AN ABSTRACT OF THE DISSERTATION OF

Murugeswari Issakkimuthu for the degree of Doctor of Philosophy in Computer Science presented on September 17, 2021.

Title: Learning and Improving Policies for Probabilistic Planning Problems

Abstract approved: ______________________________

Alan P. Fern

In this work, we study the problem of learning and improving policies for probabilistic planning problems. In the first part, we train neural network policies for probabilistic planning problems modeled as factored Markov decision problems. The objective is to train problem-specific neural networks via supervised learning to imitate the action choices of expert planners. In the second part, we focus on the problem of online policy improvement, where we try to improve on a given base policy via online search. Since search trees for these problems tend to be huge, in practice, action branches need to be pruned, which can affect policy improvement adversely. We formalize this notion by introducing the choice function framework and establish sufficient conditions on actions expanded in search trees for guaranteed policy improvement. In the next part, we draw attention to the fact that theoretical guarantees of policy improvement can fail when the ideal conditions assumed in theory do not hold in practice. We propose benchmark problems, baselines and metrics to assess the empirical performance of online policy improvement algorithms. In the final part, we focus on approximation via state aggregation in MDPs and study the theoretical guarantees of several aggregation schemes.

Learning and Improving Policies for Probabilistic Planning Problems

by

Murugeswari Issakkimuthu

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented September 17, 2021
Commencement June 2022

ACKNOWLEDGEMENTS

I would like to thank my Ph.D. advisor, Prof. Alan Fern, for the great learning experience I have had as his student, and my Ph.D. committee members, Prof. Thomas Dietterich, Prof. Prasad Tadepalli, Prof. Jinsub Kim and Prof. Kyle Niemeyer, for their time and support. I would like to thank Prof. Thomas Dietterich for his review comments on this dissertation.

I would like to acknowledge the sources of support I have received for my studies. I would like to thank Prof. Ben Lee for giving me the opportunity to teach an undergraduate class this summer, and the OSU, EECS and IT staff for the help they have given me on several occasions.

I would like to thank all my teachers, mainly, Prof. N. S. Narayanaswamy and Prof. Deepak Khemani at IIT Madras, and Prof. Anselm Blumer at Tufts University, for their influence on my intellectual growth. I have benefited a lot from the textbooks and online lectures of Prof. Dimitri Bertsekas and Prof. Gilbert Strang at MIT. I would like to thank them for that.

Finally, I would like to express my gratitude to the students in our research group, my friends, and my family members, mainly, my mom, my brother, and my sister.

CONTRIBUTION OF AUTHORS

Prof. Prasad Tadepalli shared his thoughts about the work in meetings and discussions.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF TABLES

Chapter 1: Introduction

Many realistic sequential decision-making problems have huge state and action spaces with highly stochastic state-transition dynamics. The state and action spaces of these problems are usually defined in a factored manner using a set of state and action variables. The state-transition dynamics are specified as Dynamic Bayesian Networks (DBN) [48] over these variables, and the reward function is defined as a function of state and action variables. The framework of factored Markov Decision Processes (MDP) [42] is used to model these problems mathematically, and the language RDDL ([49]) was designed to encode factored MDPs compactly. We focus on RDDL probabilistic planning problems with binary state and action variables in this work.

Probabilistic planning differs from Reinforcement Learning (RL) in that the planning agent has access to a precise model of the environment in the form of a factored MDP. The MDP is usually too large for exact offline solution techniques such as value iteration, policy iteration and linear programming [42]. A typical solution technique for probabilistic planning is online search such as Monte Carlo Tree Search (MCTS) [9] with an environment simulator that can return a sample next state and reward for any state-action pair. The state-of-the-art planner for benchmark RDDL problems, PROST ([28]), performs MCTS with the Upper Confidence bound for Trees (UCT) [31] heuristic for expanding the search tree.

Online search methods make an action choice only for the current environment state as opposed to offline methods that compute a policy for the entire state space. The action selected for the current state is immediately executed to get to the next environment state at which the online search process is repeated. Although very effective in practice, online search can take considerable time to make good action choices for the states encountered in the online search process, which can be a problem in applications that require fast decision making. A good alternative to both online planning and exact offline methods for these applications is high-quality reactive policies compactly encoded for the entire state space that can be applied instantly at any given state. We explore the possibility of using deep learning to train such reactive policies.

In the first part of our work, we train reactive Deep Neural Network (DNN) policies via supervised learning for probabilistic planning problems. We train several problem-dependent network architectures to imitate the action choices of two high-quality expert planners. The key feature of our work is a sparse network architecture with connections mirroring the DBN defining the state-transition dynamics of the problem. This is similar in spirit to Convolutional Neural Network (CNN) architectures taking advantage of the spatial proximity of pixels in images. This work was published in the proceedings of the 28th International Conference on Automated Planning and Scheduling (ICAPS) [23]. It can be found in section 2 of the thesis.

Reactive policies can be applied instantly at any state to make action decisions really quickly. However, in practice, we may have more time to make action decisions than what we need to just apply a reactive policy. The extra time, which might not be enough for a complete online search for optimal actions, can be utilized to search around the base policy to find better actions. We refer to this process of improving on a given base policy via online search as Online Search for Policy Improvement (OSPI). An OSPI procedure performs online search with the objective of computing a policy that is better than or at least as good as the given base policy.

Online search is typically done by constructing a search tree with the state for which an action decision is to be made as the root of the tree. The tree is used to estimate the state-action values of actions at the root and the action with the maximum value is returned for the state. Online search trees tend to be huge, so, in practice, action branches in the tree need to be pruned, which can affect policy improvement adversely. An OSPI procedure can return a policy that is worse than the base policy when actions are pruned in its search tree. We illustrate this with an example and formalize the idea with the choice function framework.

A choice function is a function from nodes in a search tree to action subsets. It returns a subset of applicable actions for a given node in a search tree. An OSPI procedure can be completely specified by a choice function and the leaf evaluation function when the state-transition dynamics in the search tree are exact. Since action pruning in OSPI search trees is entirely defined by the choice function, we can identify properties of choice functions that affect policy improvement.

In the second part of our work, we establish sufficient conditions on choice functions for guaranteed policy improvement in OSPI procedures. We also introduce a parameterized choice function called Limited Discrepancy Choice Function (LDCF), which satisfies the sufficient conditions and covers several existing OSPI procedures as special cases. This work was published in the proceedings of the 34th AAAI Conference on Artificial Intelligence [24]. It can be found in section 3 of the thesis.

There are several existing OSPI procedures that come with theoretical guarantees of policy improvement under ideal conditions. However, these procedures can fail when the ideal theoretical assumptions cannot be satisfied in practice. In the third part our work, we draw attention to the issue of *policy degradation*, which happens when the online policy returned by an OSPI procedure performs worse than the base policy. In order to understand the empirical performance of OSPI procedures, we propose benchmark domains with base policies, baseline OSPI procedures and evaluation metrics that take policy degradation into consideration. This work was published in the Workshop on the International Planning Competition (WIPC) in the 31st International Conference on Planning and Scheduling [21]. It can be found in section 4 of the thesis.

Most realistic probabilistic planning problems lead to factored MDPs with large state and action spaces. Such large MDPs are usually solved approximately as exact offline solution methods are impractical for large state and action spaces. State aggregation in MDPs is an approximate solution technique, where states and even actions of an MDP are grouped together to form a smaller aggregate problem. The aggregate problem is then solved and the solution is extended to the original MDP. The approximate solution thus obtained for the original MDP will usually be sub-optimal with the degree of sub-optimality depending on the precise definition of the aggregate problem.

In the final part of our work, we study existing aggregation methods for approximate solution of MDPs. We consider two basic aggregation frameworks and the associated theoretical results on the sub-optimality of the aggregate solutions. We then relate several aggregation schemes to the two frameworks with the objective of presenting a unified view of the theoretical bounds. This part of our work can be found in section 5 of the thesis.

Training Deep Reactive Policies for Probabilistic Planning Problems

Murugeswari Issakkimuthu, Alan Fern and Prasad Tadepalli

Proceedings of the 28th International Conference on Automated Planning and Scheduling
(ICAPS-2018)
AAAI Press
Palo Alto, CA, USA

Chapter 2: Training Deep Reactive Policies for Probabilistic Planning Problems

2.1 Abstract

State-of-the-art probabilistic planners typically apply look-ahead search and reasoning at each step to make a decision. While this approach can enable high-quality decisions, it can be computationally expensive for problems that require fast decision making. In this paper, we investigate the potential for deep learning to replace search by fast reactive policies. We focus on supervised learning of deep reactive policies for probabilistic planning problems described in RDDL. A key challenge is to explore the large design space of network architectures and training methods, which was critical to prior deep learning successes. We investigate a number of choices in this space and conduct experiments across a set of benchmark problems. Our results show that effective deep reactive policies can be learned for many benchmark problems and that leveraging the planning problem description to define the network structure can be beneficial.

2.2 Introduction

Many real-world planning problems involve large factored state spaces with highly stochastic exogenous and endogenous dynamics. The Relational Dynamic Influence Diagram Language (RDDL) was designed to model such problems by compactly defining large Dynamic Bayesian Networks (DBNs) over state and action variables. Current state-of-the-art planners for RDDL problems are based on online search, where at each step some combination of search and reasoning is used to select an action. For example, there are planners based on sample-based tree search [28, 32, 8], symbolic variants [13, 45, 2], and those that construct and solve integer linear programs at each step [22]. These planners can require non-trivial computation time per step, which can make them inapplicable to problems that require fast decisions.

One approach to support fast decisions is via reactive policies that can be applied online to quickly select actions. Offline Symbolic Dynamic Programming (SDP) has recently been explored for producing such policies for RDDL problems [43, 44]. SDP [18] uses symbolic operations to produce a symbolic policy representation that can be efficiently evaluated at any state. Unfortunately, while there have been significant advances, scalability is still an issue with SDP.

Reactive policies can also be produced via supervised learning or reinforcement learning. Most recently, state-of-the-art results have been achieved in a variety of domains by learning deep neural networks (DNNs) to represent reactive policies. Examples include learning to play Atari games directly from pixel input [38], robotic control (e.g., [34]), and the game of Go [52, 54]. These results motivate the investigation of learning such Deep Reactive Policies (DRPs) for planning problems described in RDDL. We note that the impressive successes of DRPs are not due to the blind application of off-the-shelf tools and DNN architectures. Rather, the successes were enabled by significant expertise and manual exploration of architectures and training methods. The objective of this paper is to present an initial exploration of the DRP design space for RDDL benchmark problems via an extensive empirical investigation covering five domains with some theoretical guarantees about the expressiveness of the architectures.

We describe three classes of architectures that support problem-specific DRPs by leveraging the RDDL problem definition. We train our DRPs to imitate the action choices of more expensive non-reactive planners by supervised learning. We consider two different choices for generating data and two different ways to optimize DRPs based on the data. Our experiments shed light on the following questions. Can we learn DRPs that are competitive with the planners that they are learned from? Can the RDDL problem definition be used to define more effective network architectures? Are there any consistently superior DRP architecture choices across RDDL problems? Are some supervised training signals and loss functions more effective in general than others? We note that this study is focused on learning DRPs for individual planning problems using supervised learning. It is an interesting future direction to consider learning DRPs that generalize across problems within an entire planning domain. However, such a step requires additional architectural considerations, which we believe should be informed by the study of individual problems. We also note that other training mechanisms such as reinforcement learning will be interesting to consider in future work.

2.3 Related Work

There is a long history of work on integrating machine learning and automated planning [36, 63, 26]. Much work focuses on learning control knowledge (heuristics and pruning rules) to speed up a planner and/or improve the plan quality. While these approaches have shown promise, they are not guaranteed to reduce planning times and can even result in net slow down of a planner [37]. An important exception is the prior work on learning reactive policies in the form of relational rule lists for deterministic STRIPS domains [29]. Extensions to the work include using richer rule representations [35, 14], iterative learning algorithms [16], and application to probabilistic STRIPS [62]. While these approaches are promising in many domains, it has been a challenge to demonstrate their robustness across a wide range of domains. One difficulty is that the rule languages, once selected, are inflexible and can not always capture key concepts. This motivates investigating DNNs for planning, since, in principle, they can induce deep features and concepts as needed.

The most closely related prior work is the Factored Policy Gradient (FPG) planner [10], which represents reactive policies using simple neural networks, most commonly a linear network per action for computing action probabilities. The network parameters are tuned using policy-gradient reinforcement learning where each learning episode begins from the starting state of the problem being considered. Promising results were demonstrated for a number of planning domains including probabilistic PDDL (PPDDL) benchmark problems. Interestingly, for most problems there was no perceived benefit to using multi-layer networks over simple linear networks. Our experiments also show that for some RDDL benchmarks linear networks are as good as or better than more complex networks. Most recently, concurrent work [57] is the first to learn deep networks for relational generalization across problems of PPDDL planning domains. PPDDL and RDDL are qualitatively different languages, however, which makes it difficult to apply that approach directly to many RDDL domains.

2.4 RDDL Planning Problems

We assume familiarity with the basic framework of Markov Decision Processes (MDPs). A factored MDP describes the state space by a finite set of binary variables $(x_1, x_2, \ldots, x_n)$

and the action space by a finite set of binary variables $(a_1, a_2, \ldots, a_m)$. In this work, we focus on the case where actions are constrained to have exactly one of the action variables set at any time and view each a_i as a distinct ground action. Most current RDDL benchmarks already have this constraint. The reward function R specifies a mapping from the state and action variables to real-valued rewards. We assume that the transition function T is compactly described as a DBN which specifies the probability distribution over each state variable x_i in the next time step, denoted x'_i, given the values of a subset of the state and action variables $parents(x'_i)$ in the current time step, in particular, $T(s, a, s') = \prod_i Pr(x'_i|parents(x'_i))$. RDDL [49] is a high-level specification language for compactly representing such DBN domains in a relationally-factored form using parameterized state and action variables. Individual problem instances then specify a set of domain objects that instantiate the state and action variables. A policy π is a mapping, possibly stochastic, from the state space to actions. We focus on optimizing the expected finite-horizon total reward of a policy.

2.5 Architectures for Deep Reactive Policies

A Deep Reactive Policy (DRP) is a policy encoded as a deep neural network. DRPs are reactive as they can be quickly evaluated in a single feed-forward pass. Our DRP architectures are organized into $L+1$ layers of nodes. $Z^l = \{z^l_k\}$ denotes layer l, where z^l_k is the k'th node in layer l and its value is denoted by $o(z^l_k)$. The input layer Z^0 contains n nodes, each taking the value of one state variable, i.e., $o(z^0_i) = x_i$. Layers 1 through $L-1$ are hidden layers each containing $C \times n$ nodes, where C parameterizes the number of *channels* which allows for scaling the DRP size with the number of state variables. The output layer $Z^L = \{z^L_0, z^L_1, z^L_2, \ldots, z^L_m\}$ contains $m+1$ nodes, where $z^L_1, \ldots, z^L_m$ correspond to the m RDDL actions and z^L_0 corresponds to the NOOP action. In all the architectures the final hidden layer Z^{L-1} is fully connected to the output layer Z^L.

For **single-channel** networks $(C = 1)$ we have $Z^l = \{z^l_1, z^l_2, ..., z^l_n\}$ for all the hidden layers. Each hidden node z^l_k is connected to a set of nodes (I^l_k) in the previous layer Z^{l-1} via real-valued weights, where $w^l_{i,k}$ is the weight from z^{l-1}_i to z^l_k and b^l_k is the bias parameter to z^l_k. We use $ReLU$ activation functions as the non-linearity for hidden nodes

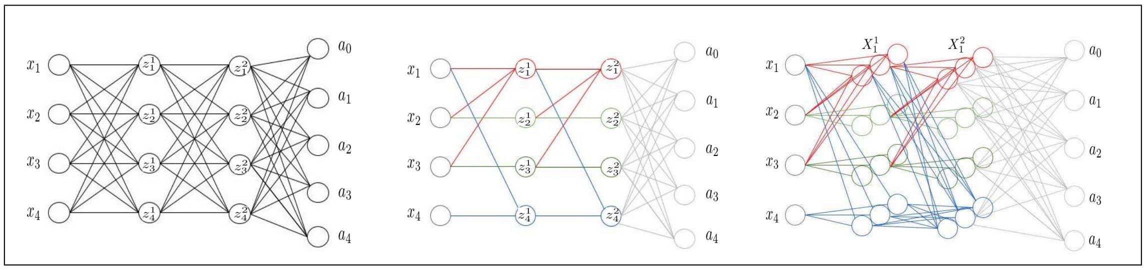


Figure 2.1: Single-channel FC-DRP, Single-channel S-DRP, Multi-channel S-DRP

so that the value of z_k^l is

$$o(z_k^l) = \text{ReLU}\left(\sum_{p \in I_k^l} o(z_p^{l-1}) w_{p,k}^l + b_k^l\right). \tag{2.1}$$

The output layer is a softmax layer computing a probability distribution over the $m+1$ actions

$$o(z_k^L) = \frac{exp(\sum_{p=1}^{|Z^{L-1}|} o(z_p^{L-1}) w_{p,k}^L + b_k^L)}{\sum_{j=0}^{m} exp(\sum_{p=1}^{|Z^{L-1}|} o(z_p^{L-1}) w_{p,j}^L + b_j^L)}. \tag{2.2}$$

For **multi-channel** networks ($C > 1$) $Z^l = \{X_1^l, X_2^l, ..., X_n^l\}$ for all the hidden layers. The set of nodes in Z^l is partitioned into n subsets, where subset X_k^l corresponds to z_k^l in a single-channel network with $|X_k^l| = C$. If the hidden layers of single-channel networks are (column) vectors of size n then those of multi-channel networks are matrices with C such (column) vectors. The k^{th} row of the matrix corresponds to nodes in X_k^l. All nodes in X_k^l receive the same set of input connections from the previous layer though with different weights. If z_p^{l-1} is connected to node z_k^l in a single-channel network then the entire subset of nodes X_p^{l-1} are connected to z_k^l in a multi-channel network.

Fully-Connected DRPs (FC-DRPs). FC-DRPs are the traditional fully-connected networks in which each hidden node at layer l is connected to each hidden node in layer $l-1$, i.e., for all $l > 1$ and k, $I_k^l = \{1, \ldots, |Z^{l-1}|\}$. We use $FC(L, C)$ to denote a FC-DRP architecture with layer and channel parameters L and C.

Sparse DRPs (S-DRPs). The large number of FC-DRP parameters raises the potential for overfitting. CNNs reduce the number of parameters, while remaining expressive by attaching spatial semantics to hidden nodes and only allowing connections to spatially close nodes. In analogy, S-DRPs associate hidden nodes with state variables and only connect nodes whose variables have probabilistic dependencies. $S(L, C)$ denotes an S-DRP with the associated layer and channel parameters. The nodes in hidden layer Z^l are partitioned into n sets $X_1^l, X_2^l, \ldots, X_n^l$, each having C hidden units. We interpret the nodes in X_i^l as being associated with state variable x_i. A hidden node $z_k^l \in X_i^l$ is connected only to nodes in Z^{l-1} that are associated with state variables that x_i depends on in the transition function. In particular, $I_k^l = \bigcup_{j \in parents(x_i')} X_j^{l-1}$, where $parents(x_i')$ is the set of state variables in the current time step that can influence the transition probability of x_i. Thus, the S-DRP connectivity mirrors the DBN local dependency structure across time steps. Figure 2.1 illustrates an FC-DRP and S-DRP with single and multiple channels.

Representation Capacity of S-DRPs. The potential advantage of sparsity is better generalization, while the potential disadvantage is representation capacity. Consider an MDP with two binary state variables x_1 and x_2 with independent transition dynamics. Let policy $\pi(x_1, x_2) = XOR(x_1, x_2)$, which is not linearly representable. The hidden layers for any S-DRP will not be able to compute features that combine x_1 and x_2 and hence the final linear softmax layer will not be able to represent π. In general, when policies involve complex dependencies among state variables that have independent transition dynamics, S-DRPs may be inadequate. We provide an initial result that characterizes a class of policies that is S-DRP representable subject to MDP restrictions and also describe a small S-DRP modification that supports any policy.

The Q-function $Q^{\pi}(s, a)$ of π gives the value of executing action a from state s and then following policy π. We say that a policy π is *Q-representable* if there is a policy π' such that $\pi(s) = \arg\max_a Q^{\pi'}(s, a)$ and that for each state the maximizing Q-value is unique. Examples of Q-representable policies include optimal policies that have unique optimal actions in each state, policies computed by the Rollout algorithm for any base rollout policy π', or any policy from the standard policy iteration sequence. A reward function R is said to be *independently additive* if $R(s) = \sum_i R_i(x_i)$. A transition function is *DBN-representable* if it has the form $T(s, a, s') = \prod_i Pr(x_i' | parents(x_i'))$.

Theorem 1. *For any MDP with independently additive rewards and DBN-representable transition function, if π is Q-representable, then π can be represented as a finite S-DRP.*

Proof. The proof uses the concept of Krylov basis for MDPs [40]. Let P be the transition probability matrix over ground MDP states for π and R be the reward vector over ground states. The Krylov basis function of order t is given by $b^t = P^t R$. The component of vector b^t for state s gives the expected reward at step t if π is followed from s. For a finite K the value function V^π of π can be represented as a linear combination of the basis functions $b^0, \ldots, b^K$ [20]. It follows that for every action $a \in A$, the vector, $Q^\pi(., a)$, consisting of the Q-values of all the states for action a, can also be linearly represented with a basis of the same dimension K.

To relate the Krylov basis to S-DRPs, it is possible to show that for DBN-representable transition functions and independently additive rewards, each Krylov basis function has the form $b^t(s) = \sum_i R_i(x_i) P\left(x_i^t | parents_i^t\right)$, where x_i^t is the value of state variable x_i after t steps from the initial state being conditioned on $parents_i^t$, the set of state variables at the initial time step that influence x_i^t, i.e., the *t-step influencers* of x_i. Thus, the Krylov basis decomposes linearly into a set of functions that depend on the t-step influencers of each x_i. Now consider any S-DRP hidden node $z \in X_i^{L-1}$ in the last layer that is associated with x_i. It is easy to see that the output $o(z)$ is a function of only the input state variables that are L-step influencers of x_i under any policy. Thus, each such z can be viewed as computing a potentially complex non-linear feature of the L-step influencers of x_i. For large enough L and C this allows for the S-DRP to represent the above decomposition of the Krylov basis and applying a softmax layer will then return the actions of π. □

By changing the activation function of the output layer we can represent any policy under mild conditions. In particular, an RBF S-DRP is a DRP where the softmax output layer is replaced by having a radial basis activation function (RBF) for each output node. The only constraint on the RBF is that it is maximized when the affine transformation of its input is zero (e.g., a Gaussian). An RBF S-DRP selects the action in the output layer with the highest node activation. In the following a policy π is said to be Q-distinct if for any state s and any $a \neq \pi(s)$, $Q^\pi(s, \pi(s)) \neq Q^\pi(s, a)$.

Theorem 2. *For any MDP with independently additive rewards and DBN-representable transition function, if π is Q-distinct then π can be represented via a finite RBF S-DRP.*

Proof. (Sketch) The proof is similar to the previous theorem. Since V^π is linearly representable via Krylov basis functions, so is the vector $V^\pi - Q^\pi(., a)$ for any action $a \in A$. Since π is Q-distinct, this expression is zero iff $\pi(s) = a$. This means that applying an RBF to the above difference for each a will identify $\pi(s)$. The rest of the proof follows along the same lines as above. $\square$

When the reward function is not independently additive, it may decompose into factors over groups of state variables. In such cases, we can get a similar result by extending the S-DRPs to include one or more fully connected layers between the last sparse hidden layer and the output layer.

Relational Weight Sharing DRPs (R-DRPs). An R-DRP is constructed by constraining all weights in the S-DRP that are relational matchings to have the same value. Intuitively, (s, t) and (u, v) are relational matchings when the probabilistic dependency between s and t is structurally similar to that from u to v. Sharing is limited to weights between state-fluent nodes in adjacent layers. The bias parameters and the weights of the fully-connected layer at the end are not shared. Two connections are similar if the (start-node, end-node) pairs of the connections are similar. For example, in the blocksworld domain, the pairs (clear(A), on(A, B)) and (clear(C), on(C, D)) are semantically similar. Since nodes in the input and hidden layers of R-DRPs represent state fluents the (start-node, end-node) pairs (z_u^{l-1}, z_v^l) are instantiated first-order predicates.

Consider weights $w_{j,k}^l$ and $w_{u,v}^l$ between (z_j^{l-1}, z_k^l) and (z_u^{l-1}, z_v^l) respectively, where $(z_j^{l-1}, z_k^l) = (q_j(j_1, j_2, ...j_{n_j}), \quad q_k(k_1, k_2, ...k_{n_k}))$, and $(z_u^{l-1}, z_v^l) = (q_u(u_1, u_2, ...u_{n_u}), q_v(v_1, v_2, ...v_{n_v}))$. Let $J = (j_1, j_2, ...j_{n_j}), K = (k_1, k_2, ...k_{n_k}), U = (u_1, u_2, ...u_{n_u})$, and $V = (v_1, v_2, ...v_{n_v})$. Weights $w_{j,k}^l$ and $w_{u,v}^l$ are constrained to be the same if (1) $q_j = q_u$ and $q_k = q_v$ and (2) $J \times K = U \times V$. $|J \times K| = n_j n_k$ and $J \times K$ is defined as the cross-product of ordered tuples J and K giving an ordered tuple of binary values. The 1^{st} n_k entries of $J \times K$ are computed by comparing j_1 with each element of $(k_1, k_2, ..., k_{n_k})$. The 2^{nd} n_k entries of $J \times K$ are computed by comparing j_2 with each element of $(k_1, k_2, ..., k_{n_k})$. The last n_k entries of $J \times K$ are computed by comparing j_{n_j} with each element of $(k_1, k_2, ..., k_{n_k})$. The components of both J and K are strings representing objects in the problem and when j_1 is compared to k_1 the result is 1 if the strings match and 0 otherwise. For example, if $z_j^{l-1} = clear(A)$ and $z_k^l = on(A, B)$ then $J \times K = (1, 0)$ because $A \neq B$ and the second component is 0.

2.6 Supervised Training of DRPs

We use supervised learning to train DRPs for individual RDDL problems. This involves generating training data and optimizing the parameters of a chosen DRP architecture.

