by nature that works by sampling action outcomes from a generative model of the MDP. Their performance is also adversely affected by large action spaces, which can be alleviated by heuristics [51] and action pruning [96, 117]. Due to the generative model assumption, a large portion of prior work on MCTS ignores any declarative factored structure, with recent exceptions of [80] and [29, 28]. The recent works of [29, 28] on Factored MCTS only produce optimal policies under strong assumptions whereas we explore both optimal and anytime optimal factored algorithms under common setting that the state and action variables are discrete valued. Chapter 5 extends to the more general hybrid setting where we contribute a factored implementation of Hindsight Optimization. The HOP approach is related to MCTS in that it approximates the tree search problem with a set of determinsitic trees that are loosely-coupled.

The expressive subset of HSA-MDPs consists of location-scale probability distributions for the transitions of each state variable. The importance of location-scale distributions is that they allow a compile-time encoding of the sampled next states at some time step in the future, even though the parameters of the corresponding density function involves state and action variables that are unknown at compile time. In our instantiation of this approach, the MILP solver then maximizes over the unknown policy over multiple sampled futures when the parameters are piecewise linear functions of state and action variables, or location-scale distributions over them. The general approach is more broadly applicable by using a more complex solver that matches the function space of the parameters. To the best of our knowledge, this is the first characterization of a subset of HSA-MDPs, and corresponding fragment of the RDDL planning domain description language, that have a linear time compilation to an MILP of linear size.

Future work is needed in the algorithm space for HSA-MDPs because HSA-HOP is a heuristic algorithm by nature of the HOP approximation. Future work must alleviate this and explore MILP reductions of sound (e.g. RTDP) and sample-efficient algorithms (e.g. Sparse Sampling) for HSA-MDPs. The challenge is to retain the scalability of the HOP approach that is able to scale to truly massive HSA-MDPs. However, for the time

being HSA-HOP serves as a baseline for these MDPs.

Another potential direction for future work is the learning of models within the stochastic PWL fragment. Recent advances in learning bayesian networks and learning of multi-layered neural networks can be tapped. In fact, the commonly used convolutional neural network with rectified linear activations, if used to model the one step dynamics, is exactly equivalent to the deterministic PWL class. While HSA-HOP can take these expressive models as input to derive model predictive controllers, the neural models are significantly larger in terms of the number of neurons compared vs the number of state and action variables.

We considered the dispatch optimization problem faced by responders of 9-1-1 emergencies by casting it as an HSA-MDP. We adapted the HSA-HOP algorithm to work with real world data directly. The generality of our HSA-MDP formulation, specifically the stochastic PWL fragment, is demonstrated in the learning experiments. We found that HSA-HOP compares favorably to the baseline policy commonly used by the dispatcher. Our experiments showed significant improvement in the response time to emergencies that occured in Corvallis.

Bibliography

[1] Lina Aboueljinane, Evren Sahin, and Zied Jemai. A Review on Simulation Models Applied to Emergency Medical Service Operations. *Computers & Industrial Engineering*, 66(4):734–750, 2013.

[2] Angelos Angelidakis and Georgios Chalkiadakis. Factored MDPs for Optimal Prosumer Decision-Making. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 503–511. International Foundation for Autonomous Agents and Multiagent Systems, 2015.

[3] R Aringhieri, ME Bruni, S Khodaparasti, and JT van Essen. Emergency Medical Services and Beyond: Addressing New Challenges Through a Wide Literature Review. *Computers & Operations Research*, 2016.

[4] Mohd Hafiz Azizan, Cheng Siong Lim, Go TL Hatta WALWM, and SS Teoh. Simulation of Emergency Medical Services Delivery Performance Based on Real Map. *International Journal of Engineering and Technology*, 5(3):2620–2627, 2013.

[5] R.I. Bahar, E.A. Frohm, C.M. Gaona, G.D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebric Decision Diagrams and their Applications. *Formal Methods in System Design*, 10(2):171–206, 1997.

[6] Radha-Krishna Balla and Alan Fern. Uct for Tactical Assault Planning in Real-Time Strategy Games. In *Twenty-First International Joint Conference on Artificial Intelligence*, 2009.

[7] AG Barto and RH Crites. Improving Elevator Performance using Reinforcement Learning. *Advances in neural information processing systems*, 8:1017–1023, 1996.

[8] Andrew G Barto, Steven J Bradtke, and Satinder P Singh. Learning to Act Using Real-Time Dynamic Programming. *Artificial Intelligence*, 72(1):81–138, 1995.

[9] Valérie Bélanger, Yannick Kergosien, Angel Ruiz, and Patrick Soriano. An Empirical Comparison of Relocation Strategies in Real-Time Ambulance Fleet Management. *Document De Recherche Du CIRRELT*, 2014.

[10] Valérie Bélanger, Ettore Lanzarone, Angel Ruiz, and Patrick Soriano. The Ambulance Relocation and Dispatching Problem. *Document De Recherche Du CIRRELT*, 2015.

[11] Valérie Bélanger, Angel Ruiz, and Patrick Soriano. *Recent Advances in Emergency Medical Services Management.* Tech. Rep. CIRRELT-2015-28, CIRRELT, 2015.

[12] Tatiana Benaglia, Didier Chauveau, David R. Hunter, and Derek Young. mixtools: An R Package for Analyzing Finite Mixture Models. *Journal of Statistical Software*, 32(6):1–29, 2009.

[13] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming.* 1996.

[14] Ronald Bjarnason, Prasad Tadepalli, Alan Fern, and Carl Niedner. Simulation-Based Optimization of Resource Placement and Emergency Response. 2009.

[15] Blai Bonet and Hector Geffner. Labeled RTDP : Improving the Convergence of Real-Time Dynamic Programming. In *ICAPS*, volume 3, 2003.

[16] C. Boutilier, T. Dean, and S. Hanks. Planning under Uncertainty: Structural Assumptions and Computational Leverage. In *Proceedings of the Second European Workshop on Planning*, pages 157–171, 1995.

[17] C. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller. Context-Specific Independence in Bayesian Networks. In *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence*, pages 115–123, 1996.

[18] Craig Boutilier, Thomas Dean, and Steve Hanks. Decision-Theoretic Planning: Structural Assumptions and Computational Leverage. *J. of Artif. Intell. Res.*, 11(1):94, 1999.

[19] Craig Boutilier, Richard Dearden, and Moises Goldszmidt. Exploiting Structure in Policy Construction. In *International Joint Conference on Artificial Intelligence*, volume 14, pages 1104–1113, 1995.

[20] Justin A Boyan and Michael L Littman. Exact Solutions to Time-Dependent MDPs. In *Advances in Neural Information Processing Systems*, pages 1026–1032, 2001.

[21] Yuri Breitbart, H Hunt, and Daniel Rosenkrantz. On the Size of Binary Decision Diagrams Representing Boolean Functions. *Theoretical Computer Science*, 145(1):45–69, 1995.

[22] Luce Brotcorne, Gilbert Laporte, and Frederic Semet. Ambulance Location and Relocation Models. *European journal of operational research*, 147(3):451–463, 2003.

[23] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.

[24] Randal E Bryant. Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams. *ACM Computing Surveys (CSUR)*, 24(3), 1992.

[25] H. S. Chang, R. L. Givan, and E. K.P. Chong. Online Scheduling via Sampling. In *Proceedings of the Conference on Artificial Intelligence Planning and Scheduling*, 2000.

[26] E. K.P. Chong, R. L. Givan, and H. S. Chang. A Framework for Simulation-Based Network Control via Hindsight Optimization. In *IEEE CDC '00: IEEE Conference on Decision and Control*, 2000.

[27] Kenneth Chun-Ling Chong. *Models for Decision-Making and Performance Evaluation in Emergency Medical Service Systems.* PhD thesis, Cornell University, 2016.

[28] Hao Cui and Roni Khardon. Online Symbolic Gradient-Based Optimization for Factored Action MDPs. IJCAI, 2016.

[29] Hao Cui, Roni Khardon, Alan Fern, and Prasad Tadepalli. Factored MCTS for Large Scale Stochastic Planning. In *AAAI*, pages 3261–3267, 2015.

[30] Adnan Darwiche. Recursive Conditioning. *Artif. Intell.*, 126:5–41, February 2001.