Training Data Generation. Following prior work (e.g., [29, 35, 62]) we generate training data using *imitation learning*, which aims to learn a policy that imitates the actions of an expert. In our case, the expert is a non-reactive online planner that can select an action at any state. More precisely, given a planning problem with initial state s_0 and horizon H, we use the planner to generate multiple trajectories, each one starting in s_0 and then following a sequence of actions selected by the planner until the horizon. Each of the stochastic trajectories gives a sequence of state-action pairs $(s_0, a_0), (s_1, a_1), \ldots, (s_{H-1}, a_{H-1})$, which can be combined to create a standard supervised training set. A disadvantage of learning from just state-action pairs is that the learning algorithm is unable to make informed trade-offs when perfect accuracy is not possible. To address this, we can augment the training examples with Q-value estimates for each action when available from the planner. Here the Q-value of a state action pair $Q(s, a)$ is the expected finite horizon reward of starting in state s, taking action a, and then acting optimally thereafter. This idea of leveraging Q-values for supervised policy learning has been shown to be effective in prior work, e.g., [16].

Expert Planners. We consider imitation learning from two RDDL planners. The first is the state-of-the-art planner, Prost [28] (IPPC-2011), which is based on Monte-Carlo Tree Search with various heuristics and pruning mechanisms. Prost does not generate Q-value estimates for all actions in a state due to pruning mechanisms. Thus, when using Prost, the training data only contains state-action pairs. The second planner is Rollout, which performs policy rollout [56] using a random base policy. Given a state s, Rollout produces a very rough estimate of $Q(s, a)$ for each action a as follows. Simulate N trajectories that each start at s, then select action a followed by random actions until a fixed horizon. $Q(s, a)$ is estimated to be the average cumulative reward across the trajectories, and the Rollout planner returns the action that maximizes $Q(s, a)$. Since Rollout produces Q-value estimates for all actions, we include those values in the training data. Rollout can be viewed as computing a policy that is equivalent to performing one step of policy iteration starting from a random policy. In practice, Rollout is often

Table 2.1: Sysadmin, Game of Life and Skill Teaching Results

Sysadmin														
	Planners		**TRN(Rollout, 0/1)**				**TRN(Rollout, Q)**				**TRN(Prost, 0/1)**			
Problem #	Prost	Rollout	π_{lin}	π^*	π_L	π_A	π_{lin}	π^*	π_L	π_A	π_{lin}	π^*	π_L	π_A
1	339	332	342	346	344	344	341	346	341	341	342	347	344	341
				R310	F110	F110		R110	F310	Lin		S310	F310	F15
2	301	290	302	315	309	313	313	319	311	313	311	321	321	316
				S510	S110	S15		S110	S310	R110		S310	S310	F110
3	553	523	562	575	559	559	559	576	557	562	570	570	561	554
				F110	S110	S15		S110	S310	S55		Linear	F110	S35
4	489	463	502	504	492	504	495	510	501	501	496	513	490	486
				F110	S110	F110		F110	S310	S35		F11	S110	F15
5	573	588	625	645	631	618	625	649	628	634	620	650	637	638
				S15	S110	R110		R310	S310	S110		S15	S110	F110
6	527	532	583	598	590	598	583	597	597	595	576	601	578	573
				S15	S110	S15		S310	S310	R110		S310	S110	F15
7	618	658	724	734	733	714	727	737	730	737	709	730	723	711
				S310	S15	R35		S310	S110	S310		R35	S15	S35
8	498	522	589	591	589	569	596	600	584	579	583	591	591	583
				S15	Linear	R11		F11	S110	S15		S15	S15	Linear
9	728	811	872	889	875	872	884	893	883	877	832	849	833	849
				R35	S15	Linear		R310	S110	R35		F11	S15	F11
10	546	580	643	645	641	643	639	655	643	624	608	624	608	624
				F11	S15	Linear		F11	S110	R15		S11	Linear	S11
%Δ Prost	0	1.62	9.82	11.72	10.21	9.81	10.22	12.55	10.43	10.21	8.32	11.18	9.23	8.82
%Δ Rollout	-1.32	0	8.00	9.88	8.37	8.03	8.38	10.69	8.59	8.40	6.63	9.44	7.50	7.06
Game of Life														
1	210	188	77	196	196	196	70	202	197	199	49	191	191	188
				F310	F310	F310		F510	S510	S310		S310	S310	F310
2	130	122	96	125	125	125	98	135	135	121	85	129	126	129
				F510	F510	F510		F510	F510	F55		F510	F310	F510
3	150	134	128	146	146	141	121	148	148	148	119	149	148	149
				F510	S510	F310		R510	S510	S510		F510	S310	F510
4	347	347	225	331	331	331	227	339	338	338	206	321	304	321
				S310	S310	S310		S55	S510	S510		S310	S35	S310
5	309	295	240	285	285	280	234	304	304	304	229	299	287	299
				S35	S35	S510		S510	S510	S510		S510	S310	S510
6	283	266	253	268	267	263	252	277	276	274	245	277	275	277
				S510	S310	S55		F35	S35	S510		S310	S35	S310
7	486	500	330	455	449	447	308	481	481	481	280	435	421	435
				S510	S35	S310		S510	S510	S510		S510	S35	S510
8	435	450	330	431	431	431	337	449	446	446	313	408	408	406
				S55	S55	S55		S55	S510	S510		S35	S35	S510
9	410	412	340	416	414	416	344	429	419	419	335	402	399	402
				S310	S35	S55		S510	S55	S55		S55	S35	S55
10	575	602	263	488	486	488	252	531	531	531	280	513	483	476
				S510	S310	S510		S510	S510	S510		S510	S35	S310
%Δ Prost	0	-2.58	-29.91	-5.06	-5.33	-5.93	-31.18	-0.72	-1.36	-2.37	-35.31	-5.20	-7.43	-6.02
%Δ Rollout	2.98	0	-27.74	-2.20	-2.45	-3.12	-29.09	2.23	1.56	0.49	-33.43	-2.27	-4.51	-3.07
Skill Teaching														
1	67	65	66	67	67	66	64	67	66	64	67	68	65	65
				F11	S510	F31		F15	S53	S55		R110	F55	F55
2	80	76	76	78	76	78	76	78	75	77	78	80	77	77
				R35	R510	R35		F110	S53	S31		F31	F310	F310
3	74	85	83	94	85	82	80	98	92	78	87	106	89	89
				S15	R55	F31		R31	S55	F11		F15	S510	S510
4	101	84	62	104	101	82	56	110	91	89	114	114	93	91
				R51	R55	R15		R31	R510	S35		F110	F55	R35
5	10	-10	-28	-4	-23	-39	-14	-4	-42	-19	17	36	-1	-1
				R51	R53	S310		R55	R310	R53		R31	F35	F35
6	-11	-11	31	33	-1	-6	17	24	-25	9	5	21	-4	-4
				S51	R310	F110		F51	R510	S110		F310	F35	F35
7	-48	-83	-68	-46	-89	-49	-59	-40	-60	-51	-44	-23	-62	-62
				R53	R55	S310		R53	R510	S510		F31	R310	R310
8	-141	-210	-191	-142	-163	-212	-155	-139	-156	-156	-154	-109	-144	-134
				F310	R55	F110		F110	R55	R55		F510	S310	S55
9	-145	-155	-160	-138	-161	-162	-146	-122	-155	-155	-167	-122	-156	-172
				F510	R35	R55		S35	R35	R35		S15	F15	F510
10	-214	-212	-216	-194	-226	-240	-247	-188	-268	-279	-228	-178	-214	-194
				F310	R53	R31		F11	R510	F15		R15	F110	F53
%Δ Prost	0	-34.46	-13.40	28.75	-35.04	-54.41	-9.29	24.70	-72.14	-18.76	21.96	71.06	-8.41	-8.12
%Δ Rollout	29.01	0	19.23	57.26	-0.21	-20.93	20.93	52.84	-38.46	12.03	50.17	94.30	22.03	22.11

Table 2.2: Tamarisk Results

	Planners		**TRN(Rollout, 0/1)**				**TRN(Rollout, Q)**				**TRN(Prost, 0/1)**			
Problem #	Prost	Rollout	π_{lin}	π^*	π_L	π_A	π_{lin}	π^*	π_L	π_A	π_{lin}	π^*	π_L	π_A
1	-137	-160	-177	-124	-142	-142	-173	-127	-145	-145	-162	-123	-142	-142
				F35	S310	S310		S55	S35	S35		F15	S310	S310
2	-469	-524	-587	-469	-485	-475	-571	-473	-502	-484	-532	-427	-486	-472
				R15	S35	R35		R15	S35	S310		F31	S510	S310
3	-210	-243	-244	-198	-211	-211	-274	-207	-207	-207	-256	-186	-209	-200
				S15	S310	S310		S310	S310	S310		R15	S310	S35
4	-744	-783	-786	-650	-705	-705	-805	-669	-719	-719	-782	-694	-701	-694
				R110	S35	S35		S15	R310	R310		R310	S35	R310
5	-568	-646	-671	-560	-615	-560	-640	-547	-588	-588	-645	-526	-558	-579
				S35	S310	S35		S510	R310	R310		R510	S310	S35
6	-1005	-977	-940	-834	-883	-886	-969	-866	-882	-891	-1100	-893	-938	-938
				S110	S310	R310		S55	S310	R310		S55	S35	S35
7	-862	-829	-834	-662	-669	-669	-809	-677	-687	-677	-875	-679	-709	-709
				S510	S310	S310		S53	S310	S53		S110	S310	S310
8	-1380	-1229	-1210	-1087	-1139	-1165	-1203	-1104	-1131	-1144	-1361	-1228	-1243	-1243
				R510	S35	S310		R110	S55	S35		F51	S35	S35
9	-1010	-827	-803	-686	-797	-736	-867	-681	-735	-681	-961	-752	-818	-821
				F11	S310	S510		R510	S510	R510		R310	S310	S53
10	-1548	-1228	-1259	-1064	-1124	-1095	-1254	-1057	-1201	-1201	-1528	-1375	-1394	-1595
				S110	S35	S310		F11	F510	F510		S53	S35	F510
%Δ Prost	0	-0.72	-3.27	15.39	8.99	10.73	-4.39	14.14	8.87	9.71	-7.56	12.48	6.38	5.52
%Δ Rollout	-1.16	0	-2.15	15.87	9.52	11.28	-3.33	14.69	9.40	10.30	-7.70	11.91	6.15	4.90

surprisingly effective and it is often competitive or better than Prost, especially for larger planning problems.

Parameter Optimization. For each problem we use both Prost and Rollout to generate a training data set of size 10,000 state-action pairs for three domains and up to 32,000 for the other two domains (Sysadmin and Game-of-life) depending on the problem size. The data was generated by producing trajectories with horizon $H = 40$. Given one such dataset, we optimize the parameters of a DRP by defining a loss function over the training data and applying stochastic gradient descent. In this work, for all problems and networks we use the Adam optimizer built into the Tensorflow framework with a batch size of 40 and initial learning rate of 10^{-5}. We train for 2000 iterations and compute the accuracy on a validation set every 500 iterations and stop if there is no improvement in two successive stages. Training times vary significantly for different problems and architectures, which can be improved with additional hardware and further optimizations.

Training with 0/1 Loss. Our first loss is defined over just state-action pairs. Given a state s, a DRP produces a probability distribution over actions, $P(a|s)$, which we will denote by the vector $\hat{P}(s)$. Given a training state-action pair (s, a), let $t(s)$ denote the 0/1 target probability distribution over actions that assigns probability 1 to action

Table 2.3: Wildfire Results

	Planners		**TRN(Rollout, 0/1)**				**TRN(Rollout, Q)**				**TRN(Prost, 0/1)**			
Problem #	Prost	Rollout	π_{lin}	π^*	π_L	π_A	π_{lin}	π^*	π_L	π_A	π_{lin}	π^*	π_L	π_A
1	-275	-439	-481	-256 S110	-603 S35	-603 S35	-522	-368 R510	-368 F15	-368 F15	-950	-159 F35	-238 R15	-208 S510
2	-8856	-8913	-9466	-8621 S55	-8900 F15	-8783 S15	-8989	-8674 R110	-9078 S15	-9078 S15	-8807	-8428 F11	-9034 F15	-9034 F15
3	-1899	-1547	-1354	-1131 R510	-1131 R510	-1373 F11	-2037	-976 S15	-1285 F11	-1517 R310	-1355	-802 F15	-1747 S35	-1490 R310
4	-8756	-8986	-8572	-7808 R110	-8136 R510	-8459 S55	-8840	-7757 R310	-8040 S35	-8040 S35	-8888	-7693 S35	-7693 S35	-9121 F31
5	-3220	-585	-1331	-467 R15	-716 S11	-1100 R510	-800	-497 F11	-497 F11	-723 S35	-1552	-1552 Linear	-2959 S110	-2517 F510
6	-15878	-7079	-7370	-6548 F15	-7465 F11	-6820 S35	-7132	-6480 R15	-7221 R55	-7221 R55	-15313	-10948 F15	-14056 S110	-11975 F510
7	-7731	-6169	-5483	-4885 S15	-5169 S11	-6479 F510	-5452	-5178 S310	-5648 S15	-5882 F510	-7327	-6270 R31	-9259 F510	-9259 F510
8	-13673	-10192	-9975	-9389 S11	-9389 S11	-10840 S53	-9411	-9305 S110	-9529 F11	-9828 S15	-13053	-11235 R11	-16661 F510	-16661 F510
9	-16129	-5551	-4941	-4152 R310	-6662 F510	-6662 F510	-4317	-4310 R51	-5911 F510	-5911 F510	-17962	-17036 F35	-17556 F510	-17556 F510
10	-25459	-12049	-11238	-9763 R35	-12030 F510	-12030 F510	-10586	-9683 F11	-11113 F510	-11113 F510	-31343	-29047 F53	-30061 F510	-30061 F510
%Δ Prost	0	25.68	24.26	40.90	23.44	18.39	22.30	37.24	32.04	29.59	-18.67	21.66	-1.74	1.74
%Δ Rollout	-91.75	0	-9.91	18.54	-2.45	-13.43	-3.35	15.99	7.12	1.09	-81.34	-44.79	-93.72	-82.48

a. We measure the 0/1 cross entropy loss of a prediction $\hat{P}(s)$ as the cross-entropy $H(\hat{P}(s), t(s))$ between $\hat{P}(s)$ and $t(s)$, where for probability vectors P and Q, $H(P,Q) = -\sum_i P_i \log(Q_i)$. $H(P,Q)$ is minimized when $P = Q$, and hence the 0/1 loss encourages $\hat{P}(s)$ to increase the probability of the action.

Training with Q-Loss. When Q-values are available in the training data, we incorporate them by defining a Q-Loss function that prefers predictions $\hat{P}(s)$ that assign higher probabilities to actions with higher Q-values. In particular, we use the Q-values for a state s to define a Boltzmann probability distribution over actions $P(a \mid s) = \frac{\exp(Q(s,a))}{\sum_{a'} \exp(Q(s,a'))}$ with temperature equal to one. Here P assigns higher probability to actions with higher Q-values. Our Q-loss function for a training example is then simply $H\left(\hat{P}(s), P(\cdot|s)\right)$, which is minimized when the predicted probabilities match the Boltzmann probabilities.

Doing Better than the Expert. In our experiments, we will sometimes see the learned DRPs outperforming the expert planners. The exact reasons for this is not fully clear. However, results from imitation-learning theory offer a potential explanation. First, it is important to note that Prost and Rollout are both stochastic planners due to running Monte-Carlo simulations. One way to model the stochasticity is by starting with a deterministic policy π^* that captures the typical action choices of the planner and then creating a stochastic policy $\hat{\pi}$ that follows π^* with $1 - \epsilon$ probability and uses a randomized action choice with ϵ probability. It has been shown by Ross et al. [47] that the finite-horizon reward of $\hat{\pi}$ can be worse than π^* by as much as ϵH^2, where H is the horizon. Thus, even if Prost and Rollout typically select actions according to a high-quality π^*, their actual performance can be substantially worse. We can now think of the training data as being generated by π^*, but corrupted with some amount of noise. If our learning procedure is robust to the noise, then it is possible for the learned DRP to provide a better approximation of π^* than the planners. In particular, if the learned approximation has an error rate of $\epsilon' < \epsilon$, then the learned policy has the potential to achieve a performance closer to π^* than the planner.

2.7 Experiments

Benchmark Problems and Architectures. We selected five RDDL benchmark domains: Sysadmin, Game-of-Life, Skill Teaching, Tamarisk, and Wildfire. Each domain comes with a standard set of ten problems ranging from quite small to quite large. While computational constraints prevented including additional domains, there are some benchmark domains that are not a good match for DRPs. For example, the Navigation domain contains state variables that only provide the robot location. To be successful a planner needs to reason about the probabilistic navigation grid to eventually find a deterministic optimal path. In such domains, there is no room to benefit from the generalization ability of a DNNs. All the selected domains appear to offer non-trivial opportunities to learn policies that generalize across states.

We trained FC-DRPs, S-DRPs, and R-DRPs for all combinations of $L = 1, 3, 5$ and $C = 1, 5, 10$ along with a linear policy (no hidden layers). Each architecture was trained using three strategies: Rollout as the planner with 0/1 loss, Rollout with Q-loss, and Prost with 0/1 loss. The strategies are denoted by *TRN(Rollout,0/1)*, *TRN(Rollout, Q)*, and *TRN(Prost, 0/1)* respectively. In total this resulted in training 27×3 networks for each of the 50 problems.

Description of Results Tables. Tables 2.1-2.3 contain our main set of experimental results for each domain. Throughout our experiments, the expected total reward of a policy or planner is estimated using a horizon of 40 averaged over 100 simulations. In each table the top 10 rows give results for individual problems, where larger problem numbers tend to correspond to larger problems. The second and third columns give the average reward of Prost and Rollout. The next three blocks of columns correspond to one of the three training methods. For each training method, the first column gives the average reward for the trained linear policy π_{lin}. The next column, labeled π^*, gives the maximum reward achieved over all architectures trained for the problem (all combinations of F(L,C), S(L,C), and R(L,C)). Below this maximum reward is the name of the architecture that achieved the maximum reward, e.g., S110 is an S-DRP with $L = 1$ and $C = 10$. This maximum reward is what we would achieve in practice if we performed DRP model selection via simulation of the learned policies, which will be practical in some settings.

The final two columns in each block are included to assess our ability to perform model selection using validation data, rather than simulations as for π^*. In particular, for each problem in addition to the training set we generated a set of validation data containing 2000 state-action pairs from the appropriate planner. Given a learned DRP, we can evaluate the loss it achieves on the validation data (either 0/1 or Q loss as appropriate) and the accuracy of selecting the actions in the validation set. For each problem, the column π_L (π_A) gives the average reward and name of the architecture that minimized (maximized) the validation loss (accuracy). For large RDDL benchmarks and a moderate number of DRP architectures, it will often be much cheaper to select models using the validation set measures compared to using simulation to estimate expected reward. Thus, the π_L and π_A columns are included to help evaluate how effective this cheaper form of model selection might be.

Finally, the last two rows of each table aggregate results across problems. The row labeled $\%\Delta Prost$ ($\%\Delta Rollout$) gives the average percentage improvement over Prost (Rollout) across problems for each column. For example, in Sysadmin, the Rollout planner achieves a negligible average improvement over Prost of 1.62%. Negative values indicate an average decrease in performance.

Comparison to Expert Planners. First we consider the performance of the simple linear policy. We note that for Sysadmin and Wildfire that on averge π_{lin} is able to achieve a non-trivial average performance improvement over both Prost and Rollout for all of the training regimes, with the exception of TRN(Prost, 0/1) for Wildfire. This

Table 2.4: FC-DRP vs S-DRP vs R-DRP Results

		TRN(Rollout, 0/1)				TRN(Rollout, Q)				TRN(Prost, 0/1)			
Domain	%Δ	π^*	π_F	π_S	π_R	π^*	π_F	π_S	π_R	π^*	π_F	π_S	π_R
Sysadmin	Prost	11.72	10.85	11.11	10.56	12.55	11.74	11.96	11.83	11.18	10.30	10.80	9.91
	Rollout	9.88	9.04	9.27	8.73	10.69	9.89	10.13	9.96	9.44	8.59	9.07	8.18
Game of Life	Prost	-5.06	-8.01	-5.49	-12.32	-0.72	-4.48	-1.64	-6.79	-5.20	-10.01	-5.45	-13.17
	Rollout	-2.20	-5.07	-2.66	-9.68	2.23	-1.44	1.25	-3.90	-2.27	-6.96	-2.53	-10.42
Skill Teaching	Prost	28.75	-10.93	18.95	6.87	24.70	17.90	-1.89	18.55	71.06	62.72	61.09	53.67
	Rollout	57.26	19.89	48.39	36.19	52.84	46.27	28.15	47.23	94.30	86.50	85.68	79.50
Tamarisk	Prost	15.39	12.19	13.95	13.06	14.14	11.98	13.56	13.26	12.48	10.88	10.42	11.90
	Rollout	15.87	12.74	14.38	13.67	14.69	12.69	14.00	13.86	11.91	10.28	10.05	11.33
Wildfire	Prost	40.90	36.26	39.41	39.76	37.24	35.08	35.42	35.63	21.66	17.63	11.98	12.12
	Rollout	18.54	11.82	15.79	17.15	15.99	12.97	11.44	13.16	-44.79	-60.91	-75.68	-69.14

indicates that it is possible to represent good policies in these two domains using simple functions. This is also an example of where a learned policy outperforms the expert that it was learned from, for which, we presented a potential explanation. For other domains, π_{lin} performs worse than the experts, which indicates that good policies in these domains require more complex representations or that a good linear policy was not learnable using this training data. Now consider the performance of π^*, which is the best we would hope to do when using simulation for model selection. For all domains, with the exception of Game-of-Life, π^* had better average performance than both expert planners in all three training regimes. One exception was a decrease in performance relative to Rollout in Wildfire for TRN(Prost, 0/1). This decrease is understandable since the performance of Prost in this domain is quite poor compared to Rollout (average performance reduction of -91.75%), which means learning from Prost is unlikely to yield good performance. In Game-of-Life, π^* is on average worse than both expert planners by a small percentage in all three training regimes. We note that the relatively small drop in performance compared to the planners comes with a dramatic improvement in decision making time. Finally, we note that for all benchmarks and all training regimes, π^* is able to significantly outperform the linear policy on average, except in Sysadmin, where π^* is better by only a small margin. This shows that there is indeed benefit to considering deeper non-linear architectures for these RDDL benchmarks.

Comparison of Model Selection Strategies. Here we consider the impact of using the validation set loss π_L and validation accuracy π_A for model selection instead of using simulations as done for π^*. We first observe that for Sysadmin, Game-of-Life, and Tamarisk, the drop in performance of π_L and π_A compared to π^* for all training regimes is relatively small in terms of average percent improvement over the planners. For Wildfire, the performance drop is more significant when learning based on 0/1 loss, especially for TRN(Prost, 0/1), which may again be due to the low quality of Prost's data in this domain. In Skill Teaching, the drop in average performance of both π_L and π_A is substantial in all training regimes. This indicates a poor match between the validation loss and actual reward accumulated during planning. The reasons for this are currently unclear. We also observe that except for Skill Teaching, in all cases where π^* showed an average positive improvement over an expert planner, π_L and π_A were also able to achieve an improvement. Thus, in most cases if we were satisfied with the performance of the expert planner, we would also be satisfied with the much faster DRPs selected by π_L and π_A. Finally, there does not appear to be a clear winner between π_L and π_A, nor does there appear to be a training regime where model selection based on validation data performs best.

Comparison of Training Methods. For Sysadmin, Game-of-Life, and Tamarisk we see that Prost and Rollout achieve similar average performance across problems. In these cases, we see that the performance of π^* is also similar across the three training regimes. In Skill Teaching, Rollout is significantly worse on average than Prost (by -34.46%) and we see that π^* trained with TRN(Rollout, 0/1) and TRN(Rollout, Q) is significantly worse on average than with TRN(Prost, 0/1). This agrees with the intuition that learning from a lower quality planner should result in a worse learned policy. For Wildfire, the situation is reversed and we see that training from Prost data is significantly worse than training from rollout with TRN(Rollout, 0/1) and TRN(rollout, Q). Overall these results indicate that for these experiments, the quality of the planner used to generate data is the dominating factor in training, compared to using 0/1 or Q-based loss. This is in contrast to prior studies [16, 3], where Q-based loss improved performance. This suggests investigating improved ways to incorporate Q-values into loss functions.

Comparison of Architectures. From the tables, we can observe for each problem and training regime, which architecture was selected by π^*. Overall, from this data we do not see a consistent trend that would favor particular architectural properties. In particular, we see FC-DRPs, S-DRPs, and R-DRPs all appearing with reasonable frequencies. We do see that in Game-of-Life, which is perhaps the most complex policy to learn based on the difficulty of competing with the planners, we see that, for large problems, S-DRPs with larger values of L and C tend to be chosen. It is also difficult to spot an overall trend in terms of L or C. We did find that sparse architectures suffer much more than FC-DRPs when $C = 1$, which requires further investigation. We note that these results are at best suggestive, since the tables do not indicate how close other architectures were to the performance of π^*.

Table 2.4 summarizes the performances of the best FC-DRP, S-DRP, and R-DRP across the domains. Each row gives the averaged % improvement over Prost or Rollout for each domain. The first column in each training regime, copies results for π^* from the previous table and represents the best performance over all architectures. The next three columns record the average improvement of the best architecture restricted to FC-DRPs (π_F), S-DRPs (π_S), and R-DRPs (π_R). Again we see that there is not a consistently best single top performing class. We do see that at least one of the sparsely connected DRPs, π_S and π_R, always outperform the fully connected architectures (π_F), with the exception of Wildfire and Tamarisk for TRN(Prost, 0/1). This suggests that the sparse architectures can leverage the RDDL definition to realize a benefit. We have also seen that frequently the sparse architectures are able to achieve similar results to FC-DRPs using many fewer parameters. It is also encouraging to see that the weight sharing approach of π_R is usually competitive on average even though it uses dramatically fewer parameters. This suggests the potential effectiveness of relational generalization across planning problems in a domain. Finally, we see that π^* sometimes significantly outperforms the others. This indicates that within a single problem domain, the best architecture class differs across the problems.

The Choice Function Framework for Online Policy Improvement

Murugeswari Issakkimuthu, Alan Fern and Prasad Tadepalli

Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI-2020)
AAAI Press
Palo Alto, CA, USA

Chapter 3: The Choice Function Framework for Online Policy Improvement

3.1 Abstract

There are notable examples of online search improving over hand-coded or learned policies (e.g., AlphaZero) for sequential decision making. It is not clear, however, whether policy improvement is guaranteed for many of these approaches, even when given a perfect leaf evaluation function and transition model. Indeed, simple counterexamples show that seemingly reasonable online search procedures can hurt performance compared to the original policy. To address this issue, we introduce the choice function framework for analyzing online search procedures for policy improvement. A choice function specifies the actions to be considered at every node of a search tree, with all other actions being pruned. Our main contribution is to give sufficient conditions for stationary and non-stationary choice functions to guarantee that the value achieved by online search is no worse than the original policy. In addition, we describe a general parametric class of choice functions that satisfy those conditions and present an illustrative use case of the empirical utility of the framework.

3.2 Introduction

For many applications of sequential decision making, it is possible to learn or hand-code a reactive policy for online operation, e.g., [19, 30, 50]. While such policies are computationally cheap to apply, they will generally be sub-optimal in some or many states. This motivates using additional computation during online operation to improve upon such base policies. The focus of this paper is on approaches that use online lookahead search for this purpose, which we refer to as Online Search for Policy Improvement (OSPI).

At each state encountered during online operation, OSPI approaches use an environment simulator or model to construct a search tree that includes the base-policy action choices along with a subset of other action choices. The action values at the root can then be used to select an action. Well-known examples of OSPI include the policy-rollout algorithm [56], which was first shown to improve Backgammon policies, and AlphaZero [54], which improves over an underlying greedy policy via Monte-Carlo Tree Search.

Ideally, due to the additional online computation, we would like an OSPI procedure to yield improved performance compared to the base policy. Perhaps more importantly, we would at least like to guarantee that an OSPI procedure is “safe” in the sense that it does not perform worse than just using the base policy. For example, the policy rollout algorithm is guaranteed to be safe in this sense. However, as we show in Section 3.4, many OSPI procedures are not safe, even when 1) using a perfect transition model, 2) using the exact policy value function for leaf evaluation, and 3) the base policy action is expanded at each tree node.

Our primary goal is to derive safety conditions for OSPI. For this purpose, we introduce the choice-function framework for analyzing OSPI procedures. The key idea is to notice that OSPI procedures primarily differ in their choice of which actions other than the base policy action to expand at each tree node. Thus, each procedure can be characterized by a choice function, which specifies the actions to consider at each node of the search tree. Thus, we can characterize properties of an OSPI procedure, such as safety, via properties of the corresponding choice function.

Our main contribution is to give sufficient conditions on choice functions that guarantee safety. This is done for both stationary and non-stationary choice functions. In addition, we describe a parametric class of safe choice functions, that captures a number of existing approaches. This allows for hyperparameter search over a safe space of OSPI procedures in order to optimize online performance. Using this class we provide illustrative empirical results that demonstrate the practical potential of the framework.

3.3 Related Work

An early approach for OSPI is the policy-rollout algorithm [6, 56], which has been shown to significantly improve policies in a variety of applications, e.g., Backgammon [56], combinatorial optimization [7] and stochastic scheduling [5]. Nested rollout [61, 11] allows for leveraging additional computation time to further improve a policy by approximating multiple steps of policy iteration. Policy Switching [12] allows rolling out multiple policies instead of just one and improves over all the base policies.

Monte-Carlo Tree Search (MCTS) has commonly used policies as a form of knowledge to guide and prune the search, often as part of the rollout policy applied at the leaves [9]. Recent, high-profile examples include AlphaGo and AlphaZero [52, 54], which combine a learned base policy and value function to guide MCTS. One view of the search approach of AlphaZero is as OSPI, where the search aims to improve over the learned greedy base policy. Indeed, the basis for learning is to use search to generate training data from a (hopefully) improved policy. A related approach [41] uses a learned policy to prune actions from consideration at each tree node that are not highly ranked by the policy. Another example of combining MCTS with policies [39] allows the base policy to be treated as a temporally extended action at each node in the search tree.

The idea of searching around a base policy has also been considered in the area of deterministic heuristic search. Limited Discrepancy Search (LDS) [17] uses a heuristic to define a greedy policy for guiding search. LDS generates all paths in the search tree that disagree with at most K choices of the base policy and returns the best solution uncovered by the search. LDS has been used effectively in a variety of search problems ranging from standard benchmarks to structured prediction [15] and non-deterministic AND/OR search graphs [33].

3.4 Problem Setup

We formulate sequential decision making in the formalism of Markov Decision Processes (MDPs). An MDP is a 4-tuple $\langle S, A, P, R\rangle$, where S is a finite set of states, A is a finite set of actions, $P : S \times A \times S \to [0, 1]$ is the state-transition function and $R : S \times A \to \mathbb{R}$ is the reward function. $P_{ss'}(a)$ denotes the probability of reaching state s' from state s taking action a and $R(s, a)$ denotes the immediate reward for taking action a in state s. We focus on the discounted infinite-horizon setting with discount factor γ. A policy, $\pi : S \to A$, is a mapping from states to actions with value function V^π given by

$$V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} P_{ss'}(\pi(s)) \cdot V^\pi(s')$$

for all $s \in S$. Offline computation of an optimal policy can be computationally expensive and impractical for applications with large state spaces. In such cases, online search is a practical alternative to offline solutions.

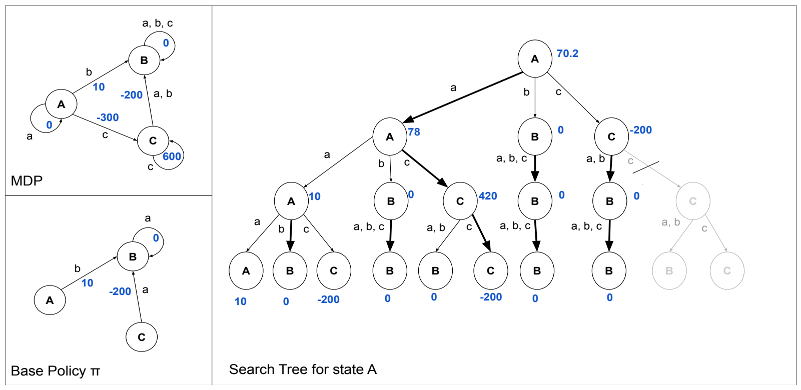


Figure 3.1: Figure shows a search tree constructed by an unspecified search procedure for the shown deterministic MDP. The base policy is shown and the leaf nodes are assigned the value of the base policy and every internal state node includes the base-policy action. The grayed out part of the tree is the part not expanded by the search procedure. The values of the internal state nodes have been computed via Bellman backup using a discount factor of 0.9. The best action at the root is a since it leads to the depth 1 state with the highest value. The text describes how this choice is not π-safe.

Online Search for MDPs. In online search, actions are selected only for the states actually encountered during online operation. At each decision point, online search constructs a finite-horizon search tree rooted at the current state. A search tree alternates between layers of state and action nodes. The leaf nodes of a search tree are state nodes and are often evaluated via a state evaluation function. The tree is used to estimate action values at the root, and the action with the highest estimate is executed in the environment.

When a model of the MDP is available, an expectimax tree can be built that assigns exact probabilities to child states of actions. For large enough search depths or accurate leaf evaluations, near-optimal actions can be selected. In some applications, only a simulator of the MDP is available, which allows for sampling state transitions and rewards. Monte-Carlo sampling can then be used to construct an approximation to the exact tree by sampling a number of child states for each action node. The Sparse Sampling algorithm [27] follows this approach and guarantees near optimal action selection in time independent of the size of the state space. Monte-Carlo Tree Search algorithms also use simulators for online search, typically producing search trees of non-uniform depth.

Online Search for Policy Improvement. In practice, online computational constraints can limit the search tree size, which can lead to poor performance of online search. To address this issue, it is common to learn or provide different types of prior knowledge into the search process. For example, the search depth can be reduced by utilizing higher-quality leaf evaluation functions, or the search breadth can be reduced via action pruning functions.

While such knowledge sources can reduce computational cost, there are typically no guarantees on the value achieved by the online search procedure. Most theoretical results aim to guarantee near optimal performance (e.g., [27]), but require impractical computational costs. Rather, it is desirable to develop approaches that support performance guarantees within practical computational limits. This motivates the framework of OSPI.

An OSPI procedure takes a policy, an environment simulator, an optional leaf evaluation function and a state as input and produces an action as output. OSPI aims for performance guarantees relative to a base policy π. The policy may be learned or hand-coded, but is assumed to be computationally cheap to apply. While π could be directly used for online action selection, this may not fully use the computational resources. OSPI aims to leverage those resources to improve over π via online search to explore the decision space around π. An OSPI procedure is *π-safe* when its online performance is guaranteed to be as good or better than that of π. That is, if π' is the online policy computed by an OSPI procedure, then the procedure is π-safe if $V^{\pi'}(s) \geq V^{\pi}(s)$ for all states s. While safety is a less powerful guarantee compared to near optimality, in practice, it is more attainable and still useful.

It can be difficult to determine, in general, whether a given OSPI procedure is π-safe. For example, one might expect that an OSPI procedure that considers the actions of π at every tree node and uses V^{π} for leaf evaluation might be π-safe when combined with a perfect environment model. However, this is not the case as the following counterexample shows. The next section develops a framework for assessing the safety of OSPI procedures.

Counterexample. Figure 3.1 shows a deterministic MDP, a base policy π, and the search tree constructed for state A by an unspecified deterministic search procedure, which produces the same tree every time state A is encountered. The tree respects the exact MDP model and the leaf evaluation function is exactly V^{π}. Further, each node of the search tree includes the action corresponding to the choice of π.

The action choice of the search procedure (i.e., the highest valued root actions) will be denoted by the policy π'. Given the tree properties relative to π, we might expect that the value of π' would be at least as good as π. For state A, the base policy selects $\pi(A) = b$ and the corresponding value of state A under π is $V^{\pi}(A) = 10$. However, the online search suggests the alternative action $\pi'(A) = a$, which results in a lower value of $V^{\pi'}(A) = 0$. Thus, the online search procedure is not π-safe, at least for state A.

To understand the failure to be π-safe at state A, consider using the search tree to make a decision at state A at time step t. The reason action a looks best is that the state-action sequence $A \rightarrow a \rightarrow A \rightarrow c \rightarrow C \rightarrow c \rightarrow C$ achieves a high value due to the 600 reward of the final transition. However, after actually taking action a and ending up in state A again at time step $t+1$ the tree does not include the promising path of $A \rightarrow c \rightarrow C \rightarrow c \rightarrow C$, due to pruning of the lower levels of the search tree. Thus, at time step $t+1$ the search procedure does not recognize the value of taking action c in A and takes a again. This is just one of several failure-mode types of OSPI procedures, even when their trees satisfy the assumptions of this example relative to π.

3.5 The Choice Function Framework

Search trees encode the future trajectories to be considered when evaluating actions at a state. Search algorithms vary in how they expand the paths, which results in different search trees and hence different action values. Thus, one way to characterize online search approaches is by describing the trees they construct. In our framework, this is done using choice functions, which allows for properties, such as safety of search, to be analyzed via choice-function properties.

Choice Functions for General Online Search

Search trees have two sources of branching: 1) action branching and 2) state branching. Choice functions describe the action branching by specifying the subset of possible action choices to be considered at each state node. Leaf nodes are assigned the empty set of choices. For state branching, we assume an exact MDP model so that all non-zero probability child states of an action node are included in the tree. When a model is not available, but a simulator is, sparse sampling can be used to approximate the dynamics.

State and action nodes in a tree are identified by paths that list the alternating sequence of states and actions starting at the root state. The path of a state node labeled with state s will be denoted by $p; s$ where p is the path starting at the root leading to the parent action node of the state node. Action nodes will often similarly be denoted by $p; s; a$, where $p; s$ designates the parent state node. The length of any path denoted by $|p|$ is the number of actions that it contains. Thus, a path corresponding to a single state s has length zero. The set of all paths that end with a state is denoted by $\mathcal{SP}$ and a *choice function* is a mapping $\psi : \mathcal{SP} \rightarrow 2^A$ from paths that end with a state to action subsets.

In order to define the trees associated with a choice function, several definitions are needed. A path is *ψ-satisfying* if all of its actions are "allowed" by ψ. That is, for each prefix $p'; s'; a'$ of the path, we have $a' \in \psi(p'; s')$. A state path $p; s$ is a *leaf path* of ψ if it is ψ-satisfying and $\psi(p; s) = \emptyset$. A leaf path of ψ cannot be extended to a ψ-satisfying path. A choice function ψ is *finite horizon* if there is a finite upper bound on the length of any ψ-satisfying state path. For finite-horizon ψ, the *horizon* $H(\psi)$ is the maximum length of any ψ-satisfying path, or equivalently, of any leaf path.

Given a current, or root state, s_0, the tree corresponding to ψ, denoted $T^{\psi}(s_0)$, is the tree containing exactly the ψ-satisfying state paths that begin with s_0. Thus, the leaf nodes of $T^{\psi}(s_0)$ correspond to leaf paths of ψ. The tree will be finite when ψ is finite horizon, with $H(\psi)$ bounding the depth of any leaf node. In this paper, we will restrict attention to finite-horizon choice functions and hence finite trees.

To use the tree $T^{\psi}(s_0)$ for action selection at state s_0, it is necessary to specify a *leaf evaluation function* u, which is a function of states $u : S \rightarrow \mathbb{R}$. Often u will be a learned or hand-coded function that provides an estimate of a state's optimal value or value under a policy. Alternatively, u may be uninformative and return a constant value. Together, a choice function ψ and leaf evaluation function u allow us to define the value of each state node $p; s$ in $T^{\psi}(s_0)$, denoted $V_u^{\psi}(p; s)$, and each action node $p; s; a$, denoted $Q_u^{\psi}(p; s; a)$.

$$\begin{aligned} V_u^{\psi}(p;s) &= \begin{cases} u(s), & \psi(p;s) = \emptyset, \\ \max\limits_{a \in \psi(p;s)} Q_u^{\psi}(p;s;a), & \textit{otherwise}. \end{cases} \\ Q_u^{\psi}(p;s;a) &= R(s,a) + \gamma \sum_{s' \in S} P_{ss'}(a) \cdot V_u^{\psi}(p;s;a;s'). \end{aligned}$$

The online-search action policy, denoted Π_u^{ψ}, returns a maximum valued action at state s allowed by ψ, with ties broken arbitrarily: $\Pi_u^{\psi}(s) = \arg\max\limits_{a \in \psi(s)} Q_u^{\psi}(s;a)$.

Choice Functions for OSPI

Given a base policy π we would like to define choice functions and corresponding leaf-evaluation functions that result in (approximately) π-safe OSPI procedures. That is, we would like to guarantee that Π_u^{ψ} is π-safe. Section 3.6 develops sufficient conditions on choice functions to give such a guarantee. First, however, we provide examples of choice functions for several existing OSPI procedures whose safety will later be assessed according to the conditions.

Policy Rollout. This simple OSPI procedure [56] returns the action at state s that maximizes a Monte-Carlo estimate of $Q^{\pi}(s,a)$. This estimate can be viewed as evaluating a tree that considers all actions at the root s and then only contains the actions of π thereafter until some horizon H. Policy rollout can thus be characterized by the following choice function.

$$\psi_{\text{ro}}(p;s) = \begin{cases} A, & |p| = 0, \\ \{\pi(s)\}, & 0 < |p| < H, \\ \emptyset, & \text{otherwise}. \end{cases}$$

Policy rollout can be proven to be π-safe as it corresponds to the policy improvement step of the policy iteration algorithm. Our π-safe conditions will imply this for ψ_{ro}.

Limited Discrepancy Search (LDS). This procedure was originally introduced for deterministic offline search problems [17]. LDS searches around π by limiting the number of discrepancies (off-policy actions) along every root-to-leaf path to K up to some maximum horizon H. The idea was later extended to offline non-deterministic AND/OR tree/graph search [33] using a similar limit on discrepancies. LDS for MDPs is easily captured via the following choice function, where $\#[p \neq \pi]$ is the number of off-policy actions in path p.

$$\psi_{lds}(p; s) = \begin{cases} A, & |p| < H \text{ and } \#[p \neq \pi] < K, \\ \{\pi(s)\}, & |p| < H \text{ and } \#[p \neq \pi] = K, \\ \emptyset, & |p| = H. \end{cases}$$

Our conditions will imply that ψ_{lds} is π-safe.

Pruned Online Search with Learned Policies. Reinforcement Learning (RL) algorithms typically learn policies that select actions by maximizing an action ranking function, such as a Q-function or probability distribution over actions. Such action rankings can be used for action pruning in online tree search. Let $q(p; s, a)$ be the learned action ranking function, which may depend on the full path $p; s$ (e.g., when q is a recurrent neural network) or depend only on s. A simple pruning approach such as the one studied in Pinto et al. [41], allows only the set of top k actions at each search node, denoted $\text{TOP}_{q,k}(p; s)$, as captured by the following choice function.

$$\psi_{q,k}(p; s) = \begin{cases} \text{TOP}_{q,k}(p; s), & |p| < H, \\ \emptyset, & |p| = H. \end{cases}$$

As discussed in Section 3.7, our results will help clarify conditions on q that ensure safety.

3.6 Performance Guarantee

Our goal is to identify properties of a choice function ψ that guarantee that the online-search action policy Π_u^ψ is approximately π-safe. That is, we seek to bound $V^\pi(s) - V^{\Pi_u^\psi}(s)$. A natural property to suggest is that ψ be consistent with π. A choice function ψ is π-consistent if $\pi(s) \in \psi(p; s)$ for each path $p; s$ that ends with a state. Our counterexample in section 3.4, however, is based on a π-consistent choice function, since all tree nodes include π. Thus, π-consistency of ψ is not sufficient for π-safety, requiring the introduction of additional concepts and notation.

We will often treat value functions as vectors, indexed by states, with arithmetic and comparison operators being applied element-wise. The max-norm of a vector $\|V\|_\infty$ returns the maximum absolute value of the elements. The *min-horizon* of ψ, denoted $h(\psi)$, is the minimum depth of any leaf node in $T^\psi(s_0)$ for any state s_0. Given a path $p; s$ we let $\lrcorner p; s$ denote the path obtained by removing the leftmost state-action pair of $p; s$. We say that ψ is *monotonic* if the set of actions returned by ψ for state s when reached via path p is a subset of the set of actions returned for s when reached via the path with the leftmost state-action pair of p removed, i.e., $\psi(p; s) \subseteq \psi(\lrcorner p; s)$ for all $p; s \in \mathcal{SP}$. We can now give our main result.

Theorem 3. *For any MDP, discount factor γ, and policy π, if ψ is π-consistent and monotonic and $\|u - V^\pi\|_\infty \le \epsilon$, then for $\pi' = \Pi_u^\psi$,*

$$V^\pi - V^{\pi'} \le \frac{2\epsilon\gamma^{h(\psi)}}{1-\gamma}.$$

This bound implies that in the ideal case when $u = V^\pi$, monotonicity and π-consistency together are sufficient for safety. It also shows that the impact of inaccuracy in u with respect to V^π decreases exponentially with the min-horizon due to discounting of future returns.

From the theorem we get an immediate corollary that applies to the set of all policies ψ is consistent with, denoted $\mathcal{C}_\psi$, where $\epsilon(u, \pi) = \|u - V^\pi\|_\infty$.

Corollary 1. *For any MDP, discount factor γ, leaf evaluation function u, π-consistent and monotonic choice function ψ, the policy $\pi' = \Pi_u^\psi$ satisfies*

$$V^{\pi'} \ge \max_{\pi \in \mathcal{C}_\psi} \left[V^\pi - \frac{2\epsilon(u,\pi)\gamma^{h(\psi)}}{1-\gamma} \right].$$

This implies that for a large min-horizon, the online policy is guaranteed to be safe with respect to the best policy that ψ is consistent with. In general, this shows the performance trade-off between larger min-horizons (i.e., minimum search depth) and the closeness of u to a good policy.

Proof of Theorem 3

All proofs not in the main text are in the appendix (section 7.1). The high-level idea of our proof is inspired by the analysis of offline multi-step policy improvement [6]. This procedure starts with the value function $V_0 = V^\pi$ and then performs m applications of the *Bellman Backup* operator B to get a sequence of value functions $V_i = B[V_{i-1}]$, where B is defined as follows.

$$B[V](s) = \max_{a \in A} R(s,a) + \gamma \sum_{s' \in S} P_{ss'}(a) \cdot V(s')$$

The greedy policy π_m with respect to the final value function V_m is then returned as the improved policy over π. The monotonicity of B can be used to guarantee that $V^{\pi_m} \geq V^\pi$.

An OSPI procedure for computing $\pi_m(s)$ is to evaluate a depth $m+1$ tree with root s using V^π for leaf values. This computes the greedy action with respect to V_m, but without synchronously updating all states from the bottom up. Thus, the offline guarantee carries over to OSPI. OSPI with choice functions can be viewed similarly but with backups restricted to actions allowed by the choice function. Our analysis below generalizes these ideas to path-sensitive choice functions and approximate leaf values.

To prove the main result we start by introducing a number of lemmas. It will be useful to introduce the *policy-restricted Bellman Backup* operator $B_\pi[V]$, which restricts backups to only consider the actions of π.

$$B_\pi[V](s) = R(s,\pi(s)) + \gamma \sum_{s' \in S} P_{ss'}(\pi(s)) \cdot V(s').$$

Lemma 1 gives a lower bound on V^π in terms of a value vector V and how much B_π decreases the value of V.

Lemma 1. *For any policy π and value vector V, if $V - B_\pi[V] \leq \delta$, then $V - V^\pi \leq \dfrac{\delta}{1-\gamma}$.*

Next, Lemma 2 generalizes the conditions that guarantee policy improvement in the offline multi-step lookahead policy improvement procedure to OSPI with choice functions. Proposition 1 follows from lemma 2.

Lemma 2. *If a stationary choice function ψ is π-consistent and monotonic and $u = V^\pi$ then for any path $p;s$ such that $1 \leq |p;s| \leq H(\psi)$, $V_u^\psi(\lrcorner p;s) \geq V_u^\psi(p;s)$.*

Proposition 1. *If a stationary choice function ψ is π-consistent and monotonic and $u = V^\pi$, then $V_u^\psi(s) \geq V^\pi(s)$.*

Lemma 3 below bounds the values of paths of length less than or equal to $h(\psi)$ for two different leaf evaluation functions u and u' satisfying $\|u - u'\|_\infty \leq \epsilon$. This will be useful for quantifying the impact of the leaf evaluation function being an approximation to the base policy value function.

Lemma 3. *If ψ is a stationary choice function and $\|u - u'\|_\infty \leq \epsilon$ then for any path $p;s$ with $|p;s| \leq h(\psi)$, $\left|V_u^\psi(p;s) - V_{u'}^\psi(p;s)\right| \leq \epsilon\gamma^{h(\psi)-|p;s|}$.*

For the following we use the notation $V_{u,k}^\psi$ to denote the vector consisting of the elements of V_u^ψ for $|p;s| = k$. In particular, $V_{u,0}^\psi$ is the vector of the values of all root nodes, i.e., $|p;s| = 0$. Lemma 4 and proposition 2 below are key results that combine to bound the difference between $V_{u,0}^\psi$ and the value of Π_u^ψ.

Lemma 4. *If a stationary choice function ψ is π-consistent and monotonic and $\|u - V^\pi\|_\infty \leq \epsilon_\pi$, then for $\pi' = \Pi_u^\psi$, $V_{u,0}^\psi - B_{\pi'}[V_{u,0}^\psi] \leq \epsilon_\pi\gamma^{h(\psi)}(1+\gamma)$.*

Proposition 2. *If a stationary choice function ψ is π-consistent and monotonic and $\|u - V^\pi\|_\infty \leq \epsilon_\pi$, then for $\pi' = \Pi_u^\psi$, $V_{u,0}^\psi - V^{\pi'} \leq \dfrac{\epsilon_\pi\gamma^{h(\psi)}(1+\gamma)}{1-\gamma}$.*

Proof. Directly combine lemmas 4 and 1. □

Using the above lemmas we can now prove the main result.

Theorem 3. *If a stationary choice function ψ is π-consistent and monotonic and $\|u - V^\pi\|_\infty = \epsilon_\pi$, then for $\pi' = \Pi_u^\psi$,*

$$V^\pi - V^{\pi'} \leq \frac{2\epsilon_\pi\gamma^{h(\psi)}}{1-\gamma}.$$

Proof. In the proof, V^π is denoted as $\bar{u}$ to simplify notation.

$$\begin{aligned}
V^\pi(s) - V^{\pi'}(s) &= V^\pi(s) - V^\psi_{u,0}(s) + V^\psi_{u,0}(s) - V^{\pi'}(s). \\
&\leq V^\pi(s) - (V^\psi_{\bar{u},0}(s) - \epsilon_\pi \gamma^{h(\psi)}) + V^\psi_{u,0}(s) - V^{\pi'}(s), \ \textit{by lemma 3} \\
&\leq \epsilon_\pi \gamma^{h(\psi)} + V^\psi_{u,0}(s) - V^{\pi'}(s), \ \textit{since } V^\psi_{\bar{u},0}(s) \geq V^\pi(s) \ \textit{by proposition 1} \\
&\leq \epsilon_\pi \gamma^{h(\psi)} + \frac{\epsilon_\pi \gamma^{h(\psi)}(1+\gamma)}{1-\gamma}, \qquad \textit{by proposition 2} \\
&= \frac{2\epsilon_\pi \gamma^{h(\psi)}}{1-\gamma}.
\end{aligned}$$

□

Non-Stationary Choice Functions

We have assumed that choice functions are stationary, i.e., the same choice function is used across online decision steps. Some OSPI approaches, however, correspond to a non-stationary choice function that varies across steps. For example, some search algorithms use a sub-tree produced at time step t as a starting point for search at time step $t+1$. Randomized OSPI approaches are non-stationary due to different random seeds across steps. Here, we extend our analysis to the non-stationary case.

A non-stationary choice function $\Psi = (\psi_1, \psi_2, \psi_3, \ldots)$ is a sequence of time-step indexed stationary choice functions ψ_t. To relate two different stationary choice functions ψ and ψ' we say that ψ subsumes ψ', denoted $\psi \supseteq \psi'$, if for every path $p; s$, $\psi(p; s) \supseteq \psi'(p; s)$. We can extend the bound in Theorem 3 to a non-stationary choice function Ψ when each ψ_t satisfies the conditions of that theorem, each ψ_t has the same set of leaf paths, and $\psi_{t+1} \supseteq \psi_t$ for each time-step t.