[31] Adnan Darwiche and Pierre Marquis. A Knowledge Compilation Map. *Journal of Artificial Intelligence Research*, 17(1):229–264, 2002.

[32] Jessica Davies and Fahiem Bacchus. Exploiting the Power of MIP Solvers in MAXSAT. In *Theory and Applications of Satisfiability Testing–SAT 2013*, pages 166–181. Springer, 2013.

[33] Stephen F Dean. Why the Closest Ambulance Cannot be Dispatched in an Urban Emergency Medical Services System. *Prehospital and Disaster Medicine*, 23(02):161–165, 2008.

[34] Thomas Dean and Keiji Kanazawa. A Model for Reasoning about Persistence and Causation. *Computational intelligence*, 5(2):142–150, 1989.

[35] Boris Defourny, Damien Ernst, and Louis Wehenkel. Multistage Stochastic Programming: A Scenario Tree Based Approach. *Decision Theory Models for Applications in Artificial Intelligence: Concepts and Solutions: Concepts and Solutions*, page 97, 2011.

[36] Josée Desharnais, Abbas Edalat, and Prakash Panangaden. A Logical Characterization of Bisimulation for Labeled Markov Processes. In *Logic in Computer Science, 1998. Proceedings. Thirteenth Annual IEEE Symposium on*, 1998.

[37] Thomas G Dietterich, Majid Alkaee Taleghan, and Mark Crowley. Pac Optimal Planning for Invasive Species Management: Improved Exploration for Reinforcement Learning from Simulator-Defined MDPs. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*, pages 1270–1276. AAAI Press, 2013.

[38] Carmel Domshlak and Jörg Hoffmann. Fast Probabilistic Planning through Weighted Model Counting. In *ICAPS*, pages 243–252, 2006.

[39] James George Dunham. Optimum Uniform Piecewise Linear Approximation of Planar Curves. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (1):67–75, 1986.

[40] Stefan Edelkamp, Shahid Jabbar, and Blai Bonet. External Memory Value Iteration. In *ICAPS*, pages 128–135, 2007.

[41] Hélène Fargier and Pierre Marquis. Extending the Knowledge Compilation Map: Closure Principles. In *ECAI*, pages 50–54, 2008.

[42] Hélene Fargier and Pierre Marquis. Extending the Knowledge Compilation Map: Krom, Horn, Affine and Beyond. In *AAAI*, pages 442–447, 2008.

[43] Hélene Fargier, Pierre Marquis, Alexandre Niveau, and Nicolas Schmidt. A Knowledge Compilation Map for Ordered Real-Valued Decision Diagrams. In *AAAI*, pages 1049–1055, 2014.

[44] Z. Feng, E.A. Hansen, and S. Zilberstein. Symbolic Generalization for Online Planning. In *Proceedings of the Nineteenth conference on Uncertainty in Artificial Intelligence*, pages 209–216. Morgan Kaufmann Publishers Inc., 2002.

[45] Zhengzhu Feng, Richard Dearden, Nicolas Meuleau, and Richard Washington. Dynamic Programming for Structured Continuous Markov Decision Problems. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 154–161. AUAI Press, 2004.

[46] Zhengzhu Feng and Eric A Hansen. Symbolic Heuristic Search for Factored Markov Decision Processes. In *AAAI/IAAI*, pages 455–460, 2002.

[47] Norm Ferns, Prakash Panangaden, and Doina Precup. Metrics for Finite Markov Decision Processes. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 162–169. AUAI Press, 2004.

[48] Norman Ferns, Pablo Samuel Castro, Doina Precup, and Prakash Panangaden. Methods for Computing State Similarity in Markov Decision Processes. *arXiv preprint arXiv:1206.6836*, 2012.

[49] Norman Ferns, Prakash Panangaden, and Doina Precup. Metrics for Markov Decision Processes with Infinite State Spaces. *arXiv preprint arXiv:1207.1386*, 2012.

[50] Jerome H Friedman. Multivariate Adaptive Regression Splines. *The Annals of statistics*, pages 1–67, 1991.

[51] Sylvain Gelly and David Silver. Monte-carlo Tree Search and Rapid Action Value Estimation in Computer Go. *Artificial Intelligence*, 175(11):1856–1875, 2011.

[52] Supriyo Ghosh and Pradeep Varakantham. Strategic Planning for Setting up Base Stations in Emergency Medical Systems. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2016.

[53] Robert Givan, Thomas Dean, and Matthew Greig. Equivalence Notions and Model Minimization in Markov Decision Processes. *Artificial Intelligence*, 147(12):163 – 223, 2003.

[54] Jeffrey B Goldberg. Operations Research Models for the Deployment of Emergency Services Vehicles. *EMS management Journal*, 1(1):20–39, 2004.

[55] Ignacio E Grossmann, Robert M Apap, Bruno A Calfa, Pablo Garcia-Herreros, and Qi Zhang. Recent Advances in Mathematical Programming Techniques for the Optimization of Process Systems under Uncertainty. In *12th International Symposium on Process Systems Engineering and 25th European Symposium on Computer Aided Process Engineering*, volume 37, page 1. Elsevier, 2015.

[56] Joshua T Guerin, Josiah P Hanna, Libby Ferland, Nicholas Mattei, and Judy Goldsmith. The Academic Advising Planning Domain. In *Proceedings of the 3rd Workshop on the International Planning Competition at ICAPS*, pages 1–5, 2012.

[57] C. Guestrin, D. Koller, and R. Parr. Multiagent Planning with Factored MDPs. *Advances in neural information processing systems*, 14:1523–1530, 2001.

[58] Carlos Guestrin, Milos Hauskrecht, and Branislav Kveton. Solving Factored MDPs with Continuous and Discrete Variables. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 235–242. AUAI Press, 2004.

[59] Carlos Guestrin, Daphne Koller, Ronald Parr, and Shobha Venkataraman. Efficient Solution Algorithms for Factored MDPs. *Journal of Artificial Intelligence Research*, 19:399–468, 2003.

[60] Emil Julius Gumbel. *Statistical theory of extreme values and some practical applications: a series of lectures*. Number 33. US Govt. Print. Office, 1954.

[61] Inc. Gurobi Optimization. Gurobi Optimizer Reference Manual, 2015.

[62] Ali Haghani, Huijun Hu, and Qiang Tian. An Optimization Model for Real-Time Emergency Vehicle Dispatching and Routing. In *Transportation Research Board 82nd Annual Meeting*, 2003.

[63] Ali Haghani, Qiang Tian, and Huijun Hu. Simulation Model for Real-Time Emergency Vehicle Dispatching and Routing. *Transportation Research Record: Journal of the Transportation Research Board*, (1882):176–183, 2004.

[64] Milos Hauskrecht and Branislav Kveton. Linear Program Approximations for Factored Continuous-State Markov Decision Processes. In *Advances in Neural Information Processing Systems*, 2003.

[65] Kevin Henshall, Peter Schachte, Harald Søndergaard, and Leigh Whiting. Boolean Affine Approximation with Binary Decision Diagrams. In *Proceedings of the Fifteenth Australasian Symposium on Computing: The Australasian Theory-Volume 94*, pages 121–130. Australian Computer Society, Inc., 2009.

[66] J. Hoey, R. St-Aubin, A. Hu, and C. Boutilier. SPUDD : Stochastic Planning using Decision Diagrams. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 279–288, 1999.

[67] Jesse Hostetler, Alan Fern, and Tom Dietterich. State Aggregation in Monte Carlo Tree Search. In *Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI)*, 2014.

[68] Ronald A Howard and James E Matheson. Influence Diagrams. *Decision Analysis*, 2(3):127–143, 2005.

[69] Nathanael Hyafil and Fahiem Bacchus. Utilizing Structured Representations and CSPs in Conformant Probabilistic Planning. In *ECAI*, volume 16, page 1033, 2004.

[70] Keisuke Inakawa, Takehiro Furuta, and Atsuo Suzuki. Effect of Ambulance Station Locations and Number of Ambulances to the Quality of the Emergency Service. In *The Ninth International Symposium on Operations Research and Its Applications (ISORA10)*, pages 340–347, 2010.