Theorem 4. *Let $\Psi = (\psi_1, \psi_2, \ldots)$ be a non-stationary choice function such that each component choice function ψ_t is monotonic and π-consistent and $\|u - V^\pi\|_\infty = \epsilon_\pi$. If all ψ_t have the same set of leaf paths and for each time-step t, $\psi_{t+1} \supseteq \psi_t$, then for $\pi' = \Pi^\psi_u$ and all $s \in S$,*

$$V^\pi(s) - V^{\pi'}(s) \leq \frac{2\epsilon_\pi \gamma^{h(\psi_1)}}{1-\gamma}.$$

The proof is in the appendix (section 7.1). This result has implications on the design of OSPI procedures that correspond to non-stationary choice functions. For example, many MCTS-based approaches, such as that used by AlphaZero, do not appear to correspond to non-stationary choice functions that satisfy these conditions. This does not mean that they will not perform well in a particular application, but suggests that for some applications they can have fundamental issues that degrade the performance of a base policy, even ignoring inaccuracies due to Monte-Carlo sampling. One way to adjust some of these and other algorithms to achieve the subsumption property is to build upon the relevant subtrees constructed at time t at time step $t + 1$. The practical impact of such a change is an empirical question worth investigating.

3.7 Limited Discrepancy Choice Functions

We do not expect a single type of choice function to perform best across all problems. Rather, in practice, the selection of a choice function can be similar to the selection of other application-dependent hyperparameters. This motivates defining parametric families of choice functions that span different trade-offs between decision time and quality. In particular, given an application's decision-time constraints, offline optimization can be used to select a high-performing choice function that satisfies the constraints.

To support this vision, we introduce the parametric family of *Limited Discrepancy Choice Functions (LDCFs).* We show that all LDCFs are monotonic and π-consistent for any π and hence satisfy our safety conditions. We then analyze how the parameters of the LDCF family relate to the computational complexity of online action selection. Finally we relate LDCFs to previously introduced examples.

LDCF Definition and Safety. An LDCF ψ defines a uniform-horizon tree, which limits the discrepancies w.r.t. a base policy along root-to-leaf paths by their number and depth. In cases where a base policy only makes occasional errors the discrepancy limit makes intuitive sense. Indeed, search can improve the policy by "correcting" the rare errors along paths by introducing discrepancies. This suggests that there can be a computational advantage to bounding search by discrepancies in applications of OSPI to learned policies that already perform well but can still be improved.

A discrepancy w.r.t. π in path p is a state-action pair (s, a) in p such that $a \neq \pi(s)$. LDCFs are parameterized by the tuple (π, H, K, D, Δ), where π is the base policy, H is the uniform horizon bound, $K \leq H$ is a bound on the number of discrepancies in a path, and $D < H$ is a bound on the maximum depth that a discrepancy may appear in a path. Finally, the *discrepancy proposal function* $\Delta : S \times \{0, \ldots, D\} \to 2^A$ maps pairs of states and depths to action subsets. Intuitively Δ returns the discrepancies that can be considered at a state node $p; s$ at depth $|p|$ which has not yet reached the discrepancy limit imposed by K and D. We allow Δ to depend on depth, since it is often useful to allow for more discrepancies at shallower depths. Given parameters $\theta = (\pi, H, K, D, \Delta)$ the corresponding LDCF is defined as follows.

$$\psi_\theta(p; s) = \begin{cases} \Delta(s, |p|) \cup \{\pi(s)\}, & |p| \leq D \text{ and } \#[p \neq \pi] < K, \\ \{\pi(s)\}, & D < |p| < H \text{ or } \#[p \neq \pi] = K, \\ \emptyset, & |p| = H. \end{cases}$$

All members of the LDCF family are π-consistent by construction. However, monotonicity of an LDCF requires constraining Δ. We say that Δ is *depth monotonic* if $\Delta(s, d) \supseteq \Delta(s, d + 1)$ for all $s \in S$ and d.

Theorem 5. *For LDCF parameters $\theta = (\pi, H, K, D, \Delta)$, if Δ is depth monotonic, then ψ_θ is monotonic.*

The proof is in the appendix (section 7.1). A straightforward way to obtain a depth-monotonic Δ is to use a learned action-ranking function over states and return the top ranked actions at a state, where the number of returned actions is non-increasing with tree depth.

Application to Special Cases. The choice function ψ_{lds} is a restricted LDCF with $\Delta(s, d) = A$, which trivially satisfies our safety conditions. The LDCF space provides more flexibility on how to better control the introduction of discrepancies compared to traditional LDS.

The policy rollout choice function ψ_{ro} is a special case of an LDCF with $D = 0$ and $\Delta(s, 0) = A$, which allows all choices at the root and only the base policy's choices thereafter. Since Δ is trivially depth monotonic, our safety result applies. This can be generalized to multi-step look-ahead rollout [6], where the top M levels of the tree are fully expanded followed by restricting actions to those of the base policy until the horizon. Specifically, $D = M$ and $\Delta(s, d) = A$, which again satisfies our safety conditions. Note that when $D = H - 1$ this degenerates to value iteration with horizon H.

Finally, the pruned-search choice function $\psi_{q,k}$ is an LDCF with a specific choice of discrepancy function $\mathrm{TOP}_{q,k}$. Our safety conditions specify sufficient constraints that the action ranking function q should satisfy. When q is history-independent, $\mathrm{TOP}_{q,k}$ is depth-monotonic. Otherwise, if q is history-dependent, e.g., a recurrent neural network, no such guarantee can be made. However, it is relatively straightforward to put a wrapper around such a q to ensure depth monotonicity.

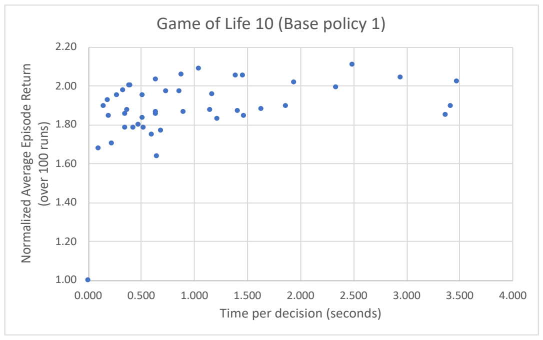
Computational Complexity. Increasing H, K, and D, and the size of sets returned by Δ can be expected to improve $\Pi_u^{\psi_\theta}$ for reasonable u. This comes at the cost of higher computational complexity typically dominated by the number of leaves in T^{ψ_θ}. In addition to the LDCF parameters, the number of leaf nodes depends on the *state branching factor* C, which is the maximum number of state nodes under an action node. When an exact model is used, this is the maximum number of non-zero probability successor states. For Monte-Carlo algorithms, this is the number of successor states sampled for each action node. Given the state branching factor and an upper bound on the number of actions returned by Δ, we get the following bound on tree size.

Proposition 3. *Let ψ_θ be an LDCF with $\theta = (\pi, H, K, D, \Delta)$, such that $\Delta(s) \leq W$ for any $s \in S$. The number of leaf nodes in T^{ψ_θ} with state branching factor C is upper bounded by $2C^H$ for $(D+1)W = 1$ and by $\frac{((D+1)W)^{K+1}-1}{(D+1)W-1}C^H = O\left((DW)^K C^H\right)$ for $(D+1)W \neq 1$.*

Ignoring the impact of C, which is controlled by the search algorithm, the complexity is dominated by K. We also see that for deterministic domains where $C = 1$, there is no exponential dependence on H.

3.8 Illustrative Empirical Results

Our primary contribution is theoretical. However, here we illustrate the potential practical utility of our framework using the LDCF family. The experiments are intended to demonstrate how the choice function functions as a hyperparameter to be tuned offline.



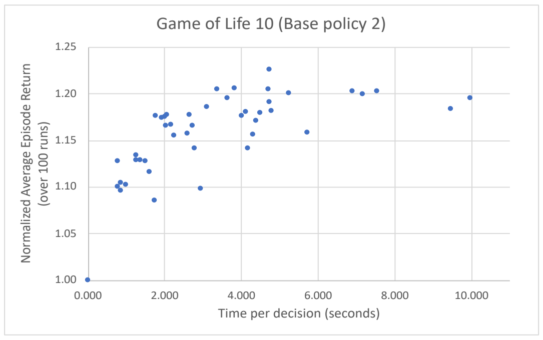


Figure 3.2: Performance vs time-per-step for LCDF choice functions applied to linear (left) and non-linear (right) base policies.

We implemented a variant of Forward Search Sparse Sampling (FSSS) [59] for approximately computing the online policy Π_u^ψ for any LDCF ψ and leaf-evaluation function u using an MDP simulator. The key parameter, other than the choice-function, is the sampling width C, which controls how many state transitions are sampled for each action node. The appendix (section 7.1) contains a summary of the algorithm. Our implementation can be found here: `https://bitbucket.org/eshkim/ld-fsss/src/master/`.

Experiments Setup. We run experiments in the domain Game-of-Life, a benchmark domain from the International Probabilistic Planning Competition. This is a grid-based domain with each grid-cell either being alive with some probability depending on its neighbors. Actions allow for selecting one cell at each time step (or none) to set to be alive in the next time step. The reward is based on the number of alive cells at each step. There are 10 problems of grid sizes 3×3, 4×4, 5×5 and 10×3. Problems have different levels of stochasticity in the dynamics.

We used supervised learning via imitation of a planner to train two base policies represented as neural networks, using the same approach as in Issakkimuthu et al. [23]. Each network outputs a probability distribution over actions. Policies 1 and 2 are base policies. Policy 1 is a linear network, while Policy 2 is a non-linear network with 3 hidden layers. For each base policy, we consider four leaf evaluation functions. The first is the constant zero function. The remaining three are neural networks with different configurations trained on a dataset of 5000 state-value pairs obtained via Monte-Carlo simulation of each policy. All the networks have been trained using Tensorflow to minimize the mean squared error.

We experiment with 11 choice functions in the LDCF family with parameters H in $\{3, 4, 5\}$ and (D, K) in $\{(0, 1), (1, 1), (1, 2), (2, 1)\}$. The combination $(2, 1)$ is not applicable for $H = 3$. The number of non base-policy actions considered at the root node is 9 and other internal nodes is 1. The non base-policy actions are determined from the action probabilities given by the base policy. We use $C = 3$ for FSSS.

Given one of these policies and a problem, we would like to identify the best combination of LDCF parameters and leaf evaluation function given a constraint on the time-per-step. In practice, this could be done in an offline tuning phase where different configurations are evaluated. Figure 3.2 shows a scatter plot of the normalized reward versus time-per-step for each of the 44 configurations (11 LDCF settings and 4 leaf evaluation functions). The normalized reward is the average reward per episode divided by the average reward per episode of the base policy. Values greater than 1 perform better than the base policy. For both base policies, all LDCF configurations perform better. There is also a larger improvement for base policy 1, which makes sense due to the fact that policy 2 is a much higher-quality policy and hence more difficult to improve. We also see that the LDCF space shows considerable coverage of the performance vs. time space, which shows the utility of offline tuning. There is a general trend toward better performance for larger times, but this is not uniformly true. There are complex interactions between the LDCF parameters and a particular problem, which makes it unlikely that a single feature such as time-per-decision is always the best indicator.

Table 3.1 gives results for each of the 10 problems, which includes the normalized rewards with confidence intervals for the best performing LDCF configuration for each of the policies. We see that for the linear policy, the best LDCF configuration is never significantly worse (lower interval is greater than 1) and often significantly better. For the second non-linear policy, we see that for most problems the LDCF performance is not significantly worse than the policy (confidence interval contains 1) and sometimes significantly better. For three problems, the upper confidence bound is less than one, indicating a significant decrease in performance. These problems happen to be among the most stochastic problems in the benchmark set. This suggests that a likely reason for the decrease in performance is due to the relatively small sampling width used for FSSS ($C = 3$), which provides a poor approximation for such problems.

	LDCF Policy 1		**LDCF Policy 2**	
Prob. #	Normalized Avg. Reward	Decision Time (s)	Normalized Reward	Decision Time (s)
1	2.57 ± 0.07	0.081	1.08 ± 0.04	0.491
2	1.27 ± 0.10	0.345	0.95 ± 0.06	0.631
3	1.11 ± 0.05	0.129	0.92 ± 0.03	0.374
4	1.51 ± 0.03	0.523	1.03 ± 0.02	0.830
5	1.14 ± 0.03	1.084	1.00 ± 0.03	6.298
6	1.05 ± 0.02	1.223	0.96 ± 0.02	9.163
7	1.54 ± 0.02	0.523	1.05 ± 0.01	1.957
8	1.21 ± 0.02	4.488	1.02 ± 0.02	10.463
9	1.13 ± 0.02	3.262	0.96 ± 0.01	3.749
10	2.11 ± 0.04	2.493	1.23 ± 0.02	4.746

Table 3.1: Game-of-Life - Best Normalized reward.

3.9 Summary

We have introduced a framework for analyzing online search procedures for policy improvement guarantees. The key idea is to separate the action specification part of search from the search process and create an abstract concept called choice functions. A choice function instance will then be a parameter of search. We identify properties of choice functions to provide sufficient conditions for guaranteed online policy improvement when the leaf evaluation function is perfect. Our main result is a bound on the performance of the online policy relative to the base policy for any leaf evaluation function. We have also introduced a parameterized class of choice functions called LDCF. Our next directions are to explore the practical application of the framework across a wide range of problems and to integrate notions of state abstraction into the framework.

Online Policy Improvement for Probabilistic Planning: Benchmarks and Baselines

Murugeswari Issakkimuthu and Alan Fern

Workshop on the International Planning Competition (WIPC) at the 31st International Conference on Automated Planning and Scheduling (ICAPS-2021)

Chapter 4: Online Policy Improvement for Probabilistic Planning: Benchmarks and Baselines

4.1 Abstract

The goal of Online Policy Improvement (OPI) is to use a given base policy to compute a better policy via online planning. There are OPI algorithms that come with theoretical guarantees of policy improvement under ideal conditions. However, when the ideal conditions are not met in practice, these algorithms can result in policy degradation, i.e., the new policy can perform worse than the base policy. Our goal in this paper is to move towards a better understanding of the empirical performance of OPI algorithms. We propose benchmark problems and base policies and suggest evaluation metrics for OPI. We also present baseline results on the benchmark set for two OPI algorithms, which demonstrate the baselines are a solid starting point for comparison.

4.2 Introduction

Online planning is a practical approach to solving Markov decision problems with large state spaces. An action choice is made for the current state, and the selected action is executed immediately, so action decisions need to be made only for the states visited during the online planning process. Online planning aims at computing a near optimal policy. Online policy improvement (OPI) is online planning with the goal of computing a policy that performs better than a given base policy.

When a base policy is available, OPI can sometimes be a safer alternative to optimal planning. For example, attempts at optimal planning under computational limits may completely fail, while OPI may provide useful results. There are different approaches to online planning. Our focus is on search-based approaches, where the Q-values of actions at the current state are estimated via lookahead search. Online planning returns an action that maximizes the Q-value estimate, while OPI can return any action with a Q-value estimate greater than that of the base policy action.

There are OPI algorithms that come with theoretical guarantees of policy improvement under ideal conditions, e.g., policy rollout [56], [6], nested rollout [11], parallel rollout [12], and Limited Discrepancy Forward Search Sparse Sampling (LD-FSSS) [24]. However, when the ideal conditions are not met in practice, OPI algorithms can result in policy degradation, i.e., the new policy can perform worse than the base policy.

Our goal in this work is to move towards a better understanding of the empirical performance of OPI algorithms. We propose benchmark problems and base policies and suggest evaluation metrics for OPI. Our benchmark set consists of 5 domains from past International Probabilistic Planning Competitions with 10 problems of varying levels of difficulty in each domain and 2 base policies of different qualities for each problem. We also present baseline results on the benchmark set for two classes of OPI algorithms. In particular, these classes include algorithms that can leverage transition probabilities when available or just use the ability to sample transitions. We show that these classes are able to cover different points in the performance trade-off space, making them useful for future comparisons.

4.3 Background and Related Work

We assume basic familiarity with Markov Decision Processes (MDPs). A discrete finite-horizon MDP is a tuple $\langle S, A, P, R, H \rangle$, where S is a finite set of states, A is a finite set of actions, $P : S \times A \times S \to [0, 1]$ is a state-transition function with $P_{ss'}(a)$ denoting the probability of reaching state s' from state s on action a and $\sum_{s' \in S} P_{ss'}(a) = 1$ for all $a \in A$, $R : S \times A \to \mathcal{R}$ is a real-valued reward function defined on state-action pairs, and H is an integer representing the finite horizon.

A deterministic, non-stationary policy of the MDP is a time-dependent mapping from states to actions, i.e., $\pi = \{\mu_0, \mu_1, \ldots, \mu_{H-1}\}$, where $\mu_k : S \to A$ for $k = \{0, 1, \ldots, H-1\}$. The H steps-to-go value function of the policy is V_H^π, where

$$V_k^\pi(s) \;=\; R(s, \mu_{H-k}(s)) + \sum_{s' \in S} P_{ss'}(\mu_{H-k}(s)) \cdot V_{k-1}^\pi(s'),$$

for $k = \{1, 2, \ldots, H\}$ and $V_0^\pi(s) = 0$ for all $s \in S$. The H steps-to-go Q-value function with respect to π is

$$Q^{\pi}_{H}(s,a) \;=\; R(s,a) + \sum_{s' \in S} P_{ss'}(a) \cdot V^{\pi}_{H-1}(s'),$$

for all $s \in S$ and $a \in A$. A policy π' is said to be better than a policy π if $V^{\pi'}_{H}(s) \geq V^{\pi}_{H}(s)$ for all $s \in S$. A solution of the MDP is an optimal policy π^* with value function $V^{*}_{H}(s) = \max_{\pi} V^{\pi}_{H}(s)$ for all $s \in S$.

Online Planning

An MDP can be solved offline via approaches such as value iteration, policy iteration or linear programming [42]. The offline solution techniques can be computationally expensive for MDPs with large state spaces. An alternative practical approach is online planning, where plan execution is interleaved with planning. Action decisions are made only for the initial state and the states visited subsequently in the online planning process.

Our focus is on search-based online planning, where the Q-values of actions at the current state are estimated via finite-horizon lookahead search. Typically, a search tree is built with the current state at the root followed by alternating layers of action nodes and state nodes. Leaf nodes are initialized and the values of internal nodes are computed from the values of their successor nodes. An action that maximizes the Q-value estimate is returned for the current state. There are variants of search-based online planning, e.g., Sparse Sampling (SS) [27], Forward Search Sparse Sampling (FSSS) [59], Monte Carlo Tree Search (MCTS) [9].

Base Policies in Online Planning. Online planning does not require a base policy, but it can benefit from one if there is one available. MCTS algorithms typically employ a default base policy to initialize the values of leaf nodes using one or more rollouts of the base policy. The well-known AlphaGo and AlphaZero programs [54, 53] use base policies to expand their search trees. Nguyen et al. [39] use a base policy as an extended action at every node of the search tree to identify better actions at the nodes. Pinto et al. [41] use a partial policy that gives a subset of actions for every state to successfully prune actions in the search tree. All these approaches can be roughly viewed as a form of OPI, even though the goal is not just to perform better than the base policy.

4.4 Online Policy Improvement

Online Policy Improvement (OPI) has the goal of doing better than a given base policy, so a base policy is a required input for OPI algorithms. Once the Q-values of actions are estimated for the current state, OPI can return any action with a Q-value greater than that of the base policy action. OPI can therefore be done with as few as one off-policy action (non base-policy action) and the base policy action at the root. There are several existing OPI algorithms that come with theoretical guarantees of policy improvement. We discuss a few such algorithms below.

Policy Rollout. The policy rollout algorithm is an online implementation of a single offline policy improvement step over the base policy value function. The policy improvement step is based on the fact that actions with Q-values greater than the Q-value of the base policy action will be better than the base policy action for a given state. When the base policy is substituted with improved actions at one or more states, the resulting policy will be better than the base policy. In the online version, the Q-value of an action at the current state is estimated as an average over multiple base policy trajectories starting with that action. The estimates will get close to the actual values when the average is computed with a large number of trajectories of sufficient length. The policy rollout algorithm has been shown to be effective in different applications [56], [5].

Nested Rollout. The nested rollout algorithm [11] is an online implementation of a sequence of iterations of the policy iteration algorithm, in contrast to the policy rollout algorithm that implements just one iteration of the policy iteration algorithm. The policy computed at each iteration of the policy iteration algorithm will be better than all the previous policies along the sequence. Hence nested rollout is guaranteed to return a better policy when the Q-values are estimated with a large number of simulated trajectories of sufficient length.

Parallel Rollout. The parallel rollout algorithm takes multiple base policies as input to compute a policy that is better than all the base policies [12]. It is an online version of the offline policy switching algorithm that returns for every state the action of the base policy with the highest value among all the base policies. The resulting policy is guaranteed to be better than all the base policies. Parallel rollout estimates the values of

Table 4.1: Performance of the two base policies

	Sysadmin		**Game of Life**		**Tamarisk**		**Skill Teaching**		**Wildfire**	
#	Bad	Good	Bad	Good	Bad	Good	Bad	Good	Bad	Good
1	171 ± 7	335 ± 5	76 ± 13	177 ± 10	-178 ± 21	-144 ± 20	65 ± 2	65 ± 2	-647 ± 308	-594 ± 251
2	250 ± 11	304 ± 9	82 ± 8	119 ± 9	-590 ± 29	-502 ± 25	-60 ± 0	75 ± 2	-9015 ± 387	-8998 ± 344
3	421 ± 15	554 ± 14	112 ± 7	137 ± 5	-264 ± 38	-222 ± 33	56 ± 15	81 ± 14	-1439 ± 412	-1529 ± 432
4	379 ± 9	487 ± 15	216 ± 17	327 ± 15	-804 ± 30	-681 ± 31	-64 ± 7	57 ± 17	-9050 ± 791	-8585 ± 801
5	518 ± 9	627 ± 17	236 ± 9	281 ± 8	-642 ± 44	-618 ± 37	-64 ± 19	-64 ± 19	-1010 ± 343	-754 ± 257
6	541 ± 15	575 ± 17	244 ± 6	276 ± 5	-952 ± 29	-860 ± 33	-16 ± 31	-10 ± 30	-7697 ± 663	-7374 ± 761
7	597 ± 10	721 ± 17	303 ± 18	462 ± 10	-826 ± 48	-706 ± 45	-297 ± 10	-82 ± 26	-5729 ± 475	-5447 ± 418
8	481 ± 11	583 ± 16	336 ± 14	429 ± 9	-1188 ± 29	-1108 ± 34	-498 ± 12	-201 ± 35	-9912 ± 583	-9792 ± 619
9	780 ± 12	839 ± 15	337 ± 12	421 ± 5	-868 ± 63	-758 ± 52	-166 ± 31	-175 ± 28	-4939 ± 715	-4840 ± 747
10	523 ± 11	616 ± 17	257 ± 22	473 ± 26	-1225 ± 41	-1087 ± 47	-623 ± 12	-239 ± 37	-10834 ± 711	-10584 ± 660

all base policies for the current state as the average over multiple trajectories of the base policy. The average must be computed over a large number of trajectories of sufficient length for parallel rollout to return a better policy.

Limited Discrepancy Forward Search Sparse Sampling (LD-FSSS). LD-FSSS is a version of FSSS [59] with a class of choice functions called Limited Discrepancy Choice Functions (LDCF) [24]. A choice function defines the off-policy actions (discrepancies) available at the internal nodes of the search tree. LDCF limits the number of discrepancies along each root-to-leaf path and the depth up to which discrepancies are allowed in the search tree. It also restricts the set of discrepancies for every state to be non-increasing with depth. The base policy action is expanded at all the internal nodes of the search tree. When leaf nodes are initialized to base policy values and action values are computed with all possible successors with the true state-transition probabilities, LD-FSSS is guaranteed to return a policy better than the base policy.

The OPI algorithms mentioned above are all based on offline procedures that are theoretically guaranteed to return a policy better than the base policy. However, the ideal conditions on the number of sampled trajectories, lengths of trajectories, leaf initialization to base policy values, perfect state transitions of actions might not hold in practice. In that case, the online implementations can result in a policy that is worse than the given base policy.

4.5 OPI Baselines

Our baselines are variants of the policy rollout algorithm with a Q-value adjustment heuristic to deal with policy degradation to some extent. Let s_0 be the current state for which an action decision is to be made, A_{s_0} be the set of actions expanded at s_0, L be the lookahead horizon and π be the base policy. The base policy can be non-stationary. In order to keep the notation simple, we describe the baselines and heuristic with a deterministic, stationary policy $\pi = \{\mu_0, \mu_1, \ldots, \mu_{L-1}\}$, where $\mu_k = \mu$ for $k = 0, \ldots, L-1$ and $\mu : S \to A$. We use $\pi(s)$ instead of $\mu(s)$ for the base policy action at state s.

Baseline 1: MC Policy Rollout

Our first baseline is a version of the Monte Carlo (MC) policy rollout algorithm [6], [56]. The Q-values of actions at s_0 are estimated from sampled trajectories without building a search tree. The Q-value estimate of an action is the average of the values of multiple base-policy trajectories starting with that action. If N is the number of trajectories, then

$$\hat{Q}^{\pi}_{L}(s_0, a) \;=\; \frac{1}{N} \sum_{i=1}^{N} \Bigg(R(s_0, a) \;+\; \sum_{k=1}^{L-1} R(s^i_k, \pi(s^i_k)) \Bigg),$$

where s^i_k is the k^{th} subsequent state of trajectory i and $s^i_1 \in \{s' \in S : P_{s_0 s'}(a) > 0\}$ and $s^i_{k+1} \in \{s' \in S : P_{s^i_k s^i_{k+1}}(\pi(s^i_k)) > 0\}$ for $0 < k < L$ and $0 < i < L$. We note that $\hat{Q}^{\pi}_{L}(s_0, a)$ is an unbiased estimate of $Q^{\pi}_{L}(s_0, a)$.

Baseline 2: DAG Policy Rollout

The second baseline builds a search DAG (Directed Acyclic Graph) to make better use of samples compared to the first baseline. The DAG will have s_0 at its root followed by a sequence of state-node layers $(S_1, S_2, \ldots, S_L)$. While computing the backup values of states in the DAG, every state in layer S_i is assumed to be connected to all the states in layer S_{i+1}. This baseline is also a version of the policy rollout algorithm, so off-policy actions are allowed only at the root, and only the base policy action is allowed at all other internal nodes of the DAG.

DAG Construction. We expand the base policy action $\pi(s_0)$ and one or more off-policy actions at the root node s_0 and generate b_0 successors for each action using the true state-transition probabilities. The b_0 successors generated for an action can have repeated states. We put together all the generated successors of all the actions and eliminate duplicates to form the subsequent state-node layer S_1, i.e.,

$$S_1 \; = \bigcup_{a \in A_{s_0}} GSucc(s_0, a, b_0),$$

where $GSucc(s_0, a, b_0)$ is the set of distinct successors of action a taken b_0 times at state s_0 such that $GSucc(s_0, a, b_0) \subseteq \{s' \in S : P_{s_0 s'}(a) > 0\}$ and $|GSucc(s_0, a, b_0)| \leq b_0$.