[71] Nagisa Ishiura and Shuzo Yajima. A Class of Logic Functions Expressible by Polynomial-Size Binary Decision Diagrams. *VLSI Logic Synthesis and Design*, pages 48–54, 1990.

[72] Murugeswari Issakkimuthu, Alan Fern, Roni Khardon, Prasad Tadepalli, and Shan Xue. Hindsight Optimization for Probabilistic Planning with Factored Actions. In *ICAPS*, 2015.

[73] CJ Jagtenberg, Sandjai Bhulai, and RD van der Mei. An Efficient Heuristic for Real-Time Ambulance Redeployment. *Operations Research for Health Care*, 4:27–35, 2015.

[74] CJ Jagtenberg, Sandjai Bhulai, and RD van der Mei. Dynamic Ambulance Dispatching: Is the Closest-Idle Policy Always Optimal? *Health care management science*, pages 1–15, 2016.

[75] CJ Jagtenberg, PL van den Berg, and RD van der Mei. Benchmarking Online Dispatch Algorithms for Emergency Medical Services. *European Journal of Operational Research*, 2016.

[76] Nicholas K Jong. State Abstraction Discovery from Irrelevant State Variables. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI)*, 2005.

[77] Saket Joshi, Kristian Kersting, and Roni Khardon. Decision-Theoretic Planning with Generalized First-Order Decision Diagrams. *Artificial Intelligence*, 175(18):2198–2222, 2011.

[78] Henry A Kautz, Bart Selman, et al. Planning as Satisfiability. In *ECAI*, volume 92, pages 359–363, 1992.

[79] Ahmet B Keha, Ismael R de Farias, and George L Nemhauser. Models for Representing Piecewise Linear Cost Functions. *Operations Research Letters*, 32(1):44–48, 2004.

[80] Thomas Keller and Patrick Eyerich. PROST : Probabilistic Planning Based on UCT. In *Twenty-Second International Conference on Automated Planning and Scheduling*, 2012.

[81] Thomas Keller and Malte Helmert. Trial-Based Heuristic Tree Search for Finite Horizon MDPs. In *Twenty-Third International Conference on Automated Planning and Scheduling (ICAPS)*, 2013.

[82] Kee-Eung Kim and Thomas Dean. Solving Factored MDPs with Large Action Space using Algebraic Decision Diagrams. In *Proceedings of the 7th Pacific Rim International Conference on Artificial Intelligence: Trends in Artificial Intelligence*, PRICAI, pages 80–89, London, UK, UK, 2002. Springer-Verlag.

[83] Anton J Kleywegt, Alexander Shapiro, and Tito Homem-de Mello. The Sample Average Approximation Method for Stochastic Discrete Optimization. *SIAM Journal on Optimization*, 12(2):479–502, 2002.

[84] Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo Planning. In *Proceedings of the 17th European Conference on Machine Learning (ECML)*, 2006.

[85] Daphne Koller and Ronald Parr. Policy Iteration for Factored MDPs. In *Proceedings of the Sixteenth conference on Uncertainty in Artificial Intelligence*, pages 326–334. Morgan Kaufmann Publishers Inc., 2000.

[86] Andrey Kolobov, Peng Dai, Mausam Mausam, and Daniel S Weld. Reverse Iterative Deepening for Finite-horizon MDPs with Large Branching Factors. In *Twenty-Second International Conference on Automated Planning and Scheduling*, 2012.

[87] Andrey Kolobov, Daniel S Weld, et al. Retrase: integrating paradigms for approximate probabilistic planning. In *Proceedings of the 21st international jont conference on Artifical intelligence*, pages 1746–1753. Morgan Kaufmann Publishers Inc., 2009.

[88] Branislav Kveton and Milos Hauskrecht. Solving Factored MDPs with Exponential-Family Transition Models. In *ICAPS*, pages 114–120, 2006.

[89] Branislav Kveton, Milos Hauskrecht, and Carlos Guestrin. Solving Factored MDPs with Hybrid State and Action Variables. *J. Artif. Intell. Res.(JAIR)*, 27:153–201, 2006.

[90] Roland Leathrum and Carl Niedner. Corvallis Fire Department Response Time Simulation. 2012.