We then expand the base policy action for all the states in layer S_1 and generate b successors for each state using the true state-transition probabilities. Again, the b successors generated for a state can have repeated states. We put together all the generated successors of all the states and eliminate duplicates to form the subsequent state-node layer S_2. We follow the same process to create all subsequent state-node layers $S_3, \ldots, S_L$. Formally,

$$S_{k+1} \; = \bigcup_{s \in S_k} GSucc(s, \pi(s), b),$$

for $k = 1 \ldots, L-1$, where $GSucc(s, \pi(s), b)$ is the set of distinct successors of the base policy action $\pi(s)$ taken b times at state s such that $GSucc(s, \pi(s), b) \subseteq \{s' \in S : P_{ss'}(\pi(s)) > 0\}$ and $|GSucc(s, \pi(s), b)| \leq b$.

Q-Value Computation. We set the values of leaf nodes to zero. We estimate the value of each state node in layers S_1 through S_{L-1} as the immediate reward of the base policy action for the state plus a weighted average of the values of all the state nodes in the following layer. Let $\hat{V}_{L-k}(s)$ denote the value estimate of state s in layer S_k in the DAG. Then $\hat{V}_0(s) = 0$ for all $s \in S_L$ and

$$\hat{V}_{L-k}(s) = R(s, \pi(s)) + \sum_{s' \in S_{k+1}} \hat{P}_{k,ss'}(\pi(s)) \cdot \hat{V}_{L-k-1}(s'),$$

where
$$\hat{P}_{k,ss'}(a) = \frac{P_{ss'}(a)}{\sum_{s' \in S_{k+1}} P_{ss'}(a)}.$$

The L steps-to-go Q-value of an action at the root node s_0 is the immediate reward of the action plus a weighted average of the values of its successors in layer S_1, i.e.,

$$\hat{Q}^{\pi}_{L}(s_0, a) \;=\; R(s_0, a) \;+\; \sum_{s' \in S_1} \hat{P}_{0,s_0 s'}(a) \cdot \hat{V}_{L-1}(s').$$

We note that $\hat{Q}^{\pi}_{L}(s_0, a)$ computed using normalized weights can be a biased estimate of $Q^{\pi}_{L}(s_0, a)$.

The Q-value Adjustment Heuristic

The purpose of the Q-value adjustment heuristic is to make it harder for off-policy actions to qualify as better actions in the current state. The OPI algorithm will then be conservative in switching to off-policy actions. We achieve this by computing an error margin for the Q-value estimate of each action at s_0. We then increase (decrease) the Q-value estimate of the base policy action (off-policy actions) by their respective error margins.

Let $\epsilon^{\pi}(s_0, a)$ denote the error margin for action a. We have two formulas for the error margins and hence two different heuristics, namely, the *C-Heuristic* and the *PC-Heuristic.*

- **C-Heuristic.** The error margin is a fraction of the absolute Q-value estimate of the action, i.e.,
$$\epsilon^{\pi}(s_0, a) \;=\; C \cdot |\hat{Q}^{\pi}_{L}(s_0, a)|,$$
where $C \in [0, 1]$ is a parameter.

- **PC-Heuristic.** The error margin has an additional state-action dependent factor equal to the total probability of next-states not covered while generating successors and hence not used in estimating the Q-value, i.e.,
$$\epsilon^{\pi}(s_0, a) \;=\; D(s_0, a) \cdot C \cdot |\hat{Q}^{\pi}_{L}(s_0, a)|,$$

where $C \in [0,1]$ and $D(s_0, a) = 1 - \sum_{s' \in S_1} P_{s_0 s'}(a)$.

The adjusted Q-value estimate of action a at s_0 is then

$$\tilde{Q}_L^{\pi}(s_0, a) \ = \ \begin{cases} \hat{Q}_L^{\pi}(s_0, a) \ + \ \epsilon^{\pi}(s_0, a), & \text{if } a = \pi(s_0), \\ \hat{Q}_L^{\pi}(s_0, a) \ - \ \epsilon^{\pi}(s_0, a), & \text{if } a \neq \pi(s_0). \end{cases}$$

Both MC policy rollout and DAG policy rollout return an action $\hat{a}$ that maximizes the adjusted Q-value estimates for s_0, i.e.,

$$\hat{a} \ \in \ \arg\max_{a \in A_s} \ \tilde{Q}_L^{\pi}(s_0, a).$$

4.6 Benchmarks

Our initial OPI benchmark set consists of the following 5 domains from the past International Probabilistic Planning Competitions (IPPC): (1) Sysadmin, (2) Game of Life, (3) Tamarisk, (4) Skill Teaching and (5) Wildfire. Each domain comes with a standard set of 10 problems of varying sizes and difficulty levels. Further details on the IPPC can be found at `http://www.icaps-conference.org/index.php/Main/Competitions`. Both the domains and problems are described using RDDL [49].

Sysadmin. This domain is about keeping as many computers up as possible in a computer network. The state of a computer is affected by the states of computers connected to it plus an external random factor. Actions are to reboot computers to bring them up. The immediate reward of a state-action pair is the number of computers running minus the action cost. The state space is factored with binary state variables for the computers in the network. The size of the state space ranges from 2^{10} to 2^{50} for the 10 problems.

Game of Life. This is a grid-based domain with cells in the grid either alive or dead. The goal is to have as many cells alive as possible. The state of a cell is affected by the states of the cells around it combined with an external random factor. Actions are to set cells to bring them alive. The immediate reward of a state-action pair is the number of cells alive minus the action cost. The size of the state space ranges from 2^9 to 2^{30} for the 10 problems.

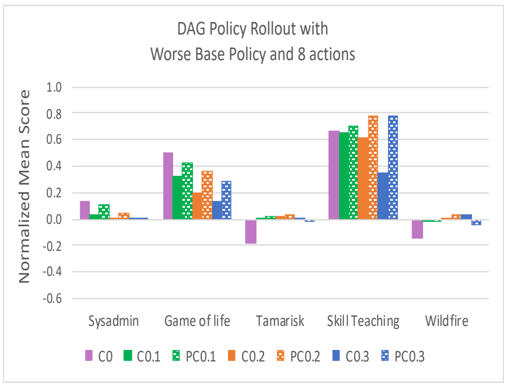
Table 4.2: WTL Scores for DAG Policy Rollout

	C0			**C0.1**			**PC0.1**			**C0.2**			**PC0.2**			**C0.3**			**PC0.3**		
Bad Base Policy	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L
Sysadmin	8	1	1	2	8	0	3	7	0	0	10	0	2	8	0	0	10	0	1	9	0
Game of life	10	0	0	8	2	0	9	1	0	5	5	0	7	3	0	2	8	0	5	5	0
Tamarisk	0	2	8	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0
Skill Teaching	5	5	0	3	7	0	5	5	0	3	7	0	6	4	0	3	7	0	6	4	0
Wildfire	0	6	4	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0
	C0			**C0.1**			**PC0.1**			**C0.2**			**PC0.2**			**C0.3**			**PC0.3**		
Good Base Policy	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L
Sysadmin	0	2	8	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0
Game of life	3	7	0	2	8	0	4	6	0	0	10	0	2	8	0	0	10	0	2	8	0
Tamarisk	0	0	10	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0
Skill Teaching	0	10	0	1	9	0	0	10	0	0	10	0	1	9	0	0	10	0	0	10	0
Wildfire	0	9	1	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0

Table 4.3: WTL Scores for MC Policy Rollout

	C0			**C0.1**			**PC0.1**			**C0.2**			**PC0.2**			**C0.3**			**PC0.3**		
Bad Base Policy	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L
Sysadmin	7	3	0	2	8	0	3	7	0	0	10	0	1	9	0	0	10	0	1	9	0
Game of life	8	2	0	6	4	0	8	2	0	2	8	0	6	4	0	1	9	0	6	4	0
Tamarisk	0	0	10	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0
Skill Teaching	5	5	0	6	4	0	5	5	0	3	7	0	5	5	0	3	7	0	5	5	0
Wildfire	0	3	7	0	10	0	0	9	1	0	10	0	0	10	0	0	10	0	0	10	0
	C0			**C0.1**			**PC0.1**			**C0.2**			**PC0.2**			**C0.3**			**PC0.3**		
Good Base Policy	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L	W	T	L
Sysadmin	0	0	10	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0
Game of life	1	6	3	0	10	0	2	8	0	0	10	0	1	9	0	0	10	0	1	9	0
Tamarisk	0	0	10	0	9	1	0	9	1	0	10	0	0	10	0	0	10	0	0	10	0
Skill Teaching	0	9	1	0	10	0	0	10	0	0	10	0	0	10	0	0	10	0	1	9	0
Wildfire	0	4	6	0	10	0	0	9	1	0	10	0	0	10	0	0	10	0	0	10	0

Tamarisk. This domain is about eradicating an invasive plant species called tamarisk to promote a native plant species in a river network. The region is divided into reaches each consisting of a certain number of slots. Tamarisk can spread from a reach to an adjacent (downstream) reach. Actions are to eradicate tamarisk or restore native species in reaches. The immediate reward of a state-action pair is a penalty for the number of slots invaded by and vulnerable to tamarisk plus the action cost. The state space is factored with two state variables per slot indicating the presence of tamarisk or native species. The size of the state space can range from 2^{16} to 2^{48} for the 10 problems.



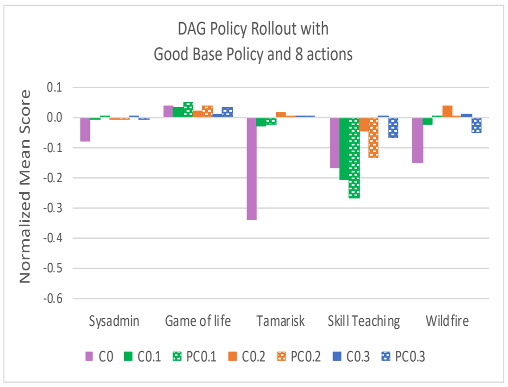
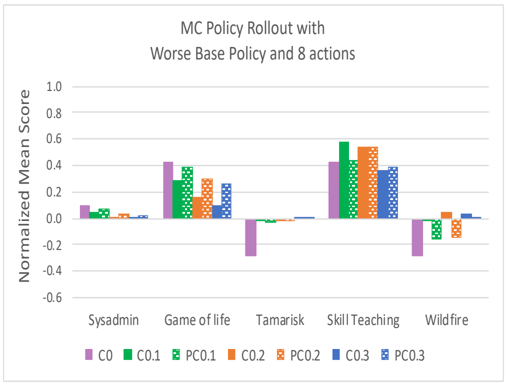


Figure 4.1: Normalized Mean Scores for DAG Policy Rollout



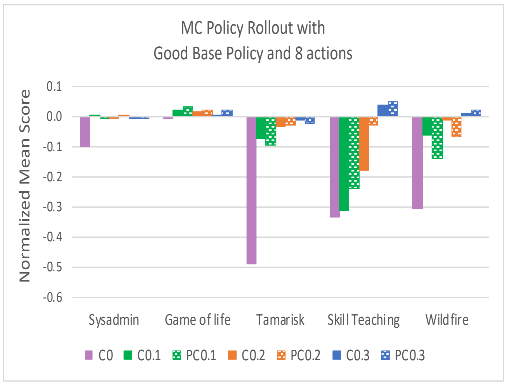


Figure 4.2: Normalized Mean Scores for MC Policy Rollout

Skill Teaching. This domain is about teaching a student a given set of skills via hints and questions. There are prerequisites for some of the skills. A student can attain medium proficiency in a skill if they know all the prerequisites. A student can attain high proficiency if they have medium proficiency in that skill and answer questions correctly. Answering a question wrong can decrease the proficiency level. Actions are to give hints or ask questions in skills. The immediate reward is a bonus for high proficiency and a penalty for medium proficiency in skills. The state space is factored with six state variables per skill. The size of the state space can range from 2^{12} to 2^{48} for the 10 problems.

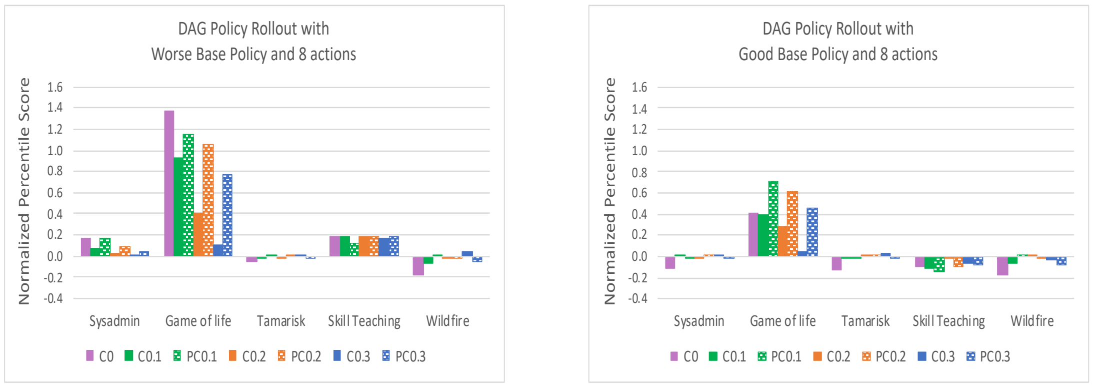


Figure 4.3: Normalized Percentile Scores for DAG Policy Rollout

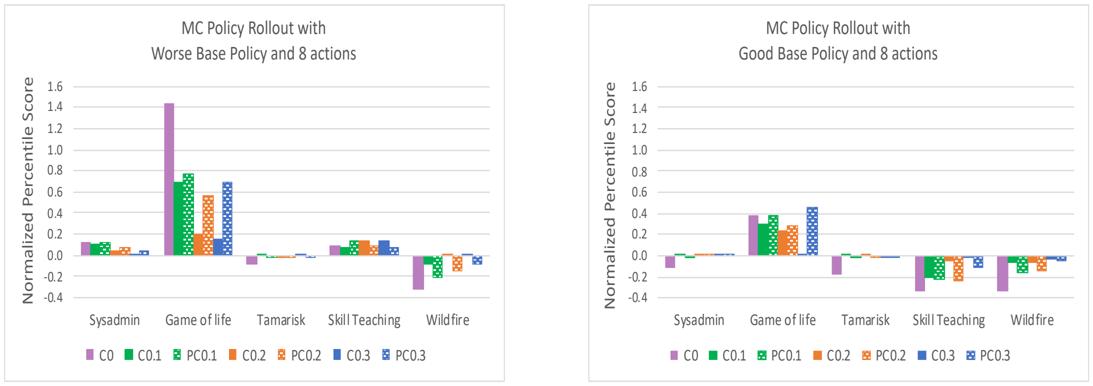


Figure 4.4: Normalized Percentile Scores for MC Policy Rollout

Wildfire. This is about controlling the spread of fire in a region modeled as a grid. A few cells are marked as targets that need to be protected from fire. A grid cell with fuel in it is more likely to burn if many of its neighbors are burning and a burning cell continues to burn until the fire is extinguished. Actions are to remove fuel from cells or put out fire in cells. The immediate reward for a state-action pair is a penalty for burned out or burning cells plus an action cost. The penalty is high for the target cells. The state space is factored with two state variables per cell indicating the presence of fuel and fire. The size of the state space can range from 2^{18} to 2^{72} for the 10 problems.

Base Policies

We have two base policies for each domain making a total of 10 base policies. For each domain, one of the base policies is of high quality, while the other is of relatively poor quality. Table 4.1 shows the performance of the two base policies. The numbers denote the average finite horizon sum of rewards for 100 evaluation runs with 95% confidence intervals. For the domains tamarisk and wildfire, the two policies are of roughly the same quality.

All our base policies are Neural Networks (NNs) taking a factored state as input and returning a probability distribution over actions as output. The action with the highest probability is taken as the base policy action for the state. We have used two distinct NN architectures for each domain: a linear architecture and one with 3 hidden layers and sparse connections defined by state-variable transitions. The linear architecture performs best for sysadmin and skill teaching, while the non-linear architecture performs best for game of life, tamarisk and wildfire. We have adopted the architecture, dataset, and training method for the NN base policies from Issakkimuthu et al. [23].

Evaluation Metrics

We estimate the expected finite horizon sum of rewards of the base policy and the OPI policy by taking the average over 100 evaluation runs. In order to assess OPI policies for performance degradation, we propose the following 3 evaluation metrics.

1. **Win-Tie-Loss (WTL) Count.** For each domain, we compute a triple of integers that add up to 10 (the number of problems in the domain). The components W, T and L stand for the number of wins, ties, and losses achieved by OPI against the base policies. The win, tie, or loss outcome for a problem is defined in terms of the standard 95% Confidence Intervals (CIs) around the mean performance of the base policy and the OPI policy. If the CI of OPI is totally above the CI of the base policy, then the outcome is a win for OPI. If the CIs overlap, then the outcome is a tie. Otherwise it is a loss. Wins are hard to achieve because the condition for wins is very strict. The WTL count is a simple and natural measure of the performance of OPI algorithms. OPI algorithms with losses can be considered unreliable for the domain. OPI algorithms with wider CIs will have many ties and no losses,

which might look fine under the WTL criterion. This drawback can be addressed by using the following metric along with WTL counts.

2. **Normalized Mean Score (NMS).** For each domain, we compute a single real number that indicates OPI performance. The NM score of a domain is the average of the NM scores of the 10 problems. The NM score for problem k is defined in terms of the mean performance of the base policy $\nu(k)$ and the mean performance of the OPI policy $\nu'(k)$ for that problem:

$$NMS(k) = \frac{\nu'(k) - \nu(k)}{|\nu(k)|}.$$

   A positive normalized mean score indicates better average improvement, a negative score indicates worse average degradation, and a zero score indicates equivalent average performance of OPI for the domain.

3. **Normalized Percentile Score (NPS).** In addition to NMS, another measure - the average of the bottom $\alpha\%$ of the 100 evaluation runs - might be useful to ensure that OPI does not crash badly when the base policy does not. The NP score of a domain is the average of the NP scores of the 10 problems. The NP score for problem k is defined in terms of the average of the bottom $\alpha\%$ of evaluation runs of the base policy $\zeta_\alpha(k)$ and the average of the bottom $\alpha\%$ of evaluation runs of the OPI policy $\zeta'_\alpha(k)$ for that problem:

$$NPS_\alpha(k) = \frac{\zeta'_\alpha(k) - \zeta_\alpha(k)}{|\zeta_\alpha(k)|}.$$

   Positive, negative, and zero NP scores indicate improved, worse, and equivalent low end average of OPI. NPS is analogous to the notion of Conditional Value at Risk (CVaR) in statistics.

4.7 Experiments

Here we provide an example of our evaluation procedure for the DAG and MC OPI baselines. The primary goal is to demonstrate the evaluation methodology and show that the baselines are strong for both the cases when transition probability information is not available (MC with C-Heuristic) and when the transition probability is available (DAG and MC with PC-Heuristic). Our implementation can be found here: `https://bitbucket.org/eshkim/ld-fsss/src/master/`.

Setup. The lookahead depth is 4 for both DAG and MC policy rollout. Leaf nodes are set to zero. The number of root actions is 8 or the number of actions applicable whichever is less. These are the top 8 actions according to the base policy probabilities and therefore include the base policy action. The number of successors generated for a state-action pair in the case of DAG policy rollout is 3, i.e., $b_0 = b = 3$. In order to compare DAG and MC policy rollout results, we first run DAG policy rollout for the current state and then run MC policy rollout for the same amount of time. The number of rollout trajectories in MC policy rollout can therefore vary from state to state.

The parameter C of the Q-value adjustment heuristic takes values from the set $\{0.0,\ 0.1,\ 0.2,\ 0.3\}$. For MC policy rollout with the PC-Heuristic, we record the successors of the root state s_0 for every action to compute $D(s_0, a)$ using the true state-transition probabilities. In our experiments, the planning horizon is 40, and actions are limited to those with at most one action bit set. Our experiments were run on an HPC cluster with the RDDLSim library available at `https://github.com/ssanner/rddlsim` for evaluation.

Notation. In all the tables and charts, the labels Cx and PCx have been used to denote OPI policies with different Q-value adjustment heuristics. Cx stands for the C-Heuristic with parameter C set to value x. PCx stands for the PC-Heuristic with parameter C set to value x. C0 corresponds to the OPI policy without Q-value adjustment, as it is equivalent to the first heuristic with parameter C set to 0.

WTL Results. Tables 4.2 and 4.3 show the WTL scores for DAG and MC policy rollout for the two base policies. The first major observation is that both DAG and MC rollout without Q-value adjustment (C0) have many losses across the domains. This is particularly the case for the higher-quality base policy, which is frequently degraded.

Next, we consider the influence of C on performance. We see, as expected, that as C increases the number of losses decreases and the wins tend to increase. In particular, for $C > 0$ there are only a very small number of losses and no losses for $C \geq 0.2$. At the same time we see that in 3 of the domains there are a significant number of wins for the lower-performing base policy even for the largest value of $C = 0.3$. There are zero wins for the high-performing base policy, which is due to the strict conditions under which we judge a win (non-overlapping confidence intervals). We will see later, however, that even in these cases there is improvement when considering the expected values.

Next, comparing the results for Cx versus PCx, we see that including the probability of states "not covered" typically results in more wins without a substantial increase (or any increase) in losses. This, shows that the PCx heuristic is able to effectively modulate the Q-value adjustment on a per state-action basis to improve OPI performance. This shows that when transition probabilities or good estimates are available, PCx can be an effective approach to maintaining safety while improving performance.

Finally, we see that the DAG policy rollout generally performs better that the MC approach in terms of total number of wins while not increasing the number of losses. This shows that there can be value in reuse of MC samples in a DAG structure and also exploiting the transition model within the DAG model compared to a pure MC approach. The design space of DAG structures used for value estimation is large and it is reasonable to expect that optimization of the structure could result in further improvement.

Overall, the results indicate that the DAG and MC approaches offer a strong parameterized space of baselines for safe OPI that span different WTL trade-offs for comparison to other approaches. In particular, MC offers a baseline that requires no information about transition probabilities, while DAG is able to exploit transition information when available. The results also indicate that with respect to number of wins, there is significant room to improve over the baselines for the higher-quality base policy.

Normalized Mean Scores. The bar charts in Figures 4.1 and 4.2 show the NMS for DAG and MC policy rollout for both the base policies. We first see that for both DAG and MC policy rollout, the scores are positive in sysadmin, game of life, and skill teaching for the bad base policy and somewhat positive in game of life for the good base policy. The scores are negative or close to zero everywhere else. The scores are particularly negative for the case of no Q-value adjustment (C0). Increasing the value of C largely mitigates the degradation, and including the "missing probability" into the adjustment tends to be better Finally, again, DAG policy rollout performs slightly better than MC policy rollout, which shows that the baselines are able to leverage the availability of transition probabilities for better performance.

Normalized Percentile Scores. The bar charts in Figures 4.3 and 4.4 show the NPS for the bottom 5% evaluation runs for DAG and MC policy rollout for both the base policies. These results are qualitatively similar to those for NMS. However, we do see that the NPS values tend to be higher than the corresponding NMS values. This indicates that the baseline OPI methods are generating more improvement or less degradation for the lower end of the performance profile (NPS) compared to the mean performance. In other words, these OPI baselines are suggesting they are particularly effective at improving worst case performance.

4.8 Summary

In this work, we have drawn attention to the important practical issue of policy degradation in OPI. We have proposed benchmarks and evaluation metrics for OPI with the goal of improving our understanding of the empirical performance of OPI algorithms. We have also presented OPI baselines with a heuristic to deal with policy degradation. The baselines form a class of methods that can use transition probabilities if available or only utilize samples. The parameterized baselines are demonstrated to span the trade-off space of OPI performance, making them useful points for future comparison. Further, the baselines demonstrate benefit from using transition probability information, making them useful for comparing to evaluation settings with and without that information. The DAG-based baseline makes better use of samples by constructing a search DAG instead of discarding sampled trajectories. It will be interesting to consider future extensions that continue to search off-policy actions within the DAG rather than only at the root. Overall, we hope that this work helps set the stage for more work on safe OPI algorithms backed by solid evaluations and comparisons to strong baselines.

Chapter 5: Theoretical Results Related to Aggregation in MDPs

5.1 Abstract

Sequential decision-making problems under uncertainty are often modeled using the framework of Markov Decision Processes (MDPs). Many real-world problems give rise to MDPs with huge state and action spaces, which has led to a variety of proposed approximate solution techniques. Aggregation is one such technique, where MDP states and actions are grouped together to form a smaller aggregate problem. The smaller problem is solved in place of the original problem, and the aggregate solution is extended to the original problem. The quality of the aggregate solution depends on properties of the aggregation, which has led to a number of theoretical analyses of the sub-optimality of aggregation. An investigation of the current set of results, however, revealed that despite the apparent similarity of prior work on aggregation, the results are not always informed by or related to one another. This has made it difficult to understand the overall landscape of results in a unified way. The main contribution of this paper is to synthesize a more unified view of the sub-optimality guarantees provided by some widely used aggregation frameworks. In particular, the collection of results is put in the context of two prior aggregation frameworks, which leads to a more common analysis approach and, in some cases, improved sub-optimality bounds.

5.2 Introduction

Sequential decision-making problems under uncertainty are typically modeled using the framework of Markov Decision Processes (MDPs). Once modeled as an MDP with an objective, a sequential decision-making problem can be solved optimally using exact MDP solution techniques such as value iteration, policy iteration and linear programming. However, many real-world problems give rise to MDPs with huge state and action spaces, which can only be solved approximately in practice. Aggregation is a problem approximation technique, where states and actions of an MDP are grouped together to form a smaller aggregate problem. The aggregate problem is then solved in place of the original problem, and the aggregate solution is extended to the original problem. Aggregation will be useful when the aggregate solution is not too sub-optimal in the original MDP.

The aggregation approach to solving an MDP can be viewed as a sequence of three steps. The first step is to define the aggregate problem, which is usually another MDP with a relatively smaller state and action space. The second step is to solve the aggregate problem, for instance, by computing the optimal value function of the smaller MDP. The final step is to specify how the aggregate solution will be extended to the original MDP.

There are several aggregation schemes in the literature that differ in one or more of the three steps. Many of them come with theoretical bounds on the sub-optimality of the aggregation solution in the original MDP. Our objective in this synthesis is to present a unified view of the sub-optimality guarantees provided by aggregation schemes. We achieve this by identifying two prior aggregation frameworks that capture a collection of existing work. This synthesis provides a detailed presentation of sub-optimality results for those frameworks and expresses a number of other prior results as special cases of these frameworks.

5.3 Background

A Markov decision process M is a tuple $(S, \{A_s\}, P, R)$, where S is a finite set of states, A_s is the finite set of actions applicable in state s, P is the state-transition function where $P_{ss'}(a_s)$ denotes the probability of reaching state s' from state s on taking action a_s at state s, and R is the reward function where $R(s, a_s) \in [R_{min}, R_{max}]$ denotes the immediate reward for taking action a_s in state s. The objective is to maximize the expected discounted infinite-horizon sum of rewards with discount factor $\gamma \in (0, 1)$.

A deterministic policy π of an MDP is a mapping from states to actions. The Q-value function of a policy π gives the expected infinite-horizon discounted cumulative reward of executing an action a_s in state s and then following π thereafter, which is denoted by $Q^\pi(s, a_s)$. The Q-function of a policy is the unique solution to the following set of constraints:

$$Q^{\pi}(s, a_s) \;=\; R(s, a_s) + \gamma \sum_{s' \in S} P_{ss'}(a_s) \cdot Q^{\pi}(s', \pi(s')) \tag{5.1}$$

for all $a_s \in A_s$. Similarly, the value of a policy at a state, denoted $V^{\pi}(s)$, gives the expected infinite-horizon cumulative reward of executing π starting at s and satisfies the constraint $V^{\pi}(s) = Q^{\pi}(s, \pi(s))$ for all $s \in S$. The Bellman backup operator with respect to vector V restricted to policy π is

$$B_{\pi}[V](s) \;=\; R(s, \pi(s)) + \gamma \sum_{s' \in S} P_{ss'}(\pi(s)) \cdot V(s') \tag{5.2}$$

for all $s \in S$ and V^{π} is the fixed point of B_{π}, i.e., $V^{\pi} = B_{\pi}[V^{\pi}]$.

An optimal policy π^* for an MDP is a policy that has a Q-value or value that is as good as or better than any other policy across the full state-action space. The Q-function and value function of an optimal policy, denoted Q^* and V^* are unique and satisfy the Bellman equation:

$$Q^*(s, a_s) \;=\; R(s, a_s) + \gamma \sum_{s' \in S} P_{ss'}(a_s) \cdot \max_{a_{s'} \in A_{s'}} Q^*(s', a_{s'}) \tag{5.3}$$

for all $a_s \in A_s$ and $V^*(s) = \max_{a_s \in A_s} Q^*(s, a_s)$ for all $s \in S$.

It will be useful to have the following definition of Q-values with respect to a given vector V.

$$Q_V(s, a_s) \;=\; R(s, a_s) + \gamma \sum_{s' \in S} P_{ss'}(a_s) \cdot V(s') \tag{5.4}$$

noting that $Q^{\pi}(s, a_s) = Q_{V^{\pi}}(s, a_s)$ and $Q^*(s, a_s) = Q_{V^*}(s, a_s)$ for all $s \in S$ and $a_s \in A_s$. The optimal value function can be computed via the value iteration algorithm by iterating the Bellman backup operator starting from any value vector V,

$$B[V](s) \;=\; \max_{a_s \in A_s} R(s, a_s) + \gamma \sum_{s' \in S} P_{ss'}(a_s) \cdot V(s') \tag{5.5}$$

for all $s \in S$. V^* is the unique fixed point of this operator, i.e., $V^* = B[V^*]$.

In addition to value iteration, other well known approaches to solving MDPs include policy iteration and linear programming [42]. Each of these algorithms scales at least polynomially in the number of states and actions, which makes them computationally expensive or impractical for MDPs with enormous state and/or action spaces. The aggregation approach reduces the size of the MDP by aggregating states, and possibly actions, to create a smaller aggregate problem. In the following sections, we introduce two prior frameworks for aggregation that capture a collection of results in the literature.

5.4 The Whitt Framework

Whitt introduced this aggregation framework in his work *Approximations of Dynamic Programs I* [60]. In this framework, the aggregate problem is an abstract MDP. Whitt's framework is for general dynamic programs. We describe a version of it here for discrete MDPs with finite state and action spaces.

The Abstract MDP

The abstract MDP is $\bar{M} = (\bar{S}, \{\bar{A}_{\bar{s}}\}, \bar{P}, \bar{R})$ with state space $\bar{S}$, action space $\{\bar{A}_{\bar{s}}\}$ where $\bar{A}_{\bar{s}}$ denotes the set of actions applicable in state $\bar{s} \in \bar{S}$, state-transition function $\bar{P}$ where $\bar{P}_{\bar{s}\bar{s}'}(\bar{a})$ is the probability of reaching state $\bar{s}'$ from state $\bar{s}$ on taking action $\bar{a}_{\bar{s}}$ at state $\bar{s}$ and $\bar{R}$ is the reward function where $\bar{R}(\bar{s}, \bar{a}_{\bar{s}}) \in [R_{min}, R_{max}]$ is the immediate reward for taking action $\bar{a}_{\bar{s}}$ in state $\bar{s}$. The objective is to maximize the expected discounted infinite-horizon sum of rewards with discount factor $\gamma \in (0, 1)$.

Definition

The abstract MDP $\bar{M} = (\bar{S}, \{\bar{A}_{\bar{s}}\}, \bar{P}, \bar{R})$ and the original MDP $M = \langle S, A, P, R\rangle$ are related by the following onto mappings

- $\phi : S \to \bar{S}$,
- $\psi = \{\psi_s : A_s \to \bar{A}_{\phi(s)},\ \forall s \in S\}$.

The onto mapping $\phi : S \to \bar{S}$ defines a partition of S, as it maps every state in S to an abstract state in $\bar{S}$. An abstract state $\bar{s} \in \bar{S}$ represents a block of the partition defined by ϕ, i.e., $\bar{s} = \{s \in S : \ \phi(s) = \bar{s}\}$. Similarly, the onto mapping $\psi_s : A_s \rightarrow \bar{A}_{\bar{s}}$ defines a partition of A_s as it maps every action $a_s \in A_s$ to an action in $\bar{A}_{\phi(s)}$. An abstract action $\bar{a}_{\phi(s)} \in \bar{A}_{\phi(s)}$ represents a block of this partition. Together, the mappings ϕ and ψ define a partition of the set of all admissible state-action pairs (s, a_s) of the original MDP. An abstract state-action pair $(\bar{s}, \bar{a}_{\bar{s}})$ represents a block of this partition, i.e., $(\bar{s}, \bar{a}_{\bar{s}}) = \{(s, a_s) : \phi(s) = \bar{s}, \ \psi_s(a_s) = \bar{a}_{\bar{s}}\}$ for all $\bar{s} \in \bar{S}$ and $\bar{a}_{\bar{s}} \in \bar{A}_{\bar{s}}$.

The state-transition and reward functions of the abstract MDP are defined in terms of the state-transition and reward functions of the original MDP. For this purpose, a probability distribution $W_{\bar{s}\bar{a}_{\bar{s}}}$ is defined over the set of original state-action pairs mapped to every abstract state-action pair $(\bar{s}, \bar{a}_{\bar{s}})$, i.e., $W_{\bar{s}\bar{a}_{\bar{s}}}$ is such that $\sum_{(s,a_s)\in(\bar{s},\bar{a}_{\bar{s}})} W_{\bar{s}\bar{a}_{\bar{s}}}(s, a_s) = 1$ for all $\bar{s} \in \bar{S}$ and $\bar{a}_{\bar{s}} \in \bar{A}_{\bar{s}}$.

State-Transition Function. The state-transition probability of an original state-action pair (s, a_s) to an abstract state $\bar{s}'$ is the sum of the transition probabilities from state s on action a_s to every original state s' in the abstract state $\bar{s}'$. The state-transition probability of an abstract state-action pair $(\bar{s}, \bar{a}_{\bar{s}})$ to an abstract state $\bar{s}'$ is a weighted average of the state-transition probabilities of all the original state-action pairs in $(\bar{s}, \bar{a}_{\bar{s}})$ to the abstract state $\bar{s}'$, i.e.,

$$\bar{P}_{\bar{s}\bar{s}'}(\bar{a}_{\bar{s}}) \ = \sum_{(s,a_s)\in(\bar{s},\bar{a}_{\bar{s}})} W_{\bar{s}\bar{a}_{\bar{s}}}(s, a_s) \sum_{s'\in\bar{s}'} P_{ss'}(a_s) \tag{5.6}$$

for all $\bar{s} \in \bar{S}$ and $\bar{a}_{\bar{s}} \in \bar{A}_{\bar{s}}$.

Reward Function. The immediate reward of an abstract state-action pair $(\bar{s}, \bar{a}_{\bar{s}})$ is a weighted average of the immediate rewards of all the original state-action pairs mapped to $(\bar{s}, \bar{a}_{\bar{s}})$, i.e.,

$$\bar{R}(\bar{s}, \bar{a}_{\bar{s}}) \ = \sum_{(s,a_s)\in(\bar{s},\bar{a}_{\bar{s}})} W_{\bar{s}\bar{a}_{\bar{s}}}(s, a_s) \cdot R(s, a_s) \tag{5.7}$$

for all $\bar{s} \in \bar{S}$ and $\bar{a}_{\bar{s}} \in \bar{A}_{\bar{s}}$.

Solution

Since $\bar{M}$ is just another MDP, the definitions of Q-functions, value functions and Bellman backup operator in equations 5.1 to 5.5 apply to $\bar{M}$ as well. We use the bar notation for the abstract MDP, that is, the Q-function and value function of the abstract MDP $\bar{M}$ with respect to an abstract policy $\bar{\pi}$ are denoted $\bar{Q}^{\bar{\pi}}(\bar{s}, \bar{a}_{\bar{s}})$ and $\bar{V}^{\bar{\pi}}(\bar{s})$ respectively for all $\bar{s} \in \bar{S}$ and $\bar{a}_{\bar{s}} \in \bar{A}_{\bar{s}}$.

Solution Extension

A value vector of the abstract MDP is extended to the original MDP by assigning the value of each abstract state to all the original MDP states mapped to it. The extension of $\bar{V}$, denoted $X[\bar{V}]$, is therefore

$$X[\bar{V}](s) = \bar{V}(\phi(s)) \tag{5.8}$$

for all $s \in S$. Similarly, a Q-vector of the abstract MDP is extended to the original MDP by assigning the value of each abstract state-action pair to all the original state-action pairs mapped to it. The extension of $\bar{Q}$, denoted $X[\bar{Q}]$, is therefore

$$X[\bar{Q}](s, a_s) = \bar{Q}(\phi(s), \psi_s(a_s)) \tag{5.9}$$

for all $s \in S$ and $a_s \in A_s$. A deterministic policy of the abstract MDP is extended to a deterministic policy for the original MDP by choosing for each state $s \in S$ a representative action for each abstract action applicable at $\phi(s)$. This is necessary because multiple actions of state s can be mapped to an abstract action for $\phi(s)$.

Let $\tilde{a}_s[\bar{a}_{\phi(s)}]$ be the representative action of the set $\{a_s \in A_s : \psi_s(a_s) = \bar{a}_{\phi(s)}\}$. The extension of $\bar{\pi}$ to the original MDP, denoted $X[\bar{\pi}]$, is

$$X[\bar{\pi}](s) = \tilde{a}_s[\bar{\pi}(\phi(s))] \tag{5.10}$$

for all $s \in S$.

Main Results

Whitt's framework in his paper [60] is for general dynamic programs. In this section, we present the main results of the framework adapted to discrete MDPs with finite state and action spaces. In the theorems, we use the following definitions from equations (6.1), (3.1) and (4.4) in Whitt [60]. The first two definitions compare the immediate rewards and state-transition probabilities of the original and abstract MDPs.

Definition 1. *K_r is a measure of the worst-case difference between the immediate reward of a state-action pair in M and $\bar{M}$. More precisely, K_r is the maximum absolute difference between the immediate reward of a state-action pair in M and that of the corresponding abstract state-action pair in $\bar{M}$, i.e.,*

$$K_r = \max_{s \in S,\, a_s \in A_s} \left| R(s, a_s) - \bar{R}(\phi(s), \psi_s(a_s)) \right| .$$

Definition 2. *K_q is a measure of the worst-case difference between the transition probabilities of a state-action pair in M and $\bar{M}$. More precisely, K_q is the maximum one-norm distance between the state-transition distribution of a state-action pair in M over the abstract state space $\bar{S}$ and that of the corresponding abstract state-action pair in $\bar{M}$, i.e.,*

$$K_q = \max_{s \in S,\, a_s \in A_s} \sum_{\bar{s}' \in \bar{S}} \left| \sum_{s' \in \bar{s}'} P_{ss'}(a_s) - \bar{P}_{\phi(s)\bar{s}'}(\psi_s(a_s)) \right| .$$

Definition 3. *For value vector V of the original MDP, $\delta(V)$ is the difference between the maximum and minimum components of V, i.e.,*

$$\delta(V) = \max_{s \in S} V(s) - \min_{s \in S} V(s).$$

The definition applies for a value vector $\bar{V}$ of the abstract MDP as well, i.e.,

$$\delta(\bar{V}) = \delta(X[\bar{V}]).$$

The following definition compares the one-step backup values of the original and abstract MDPs with respect to a given vector.

Definition 4. *$K(\bar{V})$ is a measure of the worst-case difference between the one-step backup value of a state-action pair in M and $\bar{M}$. For value vector $\bar{V}$ of the abstract MDP, $K(\bar{V})$ is the maximum absolute difference between the Q-value of a state-action pair in M with respect to $X[\bar{V}]$ and that of the corresponding abstract state-action pair with respect to $\bar{V}$, i.e.,*

$$K(\bar{V}) \;=\; \max_{s \in S,\, a_s \in A_s} \left| Q_{X[\bar{V}]}(s, a_s) - \bar{Q}_{\bar{V}}(\phi(s), \psi_s(a_s)) \right| .$$

Definition 5. *For value vector V of the original MDP, $L(V)$ is the maximum absolute difference between the Q-values of two state-action pairs of M mapped to the same abstract state-action pair, i.e.,*

$$L(V) \;=\; \max_{\bar{s} \in \bar{S},\, \bar{a}_{\bar{s}} \in \bar{A}_{\bar{s}}} \quad \max_{(s, a_s),\, (\hat{s}, a_{\hat{s}}) \,\in\, (\bar{s}, \bar{a}_{\bar{s}})} \quad \left| Q_V(s, a_s) - Q_V(\hat{s}, a_{\hat{s}}) \right| .$$

For value vector $\bar{V}$ of the abstract MDP, $L(\bar{V}) = L(X[\bar{V}])$.

Lemma 5. *For value vector $\bar{V}$ of the abstract MDP, $L(\bar{V}) \;\geq\; K(\bar{V})$.*

Proof. A proof can be found in the appendix (section 7.2). □

Theorem 6 below gives a bound on $K(\bar{V})$ in terms of K_r, K_q and $\delta(\bar{V})$. This result has been adapted from theorem 6.1(b) in Whitt [60] and it will be used in some of the sub-optimality results.

Theorem 6. *For value vector $\bar{V}$ of the abstract MDP $\bar{M}$,*

$$K(\bar{V}) \;\leq\; K_r + \gamma \cdot \frac{K_q}{2} \cdot \delta(\bar{V}).$$

Proof. The bound follows from the definitions of $K(\bar{V})$, K_r, K_q and $\delta(\bar{V})$. A proof based on the proof of theorem 6.1(b) in Whitt [60] is given in the appendix (section 7.2). $\square$

Theorem 7 below gives a bound on the max-norm distance between the extended Q-vector of an abstract policy $\bar{\pi}$ and the Q-vector of the extended abstract policy $X[\bar{\pi}]$, i.e., $\|X[\bar{Q}^{\bar{\pi}}] - Q^{X[\bar{\pi}]}\|_\infty$.

Theorem 7. *For any deterministic, stationary policy $\bar{\pi}$ of $\bar{M}$*

$$\|X[\bar{Q}^{\bar{\pi}}] - Q^{X[\bar{\pi}]}\|_\infty \;\leq\; \frac{1}{1-\gamma\left(1-\frac{K_q}{2}\right)}\;\left(K_r + \gamma\cdot\frac{K_q}{2}\cdot\frac{R_{max}-R_{min}}{1-\gamma}\right),$$

where $[R_{min}, R_{max}]$ is the reward function range for both the original and abstract MDPs.

Proof. A detailed and complete proof is given in the appendix (section 7.2). The proof structure is based on the proof of theorem 6.2(a) in Whitt [60]. $\square$

The following corollary gives a bound on the max-norm distance between the extended value vector of an abstract policy $\bar{\pi}$ and the value vector of the extended abstract policy $X[\bar{\pi}]$, i.e., $\|X[\bar{V}^{\bar{\pi}}] - V^{X[\bar{\pi}]}\|_\infty$. This is the result in theorem 6.2(a) from Whitt [60].

Corollary 2. *For any deterministic, stationary policy $\bar{\pi}$ of $\bar{M}$*

$$\|X[\bar{V}^{\bar{\pi}}] - V^{X[\bar{\pi}]}\|_\infty \;\leq\; \frac{1}{1-\gamma\left(1-\frac{K_q}{2}\right)}\;\left(K_r + \gamma\cdot\frac{K_q}{2}\cdot\frac{R_{max}-R_{min}}{1-\gamma}\right),$$

where $[R_{min}, R_{max}]$ is the reward function range of both the original and abstract MDPs.

Proof. A proof is given in the appendix (section 7.2). $\square$

Theorem 8 below gives a different bound on $\|X[\bar{V}^{\bar{\pi}}] - V^{X[\bar{\pi}]}\|_\infty$ in terms of $K(\bar{V}^{\bar{\pi}})$. This result has been adapted from theorem 3.2 in Whitt [60].

Theorem 8. *For any deterministic, stationary policy $\bar{\pi}$ of $\bar{M}$*

$$\|X[\bar{V}^{\bar{\pi}}] - V^{X[\bar{\pi}]}\|_\infty \;\leq\; \frac{1}{1-\gamma} \cdot K(\bar{V}^{\bar{\pi}}).$$

Proof. A proof is given in the appendix (section 7.2). □

Theorem 9 below gives a bound on the max-norm distance between the extended optimal value vector of the abstract MDP and the optimal value vector of the original MDP, i.e., $\|V^* - X[\bar{V}^*]\|_\infty$. This result has been adapted from theorem 3.1 in Whitt [60].

Theorem 9. *For the optimal value vectors V^* and $\bar{V}^*$ of the original MDP M and the abstract MDP $\bar{M}$, we have*

$$\|V^* - X[\bar{V}^*]\|_\infty \;\leq\; \frac{1}{1-\gamma} \cdot K(\bar{V}^*).$$

Proof. The proof follows from a bound on the max-norm distance between $X[\bar{V}^*]$ and the Bellman backup vector with respect to $B[X[\bar{V}^*]]$. A complete proof is given in the appendix (section 7.2). □

Theorem 10 below gives a bound on the max-norm distance between the value vector of the optimal policy of the abstract MDP extended to the original MDP and the optimal value vector of the original MDP, i.e., $\|V^* - V^{X[\bar{\pi}^*]}\|_\infty$. This result has been adapted from the corollary of Lemma 3.1 in Whitt [60].

Theorem 10. *For the optimal value vector V^* of the original MDP M and the optimal value vector $\bar{V}^*$ and optimal policy $\bar{\pi}^*$ of the abstract MDP $\bar{M}$, we have*

$$\|V^* - V^{X[\bar{\pi}^*]}\|_\infty \;\leq\; \frac{2}{1-\gamma} \cdot K(\bar{V}^*).$$

Proof. The result follows from theorems 8 and 9 above. A complete proof is given in the appendix (section 7.2). □

Special cases

We list several special cases of the Whitt abstraction framework in this section.

1. Simulation Lemmas

   Here the abstract MDP is an approximation of the original MDP with the same state and action spaces but different reward and state-transition functions.

   (a) Lemma 1 from Strehl et al. [55].

   The original MDP is $M = \langle S, A, P, R \rangle$ and the abstract MDP is $\bar{M} = \langle S, A, \bar{P}, \bar{R} \rangle$. The set of applicable actions is the same for all the states, i.e., $A_s = A$ for all $s \in S$. The reward function range for both M and $\bar{M}$ is $[0, R_{max}]$, i.e., $0 \leq R(s,a) \leq R_{max}$ and $0 \leq \bar{R}(s,a) \leq R_{max}$ for all $s \in S$ and $a \in A$. Since both M and $\bar{M}$ have the same state and action spaces, the expressions for K_r and K_q reduce to

$$\begin{aligned} K_r &= \max_{s \in S,\, a \in A} \left| R(s,a) - \bar{R}(s,a) \right|, \\ K_q &= \max_{s \in S,\, a \in A} \sum_{s' \in S} \left| P_{ss'}(a) - \bar{P}_{ss'}(a) \right|. \end{aligned}$$

   A policy of $\bar{M}$ is also a valid policy for M, since both M and $\bar{M}$ have the same set of policies. Lemma 1 from Strehl et al. [55] gives a bound on the max-norm distance between the Q-vector of a policy in M and $\bar{M}$. In our notation, their bound can be expressed as

$$\| Q^{\bar{\pi}} - \bar{Q}^{\bar{\pi}} \|_\infty \leq \frac{K_r + \gamma \cdot R_{max} \cdot K_q}{(1-\gamma)^2}.$$

   We can apply theorem 7 to improve this bound slightly to get

$$\begin{aligned} \| Q^{\bar{\pi}} - \bar{Q}^{\bar{\pi}} \|_\infty &= \| Q^{X[\bar{\pi}]} - X[\bar{Q}^{\bar{\pi}}] \|_\infty \\ &\leq \frac{1}{1 - \gamma\,(1 - \frac{K_q}{2})} \left(K_r + \gamma \cdot \frac{K_q}{2} \cdot \frac{R_{max}}{1-\gamma} \right). \end{aligned}$$

(b) Theorem 1 from Serban et al. [51].

The original MDP is $M = \langle S, A, P, R \rangle$ and the abstract MDP is $\bar{M} = \langle S, A, \bar{P}, R \rangle$. M and $\bar{M}$ have the same reward function with range $[0, R_{max}]$, i.e., $0 \leq R(s, a) \leq R_{max}$ for all $s \in S$ and $a \in A$. Since M and $\bar{M}$ have the same state and action spaces and reward function, the expressions for K_r, K_q and $K(\bar{V})$ reduce to

$$\begin{aligned} K_r &= 0, \\ K_q &= \max_{s \in S,\, a \in A} \sum_{s' \in S} \left| P_{ss'}(a) - \bar{P}_{ss'}(a) \right|, \\ K(\bar{V}) &= \max_{s \in S,\, a \in A} \left| Q_{\bar{V}}(s, a) - \bar{Q}_{\bar{V}}(s, a) \right|. \end{aligned}$$

The result in theorem 1 from Serban et al. [51] is in terms of the KL divergence between the state-transition distributions of P and $\bar{P}$. To state the result in our notation, we need the following definitions. Let $P_{s.}(a)$ and $\bar{P}_{s.}(a)$ denote the state-transition distributions of P and $\bar{P}$ respectively over S for the state-action pair (s, a). The KL divergence between $P_{s.}(a)$ and $\bar{P}_{s.}(a)$ is

$$D_{KL}(P_{s.}(a) \mid\mid \bar{P}_{s.}(a)) = \sum_{s' \in S} P_{ss'}(a) \, \log \frac{P_{ss'}(a)}{\bar{P}_{ss'}(a)}.$$

Let K_{kl} be the maximum absolute square root of the KL divergence between the state-transition distributions of P and $\bar{P}$ over all state-action pairs, i.e.,

$$K_{kl} = \max_{s \in S,\, a \in A} \left| \sqrt{D_{KL}(P_{s.}(a) \mid\mid \bar{P}_{s.}(a))} \right|.$$

From Serban et al.'s bound [51] on the L1 distance between $P_{s.}(a)$ and $\bar{P}_{s.}(a)$ using Pinsker's inequality, i.e.,

$$\sum_{s' \in S} \left| P_{ss'}(a) - \bar{P}_{ss'}(a) \right| \leq \sqrt{2 \, D_{KL}(P_s(a) \mid\mid \bar{P}_s(a))},$$

we get $K_q \leq \sqrt{2} \, K_{kl}$.

Theorem 1 from Serban et al. [51] gives a bound on the max-norm distance between the optimal Q-vectors of M and $\bar{M}$. In our notation, this bound can be expressed as

$$\|Q^* - \bar{Q}^*\|_\infty \;\leq\; \frac{\gamma \cdot R_{max} \cdot \sqrt{2} \cdot K_{kl}}{(1-\gamma)^2}.$$

This bound can be slightly improved using theorem 7 to get

$$\|Q^* - \bar{Q}^*\|_\infty \;\leq\; \frac{1}{1-\gamma\,(1-\frac{K_q}{2})} \left(\gamma \cdot \frac{R_{max}}{1-\gamma} \cdot \frac{K_{kl}}{\sqrt{2}}\right).$$

Proof.

$$Q^* - \bar{Q}^* \;=\; Q^{\pi^*} - \bar{Q}^{\pi^*} \;+\; \bar{Q}^{\pi^*} - \bar{Q}^* \;\leq\; Q^{\pi^*} - \bar{Q}^{\pi^*} \;\leq\; \|Q^{\pi^*} - \bar{Q}^{\pi^*}\|_\infty.$$

$$\bar{Q}^* - Q^* \;=\; \bar{Q}^{\bar{\pi}^*} - Q^{\bar{\pi}^*} \;+\; Q^{\bar{\pi}^*} - Q^{\pi^*} \leq \bar{Q}^{\bar{\pi}^*} - Q^{\bar{\pi}^*} \leq \|\bar{Q}^{\bar{\pi}^*} - Q^{\bar{\pi}^*}\|_\infty.$$

By theorem 7, we have

$$\begin{aligned}\|Q^* - \bar{Q}^*\|_\infty \;&\leq\; \frac{1}{1-\gamma\,(1-\frac{K_q}{2})} \left(\gamma \cdot \frac{R_{max}}{1-\gamma} \cdot \frac{K_q}{2}\right) \\ &\leq\; \frac{1}{1-\gamma\,(1-\frac{K_q}{2})} \left(\gamma \cdot \frac{R_{max}}{1-\gamma} \cdot \frac{K_{kl}}{\sqrt{2}}\right), \quad \text{since } K_q \leq \sqrt{2}\, K_{kl}.\end{aligned}$$

□

2. Approximate MDP Homomorphism (Ravindran et al. [46]).

The original MDP is $M = (S, \{A_s\}, P, R)$ with optimal value function V^* and the abstract MDP is $\bar{M} = (\bar{S}, \{\bar{A}_{\bar{s}}\}, \bar{P}, \bar{R})$ with optimal value function $\bar{V}^*$ and optimal policy $\bar{\pi}^*$. The main result in section 4.1 of Ravindran et al. [46] gives a bound on the sub-optimality of the stochastic extension $\mathcal{X}[\bar{\pi}^*]$ of an optimal policy $\bar{\pi}^*$ of $\bar{M}$

to M. In our notation, this result can be expressed as

$$V^* - V^{\mathcal{X}(\bar{\pi}^*)} \leq \frac{2}{1-\gamma} \cdot K(\bar{V}^*),$$

where

$$\mathcal{X}[\bar{\pi}^*](s, a_s) = \frac{\bar{\pi}^*(\phi(s), \psi_s(a_s))}{|\{a' \in A_s : \psi_s(a') = \psi_s(a_s)\}|}$$

for all $s \in S$ and $a_s \in A_s$. The result follows from theorem 10 since theorem 8 holds for the stochastic extension $\mathcal{X}[\bar{\pi}^*]$ of $\bar{\pi}^*$ as shown in Corollary 3 in the appendix (section 7.2).