[91] Lihong Li and Michael L Littman. Lazy Approximation for Solving Continuous Finite-Horizon MDPs. In *AAAI*, volume 5, pages 1175–1180, 2005.

[92] Lihong Li, Thomas J Walsh, and Michael L Littman. Towards a Unified Theory of State Abstraction for MDPs. In *Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics (ISAIM)*, 2006.

[93] Cheng Siong Lim, Rosbi Mamat, and Thomas Braunl. Impact of Ambulance Dispatch Policies on Performance of Emergency Medical Services. *IEEE Transactions on Intelligent Transportation Systems*, 12(2):624–632, 2011.

[94] I. Little and S. Thiebaux. Concurrent Probabilistic Planning in the Graphplan Framework. In *Proc. ICAPS*, volume 6, pages 263–272, 2006.

[95] Stephen M Majercik and Michael L Littman. Contingent Planning under Uncertainty via Stochastic Satisfiability. In *AAAI/IAAI*, pages 549–556, 1999.

[96] Christopher R Mansley, Ari Weinstein, and Michael L Littman. Sample-Based Planning for Continuous Action Markov Decision Processes. In *ICAPS*, 2011.

[97] Janusz Marecki, Sven Koenig, and Milind Tambe. A Fast Analytical Algorithm for Solving Markov Decision Processes with Real-Valued Resources. In *IJCAI*, pages 2536–2541, 2007.

[98] M. Mausam and D.S. Weld. Solving Concurrent Markov Decision Processes. In *Proceedings of the 19th national conference on Artifical intelligence*, pages 716–722. AAAI Press, 2004.

[99] Matthew S Maxwell, Shane G Henderson, Huseyin Topaloglu, et al. Tuning Approximate Dynamic Programming Policies for Ambulance Redeployment via Direct Search. *Stochastic Systems*, 3(2):322–361, 2013.

[100] Matthew S Maxwell, Eric Cao Ni, Chaoxu Tong, Shane G Henderson, Huseyin Topaloglu, and Susan R Hunter. A Bound on the Performance of an Optimal Ambulance Redeployment Policy. *Operations Research*, 62(5):1014–1027, 2014.

[101] Matthew S Maxwell, Mateo Restrepo, Shane G Henderson, and Huseyin Topaloglu. Approximate Dynamic Programming for Ambulance Redeployment. *INFORMS Journal on Computing*, 22(2):266–281, 2010.

[102] Richard McCormack and Graham Coates. A Simulation Model to Enable the Optimization of Ambulance Fleet Allocation and Base Station Location for Increased Patient Survival. *European Journal of Operational Research*, 247(1):294–309, 2015.

[103] Peter McCullagh and John A Nelder. *Generalized Linear Models*, volume 37. CRC press, 1989.

[104] H Brendan McMahan, Maxim Likhachev, and Geoffrey J Gordon. Bounded Real-Time Dynamic Programming : RTDP with Monotone Upper Bounds and Performance Guarantees. In *Proceedings of the 22nd International Conference on Machine Learning (ICML)*, 2005.

[105] Luc Mercier and Pascal Van Hentenryck. Performance Analysis of Online Anticipatory Algorithms for Large Multistage Stochastic Integer Programs. In *IJCAI*, pages 1979–1984, 2007.

[106] Luc Mercier and Pascal Van Hentenryck. AMSAA: A Multistep Anticipatory Algorithm for Online Stochastic Combinatorial Optimization. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 173–187. 2008.

[107] Nicolas Meuleau, Emmanuel Benazera, Ronen I Brafman, Eric A Hansen, and Mausam Mausam. A Heuristic Search Approach to Planning with Continuous Resources in Stochastic Domains. *Journal of Artificial Intelligence Research*, 34(1):27, 2009.

[108] S. Milborrow. Derived from mda:mars by T. Hastie and R. Tibshirani. *earth: Multivariate Adaptive Regression Splines*, 2011. R package.

[109] Nitis Mukhopadhyay. *Probability and Statistical Inference*. CRC Press, 2000.

[110] Rémi Munos and Andrew Moore. Variable Resolution Discretization in Optimal Control. *Machine learning*, 49(2-3):291–323, 2002.