3. Model Similarity based State Abstraction (Abel et al. [1]).

   The original MDP is $M = (S, A, P, R)$ and the abstract MDP is $\bar{M} = (\bar{S}, A, \bar{P}, \bar{R})$. M and $\bar{M}$ have the same action space A. The reward function range for both M and $\bar{M}$ is $[0, 1]$, i.e., $0 \leq R(s, a) \leq 1$ and $0 \leq \bar{R}(\bar{s}, a) \leq 1$ for all $s \in S$, $\bar{s} \in \bar{S}$ and $a \in A$. The state space of $\bar{M}$ is a partition of the state space of M defined by an onto mapping $\phi : S \to \bar{S}$. The partition is such that states of M mapped to the same abstract state have similar reward values and state-transition probabilities for every action. Formally, for a given $\epsilon \geq 0$, the mapping $\phi : S \to \bar{S}$ satisfies

$$\phi(s) = \phi(\hat{s}) \Rightarrow \forall a \in A \left[|R(s, a) - R(\hat{s}, a)| \leq \epsilon \quad \text{AND} \right.$$
$$\left. \forall \bar{s} \in \bar{S} \left\{ \left| \sum_{s' \in \bar{s}} P_{ss'}(a) - \sum_{s' \in \bar{s}} P_{\hat{s}s'}(a) \right| \leq \epsilon \right\} \right]$$

   for all $s, \hat{s} \in S$.

   Lemma 2 in Abel et al. [1] bounds the sub-optimality of the optimal policy of $\bar{M}$ extended to M. In our notation, this bound can be expressed as

$$V^*(s) - V^{X[\bar{\pi}^*]}(s) \leq \frac{2\epsilon + 2\gamma\,(|S| - 1)\,\epsilon}{(1-\gamma)^3}$$

$$= \frac{2}{1-\gamma}\left[\frac{\epsilon}{(1-\gamma)^2} + \gamma \cdot \epsilon\,(|S|-1) \cdot \frac{1}{(1-\gamma)^2}\right]$$

for all $s \in S$. We can use theorem 10 and definition 5 to improve this bound to

$$V^*(s) - V^{X[\bar{\pi}^*]}(s) \leq \frac{2}{1-\gamma}\left(\epsilon + \gamma \cdot \epsilon\,\frac{|\bar{S}|}{2} \cdot \frac{1}{1-\gamma}\right).$$

A proof can be found in the appendix (section 7.2) as Lemma 8.

4. Jiang et al.'s theorem 1 in [25].

   The original MDP is $M = (S, A, P, R)$ and the abstract MDP is $\bar{M} = (\bar{S}, A, \bar{P}, \bar{R})$. The state space of $\bar{M}$ is a partition of the state space of M defined by an onto mapping $\phi : S \to \bar{S}$. The reward function range for both M and $\bar{M}$ is $[0, R_{max}]$, i.e., $0 \leq R(s,a) \leq R_{max}$ and $0 \leq \bar{R}(\bar{s},a) \leq R_{max}$ for all $s \in S,\ \bar{s} \in \bar{S}$ and $a \in A$. It is assumed that there exists a probability distribution p defined over the set of all state-action pairs of M, i.e., $p(s,a) \geq 0$ for all $s \in S,\ a \in A$ and

$$\sum_{s\in S,\ a\in A} p(s,a) = 1.$$

   A weight function w is defined for each state-action pair of M as

$$w(s,a) = \frac{p(s,a)}{\sum_{\dot{s}\in\phi(s)} p(\dot{s},a)}.$$

   The state-transition function and reward function of $\bar{M}$ are

$$\bar{P}_{\bar{s}\bar{s}'}(a) = \sum_{s\in\bar{s}} w(s,a) \sum_{s'\in\bar{s}'} P_{ss'}(a),$$

$$\bar{R}(\bar{s},a) = \sum_{s\in\bar{s}} w(s,a) \cdot R(s,a)$$

   for all $\bar{s}, \bar{s}' \in \bar{S}$ and $a \in A$.

The expressions for of $K_r,\ K_p$ and $K(\bar{V})$ reduce to

$$\begin{aligned}
K_r &= \max_{s\in S,\ a\in A} \left|R(s,a) - \bar{R}(\phi(s),a)\right|, \\
K_p &= \max_{s\in S,\ a\in A} \sum_{\bar{s}'\in\bar{S}} \left|\sum_{s'\in\bar{s}'} P_{ss'}(a) - \bar{P}_{\phi(s)\bar{s}'}(a)\right|, \\
K(\bar{V}) &= \max_{s\in S,\ a\in A} \left|Q_{X[\bar{V}]}(s,a) - \bar{Q}_{\bar{V}}(\phi(s),a)\right|.
\end{aligned}$$

$\bar{M}$ is approximated empirically by $\hat{M} = (\bar{S}, A, \hat{P}, \hat{R})$ by sampling state-action pairs of the original MDP M according to the probability distribution p. $\hat{M}$ has the same state space and action space as $\bar{M}$ with $\hat{P}$ and $\hat{R}$ defined empirically using the sampled state-action pairs of M. The details can be found in the paper. $K(\bar{V})$ in definition 4 that relates M and $\bar{M}$. We can define $\bar{K}(\hat{V})$ that relates $\bar{M}$ and $\hat{M}$, i.e.,

$$\bar{K}(\hat{V}) = \max_{\bar{s}\in\bar{S},\ a\in A} \left|\bar{Q}_{\hat{V}}(\bar{s},a) - \hat{Q}_{\hat{V}}(\bar{s},a)\right|.$$

Lemma 3 in Jiang et al. [25] gives an upper bound on $\bar{K}(\hat{V})$ that holds with high probability for all value vectors $\hat{V}$ of $\hat{M}$. We denote this upper bound as $\bar{K}$.

Theorem 1 in Jiang et al. [25] gives a bound on the optimal value vector of the original MDP M and the value vector of the optimal policy of the empirical MDP $\hat{M}$ extended to M. In our notation, we can express this bound as

$$\|V^* - V^{X[\hat{\pi}^*]}\|_\infty \le \frac{2}{(1-\gamma)^2} \left[\left(K_r + \gamma\cdot\frac{K_p}{2}\cdot\frac{R_{max}}{1-\gamma}\right) + \bar{K}\right].$$

This bound can be improved using theorems 8, 9 and 10 by a factor of $\dfrac{1}{1-\gamma}$ as shown below.

Proof.

$$V^* - V^{X[\hat{\pi}^*]} = V^* - X[\bar{V}^*] + X[\bar{V}^*] - X[\bar{V}^{\hat{\pi}^*}] + X[\bar{V}^{\hat{\pi}^*}] - V^{X[\hat{\pi}^*]}.$$

$$\leq \left(\|V^* - X[\bar{V}^*]\|_\infty + \|\bar{V}^* - \bar{V}^{\hat{\pi}^*}\|_\infty + \|X[\bar{V}^{\hat{\pi}^*}] - V^{X[\hat{\pi}^*]}\|_\infty\right).$$

$$\leq \left(\frac{1}{1-\gamma}\cdot K(\bar{V}^*) + \frac{2}{1-\gamma}\cdot \bar{K}(\bar{V}^*) + \frac{1}{1-\gamma}\cdot K(\bar{V}^{\hat{\pi}^*})\right),$$

using theorems 8, 9 and 10

$$= \frac{2}{1-\gamma}\left[\left(K_r + \gamma\cdot\frac{K_p}{2}\cdot\frac{R_{max}}{1-\gamma}\right) + \bar{K}\right].$$

Since $V^*(s) \geq V^{X[\hat{\pi}^*]}(s)$ for all $s \in S$,

$$\|V^* - V^{X[\hat{\pi}^*]}\|_\infty \leq \frac{2}{1-\gamma}\left[\left(K_r + \gamma\cdot\frac{K_p}{2}\cdot\frac{R_{max}}{1-\gamma}\right) + \bar{K}\right].$$

□

5.5 The Feature-based Aggregation Framework

This is the state aggregation framework discussed in Tsitsiklis et al. [58] and Bertsekas [4]. The aggregate problem in this framework is an implicitly defined abstract MDP. We describe the framework briefly here.

The Abstract MDP

The abstract MDP is $\bar{M}$ with state space $\bar{S}$ and the same action space as the original MDP. State-transition and reward functions are not defined for the abstract MDP. Rather, a Bellman backup operator is defined for the abstract MDP in terms of the state-transition and reward functions of the original MDP. The objective is to maximize the expected discounted infinite-horizon sum of rewards with discount factor $\gamma \in (0, 1)$.

Definition

The abstract MDP $\bar{M}$ with state space $\bar{S}$ and the original MDP $M = \langle S, A, P, R\rangle$ are related by the onto mapping $\phi : S \to \bar{S}$, where $\phi(s)$ is based on some feature of state s. The mapping $\phi : S \to \bar{S}$ defines a partition of S as it maps every state in S to an abstract state in $\bar{S}$. An abstract state $\bar{s} \in \bar{S}$ represents a block of the partition defined by ϕ, i.e., $\bar{s} = \{s \in S : \phi(s) = \bar{s}\}$. $W_{\bar{s}}$ is a probability distribution defined on the partition $\bar{s}$, i.e., $\sum_{s \in \bar{s}} W_{\bar{s}}(s) = 1$ for all $\bar{s} \in \bar{S}$.

Solution

The Bellman backup operator of the abstract MDP $\bar{M}$ with respect to value vector $\bar{V}$ is

$$\bar{B}[\bar{V}](\bar{s}) \ = \ \sum_{s \in \bar{s}} W_{\bar{s}}(s) \ \left(\max_{a \in A} \ R(s,a) \ + \ \gamma \sum_{s' \in S} P_{ss'}(a) \cdot X[\bar{V}](s') \right) \tag{5.11}$$

for all $\bar{s} \in \bar{S}$, where $X[\bar{V}]$ is the extension of $\bar{V}$ to M as defined in equation 5.8, i.e.,

$$X[\bar{V}](s) = \bar{V}(\phi(s))$$

for all $s \in S$. $\bar{V}^*$ is the unique fixed point of this operator, i.e., $\bar{V}^* = \bar{B}[\bar{V}^*]$.

Solution Extension

An approximate policy of M based on the abstract problem is a greedy policy with respect to $X[\bar{V}^*]$, denoted $\pi_{X[\bar{V}^*]}$, i.e.,

$$\pi_{X[\bar{V}^*]}(s) \ \in \ \arg\max_{a \in A} \ R(s,a) + \gamma \sum_{s' \in S} P_{ss'}(a_s) \cdot X[\bar{V}^*](s'). \tag{5.12}$$

Main Results

Theorem 11 below gives a bound on the max-norm distance between the optimal value vector of $\bar{M}$ extended to M and the optimal value vector of M, when the mapping ϕ defining the abstract state space is based on the optimal values of states in M. This result has been adapted from theorem 1 (part b) in Tsitsiklis et al. [58].

Theorem 11. *For a given $\epsilon \geq 0$, if*

$$\phi(s) = \phi(\dot{s}) \;\Rightarrow\; |V^*(s) - V^*(\dot{s})| \;\leq\; \epsilon$$

for all $s, \dot{s} \in S$, then $\|V^* - X[\bar{V}^*]\|_\infty \;\leq\; \dfrac{\epsilon}{1-\gamma}$.

Proof. A proof based on the proof of theorem 1 (part b) in Tsitsiklis et al. [58] is given in the appendix (section 7.2). □

Theorem 12 below gives a bound on the max-norm distance between the optimal value vector of M and the value vector of the approximate policy $\pi_{X[\bar{V}^*]}$ of M based on the optimal solution of $\bar{M}$ defined in equation 5.12, when the mapping ϕ defining the abstract state space is based on the optimal values of states in M. This result has been adapted from theorem 1 (part c) in Tsitsiklis et al. [58].

Theorem 12. *For a given $\epsilon \geq 0$, if*

$$\phi(s) = \phi(\dot{s}) \;\Rightarrow\; |V^*(s) - V^*(\dot{s})| \;\leq\; \epsilon$$

for all $s, \dot{s} \in S$, then

$$\|V^* - V^{\pi_{X[\bar{V}^*]}}\|_\infty \;\leq\; \frac{2\epsilon\gamma}{(1-\gamma)^2}.$$

Proof. A proof based on the proof of theorem 1 (part c) in Tsitsiklis et al. [58] is given in the appendix (section 7.2). □

Special cases

We list a few special cases of the feature-based abstraction framework in this section.

1. The Optimal Q-value based State-Abstraction Scheme of Abel et al. [1].

The original MDP is $M = \langle S, A, P, R\rangle$ and the abstract MDP is $\bar{M} = \langle \bar{S}, A, \bar{P}, \bar{R}\rangle$. The state space of the abstract MDP is defined by an onto mapping ϕ that satisfies the condition: for a given $\epsilon \geq 0$

$$\phi(s) = \phi(\dot{s}) \ \Rightarrow\ \forall a \in A \ \ |Q^*(s,a) - Q^*(\dot{s},a)| \ \leq\ \epsilon$$

for all $s, \dot{s} \in S$.

Abel et al.'s [1] claim 1 in section 5.1 gives a bound on the max-norm distance between the optimal Q-value vector of M and the the optimal Q-value vector of $\tilde{M}$ extended to M, i.e.,

$$\left\|Q^* - X[\bar{Q}^*]\right\|_\infty \ \leq\ \frac{\epsilon}{1-\gamma}.$$

Abel et al. [1] have given an inductive proof for this bound, which can be proved easily using the same proof structure as theorem 11. This bound can also obtained from theorem 11 by defining a new original MDP M_X and abstract MDP $\bar{M}_X$ as described below. The new original MDP $M_X = \langle S_X, A, P, R\rangle$, where $S_X = S \times A = \{(s,a) : s \in S,\ a \in A\}$. The only action applicable at state (s,a) is a and the optimal value of state (s,a) is

$$V^*_X(s,a) \ =\ R(s,a) + \gamma \sum_{s' \in S} P_{ss'}(a) \max_{a' \in A} V^*_X(s',a') = Q^*(s,a).$$

The state space of the new abstract MDP is $\bar{S}_X = \bar{S} \times A = \{(\bar{s},a) : \bar{s} \in \bar{S},\ a \in A\}$ defined by the onto mapping $\phi_X(s,a) = (\phi(s),a)$ for all $(s,a) \in S_X$. The only action applicable in state $(\bar{s},a)$ is a. By theorem 11, for a given $\epsilon \geq 0$, if

$$\phi_X(s,a) = \phi_X(\dot{s},a) \ \Rightarrow\ |V^*_X(s,a) - V^*_X(\dot{s},a)| \ \leq\ \epsilon$$

for all $(s,a) \in S_X$, then $\left\|V^*_X - X[\bar{V}^*_X]\right\|_\infty \ =\ \left\|Q^* - X[\bar{Q}^*]\right\|_\infty \ \leq\ \dfrac{\epsilon}{1-\gamma}$.

Chapter 6: Conclusion

We studied the problems of learning and improving policies for probabilistic planning problems in this work.

In the first part, we learned deep reactive neural network policies for benchmark RDDL probabilistic planning problems. For each problem, we trained three different problem-specific networks on the datasets of two expert planners and made the following key observations: (1) For two domains, the expert policies could be represented well using simple linear networks. However, in general, having multiple hidden layers and channels in the networks proved to be useful; (2) The quality of the expert planner influences the quality of the learned policy more than anything else; (3) Except for one domain, the best deep reactive policy was better on average than the expert policies, and our two sparse architectures turned out to be better than the fully-connected architectures for most problems.

In the second part, we focused on Online Search for Policy Improvement (OSPI), where the goal is to improve on a given base policy via online search. We introduced the choice function framework for analysing the performance of OSPI procedures. The main idea is to parameterize search procedures with a choice function that defines the action specification part of search. We identified key properties of choice functions and established sufficient conditions on choice functions for guaranteed policy improvement. We stated a bound on the performance of the online policy returned by search in terms of the quality of the leaf evaluation function. We also introduced a parameterized class of choice function that satisfies the sufficient conditions and covers several existing OSPI procedures as special cases.

In the third part, we drew attention to the issue of policy degradation in OSPI procedures, where the online policy returned by search performs worse than the base policy. This can happen even with OSPI procedures that come with theoretical guarantees on policy improvement under ideal conditions because the ideal conditions might not be satisfied in practice. With the goal of encouraging the development of more reliable OSPI procedures in terms of empirical performance, we proposed benchmark problems with base policies, two parameterized baseline search procedures with a heuristic to deal with policy degradation, evaluation criteria and baseline results on the benchmark problems for comparison.

In the final part, we focused on state aggregation, a problem approximation technique used to compute approximate solutions for Markov decision problems. In state aggregation, an aggregate problem, which is typically smaller then the original problem, is defined and solved, and the solution is extended to the original problem. The approximate solution thus obtained for the original problem will usually be sub-optimal. The degree of sub-optimality depends on the definition of the aggregate problem. We identified two basic aggregation frameworks, the Whitt framework and the Feature-based aggregation framework, and presented the key theoretical results of the two frameworks. We looked at several existing aggregation schemes and related those to the basic frameworks in order to give a unified view of the theoretical sub-optimality results of several aggregation schemes. We were able to simplify the proofs and give tighter results for several of these previous papers.

Chapter 7: Appendix

7.1 The Choice Function Framework: Supplementary Material

Lemma 1. *For any policy π and value vector V, if*
$V - B_\pi[V] \leq \delta$, then $V - V^\pi \leq \dfrac{\delta}{1-\gamma}$.

Proof. Let $P^\pi \in \mathbb{R}^{n \times n}$, $R^\pi \in \mathbb{R}^n$ and $V^\pi \in \mathbb{R}^n$ be the state-transition matrix, reward and value functions of π respectively. We use the matrix notation $V^\pi = R^\pi + \gamma P^\pi V^\pi$ to mean

$$V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} P_{ss'}(\pi(s)) \cdot V^\pi(s')$$

for all $s \in S$. If $D = V - V^\pi$ then

$$\begin{aligned}
D &= V - R^\pi - \gamma P^\pi V^\pi, && \textit{since } V^\pi = R^\pi + \gamma P^\pi V^\pi \\
&= V - R^\pi - \gamma P^\pi V^\pi + \gamma P^\pi V - \gamma P^\pi V, && \textit{add and subtract } \gamma P^\pi V \\
&= \gamma P^\pi (V - V^\pi) + V - R^\pi - \gamma P^\pi V. && \\
&= \gamma P^\pi D + (V - B_\pi[V]), && \textit{since } B_\pi[V] = R^\pi + \gamma P^\pi V \\
&= (V - B_\pi[V]) + \gamma P^\pi D. &&
\end{aligned}$$

Therefore, $D = (V - B_\pi[V]) + \gamma P^\pi D$ is the fixed-point equation of a policy with P^π as the state-transition matrix and $V - B_\pi[V]$ as the reward function. Since the maximum value of the value function of a policy with maximum reward $R_{\max}$ is $\dfrac{R_{\max}}{1-\gamma}$, we get

$$V - V^\pi = D \leq \frac{V - B_\pi[V]}{1-\gamma} \leq \frac{\delta}{1-\gamma}.$$

□

Lemma 2. *If a stationary choice function ψ is π-consistent and monotonic and $u = V^\pi$ then for any path $p; s$ such that $1 \leq |p; s| \leq H(\psi)$,*

$$V_u^\psi(\lrcorner p; s) \geq V_u^\psi(p; s).$$

Proof. By definition,

$$V_u^\psi(p; s) = \begin{cases} V^\pi(s), & \textit{if } \psi(p; s) = \emptyset, \\ \max\limits_{a \in \psi(p;s)} R(s, a) + \gamma \sum\limits_{s' \in S} P_{ss'}(a) V_u^\psi(p; s; a; s'), & \textit{otherwise.} \end{cases}$$

The proof is by induction on $|p; s|$. If $|p; s| = H(\psi)$ then $\psi(p; s) = \emptyset$.

Induction Basis. Let $p; s$ be such that $|p; s| = H(\psi)$.
Case (1). $\psi(\lrcorner p; s) = \emptyset$. Since $|p; s| = H$, we have $\psi(p; s) = \emptyset$ and

$$V_u^\psi(\lrcorner p; s) = V_u^\psi(p; s) = V^\pi(s).$$

Case (2). $\psi(\lrcorner p; s) \neq \emptyset$.

$$\begin{aligned} V_u^\psi(\lrcorner p; s) &= \max_{a \in \psi(\lrcorner p; s)} R(s, a) + \gamma \sum_{s' \in S} P_{ss'}(a) V_u^\psi(\lrcorner p; s; a; s') \\ &= \max_{a \in \psi(\lrcorner p; s)} R(s, a) + \gamma \sum_{s' \in S} P_{ss'}(a) \cdot V^\pi(s'), \quad \textit{since } |\lrcorner p; s; a; s'| = H \\ &\geq R(s, \pi(s)) + \gamma \sum_{s' \in S} P_{ss'}(\pi(s)) \cdot V^\pi(s'), \quad \textit{since } \pi(s) \in \psi(p; s) \\ &= V^\pi(s) = V_u^\psi(p; s). \end{aligned}$$

Induction Hypothesis. Assume that $V_u^\psi(\lrcorner p; s) \geq V_u^\psi(p; s)$ for $p; s$ such that $|p; s| = k+1$.

Inductive Proof. Let $p; s$ be such that $|p; s| = k$.
Case (1). $\psi(\lrcorner p; s) = \emptyset$. Since ψ is monotonic, $\psi(p; s) \subseteq \psi(\lrcorner p; s)$ and hence $\psi(p; s) = \emptyset$.

Therefore,

$$V_u^{\psi}(\lrcorner p; s) = V_u^{\psi}(p; s) = V^{\pi}(s).$$

Case (2). $\psi(\lrcorner p; s) \neq \emptyset$.

$$\begin{aligned}
V_u^{\psi}(\lrcorner p; s) &= \max_{a \in \psi(\lrcorner p; s)} R(s,a) + \gamma \sum_{s' \in S} P_{ss'}(a) V_u^{\psi}(\lrcorner p; s; a; s') \\
&\geq \max_{a \in \psi(\lrcorner p; s)} R(s,a) + \gamma \sum_{s' \in S} P_{ss'}(a) \cdot V_u^{\psi}(p; s; a; s'), \\
&\qquad\qquad \textit{by induction hypothesis, since } |p; s; a; s'| = k+1 \\
&\geq \max_{a \in \psi(p; s)} R(s,a) + \gamma \sum_{s' \in S} P_{ss'}(a) \cdot V_u^{\psi}(p; s; a; s'), \quad \textit{since } \psi(p; s) \subseteq \psi(\lrcorner p; s) \\
&= V_u^{\psi}(p; s).
\end{aligned}$$

□

Lemma 3. *If ψ is a stationary choice function and $\|u - u'\|_\infty \le \epsilon$ then for any path $p; s$ with $|p; s| \le h(\psi)$,*

$$\left|V_u^\psi(p; s) - V_{u'}^\psi(p; s)\right| \le \epsilon\gamma^{h(\psi)-|p;s|}.$$

Proof. The proof is by induction on $|p; s|$ for $|p; s| \le h(\psi)$.
Induction Basis. The lemma holds for $|p; s| = h(\psi)$, since by lemma 5,

$$\left|V_u^\psi(p; s) - V_{u'}^\psi(p; s)\right| \le \epsilon$$

for all $p; s$ with $h(\psi) \le |p; s| \le H(\psi)$.

Induction Hypothesis. Assume that the lemma holds for $|p; s| = h(\psi) - (k - 1)$.