[111] Kevin P Murphy. Dynamic Bayesian Networks. *Probabilistic Graphical Models, M. Jordan*, 7, 2002.

[112] Joe Naoum-Sawaya and Samir Elhedhli. A Stochastic Optimization Model for Real-Time Ambulance Redeployment. *Computers & Operations Research*, 40(8):1972–1978, 2013.

[113] Andrew Y Ng and Michael Jordan. Pegasus: A Policy Search Method for Large MDPs and POMDPs. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 406–415. Morgan Kaufmann Publishers Inc., 2000.

[114] Daniel Nikovski and Weihong Zhang. Factored Markov Decision Process Models for Stochastic Unit Commitment. In *Innovative Technologies for an Efficient and Reliable Electricity Supply (CITRES), 2010 IEEE Conference on*, pages 28–35. IEEE, 2010.

[115] J. Pazis and R. Parr. Generalized Value Functions for Large Action Sets. In *Proceedings of the Twenty Eighth International Conference on Machine Learning(ICML)*, pages 1185–1193, 2011.

[116] Jason Pazis and Ronald Parr. PAC Optimal Exploration in Continuous Space Markov Decision Processes. In *AAAI*, 2013.

[117] Jervis Pinto and Alan Fern. Learning Partial Policies to Speedup MDP Tree Search. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2014.

[118] Martin L Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2014.

[119] Martin L Puterman and Moon Chirl Shin. Modified Policy Iteration Algorithms for Discounted Markov Decision Problems. *Management Science*, 1978.

[120] Aswin Raghavan, Saket Joshi, Alan Fern, Prasad Tadepalli, and Roni Khardon. Planning in Factored Action Spaces with Symbolic Dynamic Programming. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

[121] Aswin Raghavan, Roni Khardon, Alan Fern, and Prasad Tadepalli. Symbolic Opportunistic Policy Iteration for Factored-Action MDPs. In *Advances in Neural Information Processing Systems*, pages 2499–2507, 2013.

[122] Aswin Raghavan, Roni Khardon, Prasad Tadepalli, and Alan Fern. Memory-Efficient Symbolic Online Planning for Factored MDPs. 2015.

[123] Aswin Raghavan, Scott Sanner, Roni Khardon, Prasad Tadepalli, and Alan Fern. Hindsight Optimization for Hybrid State and Action MDPs. 2017.

[124] Alberto Reyes, Pablo H Ibarguengoytia, Inés Romero, David Pech, and Mónica Borunda. Open Questions for Building Optimal Operation Policies for Dam Management using Factored Markov Decision Processes. In *2015 AAAI Fall Symposium Series*, 2015.

[125] Nathan Robinson, Charles Gretton, Duc Nghia Pham, and Abdul Sattar. Partial Weighted MaxSAT for Optimal Planning. In *PRICAI 2010: Trends in Artificial Intelligence*, pages 231–243. Springer, 2010.

[126] Stéphane Ross, Geoffrey J Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, volume 1, page 6, 2011.

[127] Sherry Shanshan Ruan, Gheorghe Comanici, Prakash Panangaden, and Doina Precup. Representation Discovery for Mdps using Bisimulation Metrics. 2015.

[128] S. Sanner. Relational Dynamic Influence Diagram Language (rddl): Language Description. *Unpublished ms. Australian National University*, 2011.

[129] S. Sanner, W. Uther, and K.V. Delgado. Approximate Dynamic Programming with Affine ADDs. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1*, pages 1349–1356, 2010.

[130] Scott Sanner, Karina Valdivia Delgado, and Leliane Nunes De Barros. Symbolic Dynamic Programming for Discrete and Continuous State MDPs. *arXiv preprint arXiv:1202.3762*, 2012.

[131] Peter Schachte and Harald Søndergaard. Closure operators for robdds. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 1–16. Springer, 2006.

[132] Verena Schmid. Solving the Dynamic Ambulance Relocation and Dispatching Problem using Approximate Dynamic Programming. *European journal of operational research*, 219(3):611–621, 2012.

[133] Bart Selman and Henry A Kautz. Knowledge compilation using horn approximations. In *AAAI*, pages 904–909. Citeseer, 1991.

[134] Pedro Marinho Sizenando Silva and Luiz Ricardo Pinto. Emergency Medical Systems Analysis by Simulation and Optimization. In *Proceedings of the Winter Simulation Conference*, pages 2422–2432. Winter Simulation Conference, 2010.

[135] Hyeong Seop Sim, Kee-Eung Kim, Jin Hyung Kim, Du-Seong Chang, and Myoung-Wan Koo. Symbolic heuristic search value iteration for factored pomdps. In *AAAI*, pages 1088–1093, 2008.

[136] NC Simpson and PG Hancock. Fifty Years of Operational Research and Emergency Response. *Journal of the Operational Research Society*, pages S126–S139, 2009.

[137] R. St-Aubin, J. Hoey, and C. Boutilier. APRICODD: Approximate Policy Construction using Decision Diagrams. *Advances in Neural Information Processing Systems*, pages 1089–1096, 2001.

[138] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT press, 1998.

[139] Christian Valente, Gautam Mitra, Mustapha Sadki, and Robert Fourer. Extending Algebraic Modelling Languages for Stochastic Programming. *INFORMS Journal on Computing*, 21(1):107–122, 2009.

[140] TC Van Barneveld, S Bhulai, and RD Van der Mei. A Dynamic Ambulance Management Model for Rural Areas. *Health care management science*, pages 1–22, 2015.

[141] TC Van Barneveld, RD Van Der Mei, and S Bhulai. Compliance Tables for an EMS System with Two Types of Medical Response Units. *Computers & Operations Research*, 80:68–81, 2017.

[142] Thije Van Barneveld. The Minimum Expected Penalty Relocation Problem for the Computation of Compliance Tables for Ambulance Vehicles. *INFORMS Journal on Computing*, 28(2):370–384, 2016.

[143] Luis GR Vianna, Leliane N De Barros, and Scott Sanner. Real-Time Symbolic Dynamic Programming for Hybrid MDPs. 2015.

[144] Luis Gustavo Vianna, Scott Sanner, and Leliane Nunes De Barros. Bounded Approximate Symbolic Dynamic Programming for Hybrid MDPs. *arXiv preprint arXiv:1309.6871*, 2013.

[145] Thomas J Walsh, Sergiu Goschin, and Michael L Littman. Integrating Sample-Based Planning and Model-Based Reinforcement Learning. In *Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI)*, 2010.

[146] Chenggang Wang and Roni Khardon. Policy Iteration for Relational MDPs. *arXiv preprint arXiv:1206.5287*, 2012.

[147] Ingo Wegener. *Branching Programs and Binary Decision Diagrams : Theory and Applications*, volume 4. SIAM, 2000.

[148] Daniel S Weld et al. Solving Concurrent Markov Decision Processes. In *Proceedings of the 19th National Conference on Artifical Intelligence*, 2004.

[149] Elizabeth Ty Wilde. Do Emergency Medical System Response Times Matter for Health Outcomes? *Health economics*, 22(7):790–806, 2013.

[150] Shan Xue, Alan Fern, and Daniel Sheldon. Dynamic Resource Allocation for Optimizing Population Diffusion. In *AISTATS*, pages 1033–1041, 2014.

[151] S. Yoon, A. Fern, R. Givan, and S. Kambhampati. Probabilistic Planning via Determinization in Hindsight. In *Proceedings of the 23rd national conference on Artificial intelligence*, volume 2, pages 1010–1016, 2008.

[152] H.L.S. Younes and R.G. Simmons. Policy Generation for Continuous-Time Stochastic Domains with Concurrency. In *ICAPS04*, volume 325, 2004.

[153] Yisong Yue, Lavanya Marla, and Ramayya Krishnan. An Efficient Simulation-Based Approach to Ambulance Fleet Allocation and Dynamic Redeployment. In *AAAI*, 2012.

[154] Zahra Zamani, Scott Sanner, Cheng Fang, et al. Symbolic Dynamic Programming for Continuous State and Action MDPs. In *AAAI*, 2012.

[155] Bruno Zanuttini. Approximating Propositional Knowledge with Affine Formulas. In *Proceedings of the 15th European Conference on Artificial Intelligence*, pages 287–291. IOS Press, 2002.