Inductive Proof. Let $p; s$ be a path such that $|p; s| = h(\psi) - k$. By the definition of $h(\psi)$, for any path $p; s$ such that $|p; s| < h(\psi)$, we have $\psi(p; s) \ne \emptyset$. Therefore,

$$\begin{aligned}
\left|V_u^\psi(p; s) - V_{u'}^\psi(p; s)\right| &= \left| \max_{a\in\psi(p;s)} \left\{ R(s, a) + \gamma \sum_{s'\in S} P_{ss'}(a) \cdot V_u^\psi(p; s; a; s') \right\} \right. \\
&\qquad \left. - \max_{a\in\psi(p;s)} \left\{ R(s, a) + \gamma \sum_{s'\in S} P_{ss'}(a) \cdot V_{u'}^\psi(p; s; a; s') \right\} \right| \\
&\le \max_{a\in\psi(p;s)} \left| R(s, a) + \gamma \sum_{s'\in S} P_{ss'}(a) \cdot V_u^\psi(p; s; a; s') - \right. \\
&\qquad \left. \left\{ R(s, a) + \gamma \sum_{s'\in S} P_{ss'}(a) \cdot V_{u'}^\psi(p; s; a; s') \right\} \right| \\
&= \max_{a\in\psi(p;s)} \left| \gamma \sum_{s'\in S} P_{ss'}(a)(V_u^\psi(p; s; a; s') - V_{u'}^\psi(p; s; a; s')) \right. \\
&\le \max_{a\in\psi(p;s)} \gamma \sum_{s'\in S} P_{ss'}(a) \left| V_u^\psi(p; s; a; s') - V_{u'}^\psi(p; s; a; s') \right| \\
&\le \max_{a\in\psi(p;s)} \gamma \sum_{s'\in S} P_{ss'}(a)\ \epsilon\gamma^{h(\psi)-(k-1)}, \quad \textit{by the induction hypothesis} \\
&= \epsilon\gamma^{h(\psi)-k}.
\end{aligned}$$

□

Lemma 4. *If a stationary choice function ψ is π-consistent and monotonic and $\|u - V^{\pi}\|_{\infty} \leq \epsilon_{\pi}$, then for $\pi' = \Pi_u^{\psi}$, $V_{u,0}^{\psi} - B_{\pi'}[V_{u,0}^{\psi}] \leq \epsilon_{\pi}\gamma^{h(\psi)}(1+\gamma)$.*

Proof. Let $\bar{u} = V^{\pi}$ to simplify notation.

$$V_{u,0}^{\psi}(s) - B_{\pi'}[V_{u,0}^{\psi}](s) = R(s,\pi'(s)) + \gamma \sum_{s' \in S} P_{ss'}(\pi'(s)) \cdot V_{u,1}^{\psi}(s;\pi'(s);s') - \left\{ R(s,\pi'(s)) + \gamma \sum_{s' \in S} P_{ss'}(\pi'(s)) \cdot V_{u,0}^{\psi}(s') \right\}.$$

$$\begin{aligned}
&= \gamma \sum_{s' \in S} P_{ss'}(\pi'(s)) \cdot (V_{u,1}^{\psi}(s;\pi'(s);s') - V_{u,0}^{\psi}(s')). \\
&\leq \gamma \sum_{s' \in S} P_{ss'}(\pi'(s)) \left\{ V_{\bar{u},1}^{\psi}(s;\pi'(s);s') + \epsilon_{\pi}\gamma^{h(\psi)-1} - \left(V_{\bar{u},0}^{\psi}(s') - \epsilon_{\pi}\gamma^{h(\psi)} \right) \right\}, \\
&\qquad\qquad \textit{using lemma 3} \\
&\leq \gamma \sum_{s' \in S} P_{ss'}(\pi'(s)) \cdot (\epsilon_{\pi}\gamma^{h(\psi)-1} + \epsilon_{\pi}\gamma^{h(\psi)}), \qquad \textit{by lemma 2} \\
&\qquad\qquad V_{\bar{u},1}^{\psi}(s;\pi'(s);s') \leq V_{\bar{u},0}^{\psi}(s'). \\
&= \epsilon_{\pi}\gamma^{h(\psi)}(1+\gamma).
\end{aligned}$$

□

Lemma 5. *If ψ is a stationary choice function and $\|u - u'\|_\infty \leq \epsilon$ then for any path $p; s$ such that $h(\psi) \leq |p; s| \leq H(\psi)$,*

$$\left| V_u^\psi(p; s) - V_{u'}^\psi(p; s) \right| \leq \epsilon.$$

Proof. The proof is by induction on $|p; s|$ for $h(\psi) \leq |p; s| \leq H(\psi)$.

Induction Basis. Let $p; s$ be a path such that $|p; s| = H(\psi)$. The lemma holds for $|p; s| = H(\psi)$, since $\psi(p; s) = \emptyset$ and

$$\left| V_u^\psi(p; s) - V_{u'}^\psi(p; s) \right| = |u(s) - u'(s)| = \epsilon.$$

Induction Hypothesis. Assume that the lemma holds for any path $p; s$ such that $h(\psi) < |p; s| = H(\psi) - (k - 1)$.

Inductive Proof.
Let $p; s$ be a path such that $|p; s| = H(\psi) - k$.
Case (1). If $\psi(p; s) = \emptyset$ then

$$\left| V_u^\psi(p; s) - V_{u'}^\psi(p; s) \right| \leq \epsilon.$$

Case (2). If $\psi(p; s) \neq \emptyset$ then

$$\begin{aligned} \left| V_u^\psi(p; s) - V_{u'}^\psi(p; s) \right| = & \left| \max_{a \in \psi(p;s)} \left\{ R(s, a) + \gamma \sum_{s' \in S} P_{ss'}(a) \cdot V_u^\psi(p; s; a; s') \right\} - \right. \\ & \left. \max_{a \in \psi(p;s)} \left\{ R(s, a) + \gamma \sum_{s' \in S} P_{ss'}(a) \cdot V_{u'}^\psi(p; s; a; s') \right\} \right| \\ \leq & \max_{a \in \psi(p;s)} \left| R(s, a) + \gamma \sum_{s' \in S} P_{ss'}(a) \cdot V_u^\psi(p; s; a; s') - \right. \\ & \left. \left\{ R(s, a) + \gamma \sum_{s' \in S} P_{ss'}(a) \cdot V_{u'}^\psi(p; s; a; s') \right\} \right| \end{aligned}$$

$$
\begin{aligned}
&= \max_{a\in\psi(p;s)} \left|\gamma \sum_{s'\in S} P_{ss'}(a)(V_u^{\psi}(p;s;a;s') - V_{u'}^{\psi}(p;s;a;s'))\right| \\
&\le \max_{a\in\psi(p;s)} \gamma \sum_{s'\in S} P_{ss'}(a) \left|V_u^{\psi}(p;s;a;s') - V_{u'}^{\psi}(p;s;a;s')\right| \\
&\le \max_{a\in\psi(p;s)} \gamma \sum_{s'\in S} P_{ss'}(a)\cdot\epsilon, \qquad \textit{by the induction hypothesis} \\
&\le \epsilon.
\end{aligned}
$$

□

Proposition 1. *If a stationary choice function ψ is π-consistent and monotonic and $u = V^{\pi}$, then $V_u^{\psi}(s) \ge V^{\pi}(s)$.*

Proof. Let $p;s$ be a path such that $|p;s| = k$ and $\psi(p;s) = \emptyset$. By definition, $V_u^{\psi}(p;s) = V^{\pi}(s)$. Let $\lrcorner^i p;s$ denote the path obtained from $p;s$ by removing the first i state-action pairs of $p;s$.

$$
\begin{aligned}
V^{\pi}(s) &= V_u^{\psi}(p;s) \\
&\le V_u^{\psi}(\lrcorner^i p;s) \le V_u^{\psi}(\lrcorner^{i+1} p;s), \ \ 1 \le i < k, \ \ \textit{by lemma 2} \\
&= V_u^{\psi}(s)
\end{aligned}
$$

□

Lemma 6. *For stationary choice functions ψ and ψ', if ψ and ψ' have the same set of leaf paths and $\psi \supseteq \psi'$, then for any path $p;s$ such that $|p;s| \le H(\psi')$ and leaf evaluation function u, $V_u^{\psi}(p;s) \ge V_u^{\psi'}(p;s)$.*

Proof. The proof is by induction on $H(\psi')$. Since $\psi \supseteq \psi'$ and $\psi'(p;s) = \emptyset \Rightarrow \psi(p;s) = \emptyset$, we have $H(\psi') = H(\psi)$.

Induction Basis. Let $p;s$ be a path such that $|p;s| = H(\psi')$. Since $\psi(p;s) = \psi'(p;s) = \emptyset$,

$$V_u^{\psi}(p;s) = V_u^{\psi'}(p;s) = u(s).$$

The lemma holds for $|p;s| = H(\psi')$.

Induction Hypothesis. Assume that the lemma holds for $|p;s| = k+1 < H(\psi')$.

Inductive Proof. Let $p;s$ be a path with $|p;s| = k$.
Case (1). If $\psi(p;s) = \emptyset$ then

$$V_u^{\psi}(p;s) = V_u^{\psi'}(p;s) = u(s).$$

Case (2). If $\psi(p;s) \neq \emptyset$ then

$$\begin{aligned} V_u^{\psi}(p;s) &= \max_{a \in \psi(p;s)} R(s,a) + \gamma \sum_{s' \in S} P_{ss'}(a) \cdot V_u^{\psi}(p;s;a;s'). \\ &\le \max_{a \in \psi(p;s)} R(s,a) + \gamma \sum_{s' \in S} P_{ss'}(a) \cdot V_u^{\psi'}(p;s;a;s'), \\ &\qquad\qquad \textit{by the induction hypothesis} \\ &\le \max_{a \in \psi'(p;s)} R(s,a) + \gamma \sum_{s' \in S} P_{ss'}(a) \cdot V_u^{\psi'}(p;s;a;s'), \\ &\qquad\qquad \textit{since } \psi(p;s) \supseteq \psi'(p;s) \\ &= V_u^{\psi'}(p;s). \end{aligned}$$

□

Theorem 4. *Let $\Psi = (\psi_1, \psi_2, \ldots)$ be a non-stationary choice function such that each component choice function ψ_t is monotonic and π-consistent and $\|u - V^\pi\|_\infty = \epsilon_\pi$. If all ψ_t have the same set of leaf paths and for each time step t, $\psi_{t+1} \supseteq \psi_t$, then for $\pi' = \Pi_u^\psi$,*

$$V^\pi(s) - V^{\pi'}(s) \leq \frac{2\epsilon_\pi \gamma^{h(\psi_1)}}{1-\gamma}$$

for all $s \in S$.

Proof. Let $\pi'_t = \Pi_u^{\psi_t}$ denote the stationary online policy for the component choice function ψ_t and leaf evaluation function u. Let $P^{\pi'_t}$, $R^{\pi'_t}$ and $V^{\pi'_t}$ be the state-transition matrix, the reward and value functions of policy π'_t. From lemma 4,

$$V_{u,0}^{\psi_t} - B_{\pi'_t}[V_{u,0}^{\psi_t}] \leq \epsilon_\pi \gamma^{h(\psi_t)}(1+\gamma).$$

Since $\psi_{t+1} \supseteq \psi_t$ and $\psi_1(p; s) = \emptyset \Rightarrow \psi_t(p; s) = \emptyset$ for $t \in \{1, 2, \ldots\}$, the min-horizon $h(\psi_t)$ is the same for all the component stationary choice functions ψ_t. Therefore,

$$\begin{aligned} V_{u,0}^{\psi_t} - B_{\pi'_t}[V_{u,0}^{\psi_t}] \leq \epsilon_\pi \gamma^{h(\psi_1)}(1+\gamma) \Rightarrow\ & V_{u,0}^{\psi_t} - R^{\pi'_t} - \gamma P^{\pi'_t} V_{u,0}^{\psi_t} \leq \epsilon_\pi \gamma^{h(\psi_1)}(1+\gamma), \\ & \textit{since } B_\pi[V] = R^\pi + \gamma P^\pi V \textit{ for any policy } \pi \\ \Rightarrow\ & R^{\pi'_t} \geq V_{u,0}^{\psi_t} - \gamma P^{\pi'_t} V_{u,0}^{\psi_t} - \epsilon_\pi \gamma^{h(\psi_1)}(1+\gamma). \end{aligned}$$

Since $V_{u,0}^{\psi_{t+1}} \geq V_{u,0}^{\psi_t}$ *by lemma 6*,

$$R^{\pi'_t} \geq -\epsilon_\pi \gamma^{h(\psi_1)}(1+\gamma) + V_{u,0}^{\psi_t} - \gamma P^{\pi'_t} V_{u,0}^{\psi_{t+1}}.$$

Let $\bar{u} = V^\pi$ to simplify notation. The value function of the online non-stationary policy is

$$V^{\pi'} = \sum_{t=1}^{\infty} \left[\prod_{k=1}^{t-1} \gamma P^{\pi'_k} \right] R^{\pi'_t} \geq \sum_{t=1}^{\infty} \left[\prod_{k=1}^{t-1} \gamma P^{\pi'_k} \right] [-\epsilon_\pi \gamma^{h(\psi_1)}(1+\gamma) + V_{u,0}^{\psi_t} - \gamma P^{\pi'_t} V_{u,0}^{\psi_{t+1}}]$$

$$= \frac{-\epsilon_\pi \gamma^{h(\psi_1)}(1+\gamma)}{1-\gamma} + \sum_{t=1}^{\infty} \left[\prod_{k=1}^{t-1} \gamma P^{\pi'_k} \right] V_{u,0}^{\psi_t} - \sum_{t=1}^{\infty} \left[\prod_{k=1}^{t-1} \gamma P^{\pi'_k} \right] (\gamma P^{\pi'_t} V_{u,0}^{\psi_{t+1}})$$

$$= \frac{-\epsilon_\pi \gamma^{h(\psi_1)}(1+\gamma)}{1-\gamma} + V_{u,0}^{\psi_1} + \sum_{t=2}^{\infty} \left[\prod_{k=1}^{t-1} \gamma P^{\pi'_k} \right] V_{u,0}^{\psi_t} - \sum_{t=1}^{\infty} \left[\prod_{k=1}^{t} \gamma P^{\pi'_k} \right] V_{u,0}^{\psi_{t+1}}$$

$$= \frac{-\epsilon_\pi \gamma^{h(\psi_1)}(1+\gamma)}{1-\gamma} + V_{u,0}^{\psi_1}, \qquad \textit{cancelling out terms 2 and 4}$$

$$\geq \frac{-\epsilon_\pi \gamma^{h(\psi_1)}(1+\gamma)}{1-\gamma} + V_{\bar{u},0}^{\psi_1} - \epsilon_\pi \gamma^{h(\psi_1)}, \qquad \textit{by lemma 3}$$

$$\geq \frac{-\epsilon_\pi \gamma^{h(\psi_1)}(1+\gamma)}{1-\gamma} + V^{\pi} - \epsilon_\pi \gamma^{h(\psi_1)}, \qquad \textit{since } V_{\bar{u},0}^{\psi_1} \geq V^{\pi} \textit{ by proposition 1}$$

$$= \frac{-2\epsilon_\pi \gamma^{h(\psi_1)}}{1-\gamma} + V^{\pi}.$$

Therefore, we get

$$V^{\pi} - V^{\pi'} \leq \frac{2\epsilon_\pi \gamma^{h(\psi_1)}}{1-\gamma}.$$

□

Theorem 5. *For LDCF parameters $\theta = (\pi, H, K, D, \Delta)$, if Δ is depth monotonic, then ψ_θ is monotonic.*

Proof. Consider any path $p; s$ such that $1 \leq |p; s| \leq H$ and let $\lrcorner p; s = p'; s$. We must show that $\psi_\theta(p; s) \subseteq \psi_\theta(p'; s)$. The definition of ψ_θ has three cases, which condition on the input path's length and number of discrepancies. $p'; s$ never increases those quantities compared to $p; s$, since $|p'| = |p| - 1$ and p' does not contain state-action pairs that are not in p and hence cannot increase the number of discrepancies. This means that there are two cases to consider: 1) $p'; s$ satisfies a condition higher in the order than $p; s$, or 2) $p'; s$ satisfies the same condition as $p; s$. For (1), $\psi_\theta(p; s) \subseteq \psi_\theta(p'; s)$ is clearly satisfied when moving either from the bottom to middle or middle to top condition. For (2) we have $\psi_\theta(p; s) = \psi_\theta(p'; s)$ for the bottom and middle conditions. The top condition gives $\psi_\theta(p; s) \subseteq \psi_\theta(p'; s)$ due to the fact that Δ is depth monotonic.

□

Proposition 3. *Let ψ_θ be an LDCF with $\theta = (\pi, H, K, D, \Delta)$, such that $\Delta(s) \leq W$ for any $s \in S$. The number of leaf nodes in T^{ψ_θ} with state branching factor C is upper bounded by $2C^H$ for $(D+1)W = 1$ and otherwise by $\frac{((D+1)W)^{K+1}-1}{(D+1)W-1}C^H = O\left((DW)^K C^H\right)$.*

Proof. The number of leaf nodes is bounded by the number of root-to-leaf paths in T^{ψ_θ}. Each path is a sequence of alternating state and action nodes containing $H+1$ states and H actions. Each path has at most K discrepancies, each discrepancy being the choice of one of the first $D+1$ action nodes and one of at most W discrepancies returned by Δ. Thus, the total number of combinations of K discrepancies is bounded by $((D+1)W)^K$. This bounds the combinations of K or fewer discrepancies by $\frac{((D+1)W)^{K+1}-1}{(D+1)W-1}$ if $(D+1)W > 1$ and by 2 if $(D+1)W = 1$. For each discrepancy combination the number of paths is bounded by the number of ways to assign values to the H state nodes, which is no greater than C^H. Multiplying this with the number of discrepancy combinations completes the proof.

□

Forward Search Sparse Sampling for Choice Functions

FSSS [59] is a computationally efficient implementation of Sparse Sampling (SS) [27]. SS simply builds a search tree of depth H rooted at the current state s, with exactly C successor state-nodes for every action-node, using a simulator of the true MDP. The root action values are computed and used to select the best action. FSSS improves over SS by incrementally constructing the tree via trajectory rollouts as in MCTS algorithms. This allows for sound pruning of the tree by incrementally maintaining intervals $[L_d(s), U_d(s)]$ and $[L_d(s, a), U_d(s, a)]$ at each depth d state-node and action-node respectively. The intervals are guaranteed to contain the state and action values at those nodes, which supports pruning. For example, an action node whose upper bound is lower than another action's lower bound can be safely ignored.

Adapting FSSS to search over at tree T^{ψ} for any search function is straightforward. Since FSSS expands the tree via root-to-leaf rollouts, FSSS simply is adapted to only expand actions allowed by the choice function along the path of each rollout. For the LDCF family our implementation has an additional efficiency enhancement. Two paths $p; s$ and $p'; s$ with equal numbers of discrepancies have $\psi_\theta(p; s) = \psi_\theta(p'; s)$ and thus can be merged to get a Directed Acyclic Graph (DAG) rather than a tree. The DAGs can be significantly more compact than the corresponding trees, which leads to substantially faster search.

7.2 Aggregation in MDPs: Proofs

Lemma 7. *Let $M = (S, \{A_s\}, P, R)$ and $\bar{M} = (\bar{S}, \{\bar{A}_{\bar{s}}\}, \bar{P}, \bar{R})$ denote the original and the abstract MDPs and $\bar{V}$ be an abstract value vector for which*

$$K(\bar{V}) \;=\; \max_{s \in S,\; a_s \in A_s} \left| Q_{X[\bar{V}]}(s, a_s) - \bar{Q}_{\bar{V}}(\phi(s), \psi_s(a_s)) \right|,$$

$$L(\bar{V}) \;=\; \max_{\bar{s} \in \bar{S},\; \bar{a}_{\bar{s}} \in \bar{A}_{\bar{s}}} \quad \max_{(s,\, a_s),\, (\hat{s},\, a_{\hat{s}}) \,\in\, (\bar{s},\, \bar{a}_{\bar{s}})} \left| Q_{X[\bar{V}]}(s, a_s) - Q_{X[\bar{V}]}(\hat{s}, a_{\hat{s}}) \right|.$$

Then $L(\bar{V}) \;\geq K(\bar{V})$.

Proof.

$$\begin{aligned}
L(\bar{V}) \;&=\; \max_{\bar{s} \in \bar{S},\; \bar{a}_{\bar{s}} \in \bar{A}_{\bar{s}}} \quad \max_{(s,a_s),\, (\hat{s},a_{\hat{s}}) \,\in\, (\bar{s},\bar{a}_{\bar{s}})} \left| Q_{X[\bar{V}]}(s, a_s) - Q_{X[\bar{V}]}(\hat{s}, a_{\hat{s}}) \right| \\
&=\; \max_{\bar{s} \in \bar{S},\; \bar{a}_{\bar{s}} \in \bar{A}_{\bar{s}}} \left[\max_{(s,a_s) \,\in\, (\bar{s},\bar{a}_{\bar{s}})} Q_{X[\bar{V}]}(s, a_s) - \min_{(s,a_s) \,\in\, (\bar{s},\bar{a}_{\bar{s}})} Q_{X[\bar{V}]}(s, a_s) \right] \\
&=\; \max_{\bar{s} \in \bar{S},\; \bar{a}_{\bar{s}} \in \bar{A}_{\bar{s}}} \Bigg[\max_{(s,a_s) \,\in\, (\bar{s},\bar{a}_{\bar{s}})} Q_{X[\bar{V}]}(s, a_s) \;-\; \bar{Q}_{\bar{V}}(\bar{s}, \bar{a}_{\bar{s}}) + \\
&\qquad\qquad \bar{Q}_{\bar{V}}(\bar{s}, \bar{a}_{\bar{s}}) \;-\; \min_{(s,a_s) \,\in\, (\bar{s},\bar{a}_{\bar{s}})} Q_{X[\bar{V}]}(s, a_s) \Bigg].
\end{aligned}$$

Since

$$\begin{aligned}
\bar{Q}_{\bar{V}}(\bar{s}, \bar{a}_{\bar{s}}) \;&=\; \bar{R}(\bar{s}, \bar{a}_{\bar{s}}) + \gamma \sum_{\bar{s}' \in \bar{S}} \bar{P}_{\bar{s}\bar{s}'}(\bar{a}_{\bar{s}}) \cdot \bar{V}(\bar{s}') \\
&=\; \sum_{(s,a_s) \in (\bar{s},\bar{a}_{\bar{s}})} W_{\bar{s}\bar{a}_{\bar{s}}}(s, a_s) \left[R(s, a_s) + \gamma \sum_{s' \in S} P_{ss'}(a_s) \cdot X[\bar{V}](s') \right] \\
&=\; \sum_{(s,a_s) \in (\bar{s},\bar{a}_{\bar{s}})} W_{\bar{s}\bar{a}_{\bar{s}}}(s, a_s) \cdot Q_{X[\bar{V}]}(s, a_s)
\end{aligned}$$

for all $\bar{s} \in \bar{S},\; \bar{a}_{\bar{s}} \in \bar{A}_{\bar{s}}$, we have

$$\max_{(s,a_s) \,\in\, (\bar{s},\bar{a}_{\bar{s}})} Q_{X[\bar{V}]}(s, a_s) \;-\; \bar{Q}_{\bar{V}}(\bar{s}, \bar{a}_{\bar{s}}) \geq 0,$$

$$\bar{Q}_{\bar{V}}(\bar{s}, \bar{a}_{\bar{s}}) \; - \min_{(s,a_s) \,\in\, (\bar{s},\bar{a}_{\bar{s}})} Q_{X[\bar{V}]}(s, a_s) \geq 0$$

and therefore

$$\begin{aligned} L(\bar{V}) \;&\geq\; \max_{\bar{s}\in\bar{S},\; \bar{a}_{\bar{s}}\in\bar{A}_{\bar{s}}} \;\; \max_{(s,a_s) \,\in\, (\bar{s},\bar{a}_{\bar{s}})} \left| Q_{X[\bar{V}]}(s, a_s) \; - \; \bar{Q}_{\bar{V}}(\bar{s}, \bar{a}_{\bar{s}}) \right| \\ &=\; \max_{s\in S,\; a_s\in A_s} \left| Q_{X[\bar{V}]}(s, a_s) \; - \; \bar{Q}_{\bar{V}}(\phi(s), \psi_s(a_s)) \right| \\ &=\; K(\bar{V}). \end{aligned}$$

□

Lemma 8. *Let $M = (S, \{A_s\}, P, R)$ be an MDP with optimal value function V^* and $\bar{M} = (\bar{S}, A, \bar{P}, \bar{R})$ be an abstract MDP with optimal value function $\bar{V}^*$ and optimal policy $\bar{\pi}^*$. Let $\phi : S \to \bar{S}$ be an onto mapping, which, for a given $\epsilon \geq 0$ satisfies*

$$\phi(s) = \phi(\hat{s}) \;\Rightarrow \forall a \in A \left[|R(s,a) - R(\hat{s},a)| \leq \epsilon \quad AND \right.$$

$$\left. \forall \bar{s} \in \bar{S} \; \left\{ \Big| \sum_{s' \in \bar{s}} P_{ss'}(a) - \sum_{s' \in \bar{s}} P_{\hat{s}s'}(a) \Big| \leq \epsilon \right\} \right]$$

for all $s, \hat{s} \in S$. Let the reward function range of both M and $\bar{M}$ be $[0,1]$. Then

$$V^*(s) - V^{X[\bar{\pi}^*]}(s) \;\leq\; \frac{2}{1-\gamma} \left(\epsilon + \gamma \cdot \epsilon \, \frac{|\bar{S}|}{2} \cdot \frac{1}{1-\gamma} \right).$$

Proof.

$$V^*(s) - V^{X[\bar{\pi}^*]}(s) \;\leq\; \frac{2}{1-\gamma} \cdot K(\bar{V}^*), \qquad \text{by theorem 10}$$

$$\leq \; \frac{2}{1-\gamma} \cdot L(\bar{V}^*), \qquad \text{by definition 5}$$

$$= \; \frac{2}{1-\gamma} \cdot \max_{\bar{s} \in \bar{S},\, a \in A} \; \max_{s,\, \hat{s} \,\in\, \bar{s}} \; \left| Q_{X[\bar{V}^*]}(s,a) - Q_{X[\bar{V}^*]}(\hat{s},a) \right|$$

$$= \; \frac{2}{1-\gamma} \cdot \max_{\bar{s} \in \bar{S},\, a \in A} \; \max_{s,\, \hat{s} \,\in\, \bar{s}} \; \Big| R(s,a) \; + \; \gamma \sum_{s' \in S} P_{ss'}(a) \cdot X[\bar{V}^*](s') \; -$$

$$R(\hat{s},a) \; - \; \gamma \sum_{s' \in S} P_{\hat{s}s'}(a) \cdot X[\bar{V}^*](s') \Big|$$

$$= \; \frac{2}{1-\gamma} \left\{ \max_{\bar{s} \in \bar{S},\, a \in A} \; \max_{s,\, \hat{s} \,\in\, \bar{s}} \; \left| R(s,a) \; - \; R(\hat{s},a) \right| + \right.$$

$$\left. \max_{\bar{s} \in \bar{S},\, a \in A} \; \max_{s,\, \hat{s} \,\in\, \bar{s}} \; \gamma \left| \sum_{\bar{s}' \in \bar{S}} \left[\sum_{s' \in \bar{s}'} P_{ss'}(a) - \sum_{s' \in \bar{s}'} P_{\hat{s}s'}(a) \right] \cdot \bar{V}^*(\bar{s}') \right| \right\}$$

$$= \; \frac{2}{1-\gamma} \left\{ \epsilon \; + \; \max_{\bar{s} \in \bar{S},\, a \in A} \; \max_{s,\, \hat{s} \,\in\, \bar{s}} \; \gamma \right.$$

$$\left|\sum_{\bar{s}'\in\bar{S}} \left[\left(\sum_{s'\in\bar{s}'} P_{ss'}(a) \;-\; \min\left\{\sum_{s'\in\bar{s}'} P_{ss'}(a),\; \sum_{s'\in\bar{s}'} P_{\hat{s}s'}(a)\right\}\right) - \right.\right.$$

$$\left.\left.\left(\sum_{s'\in\bar{s}'} P_{\hat{s}s'}(a) \;-\; \min\left\{\sum_{s'\in\bar{s}'} P_{ss'}(a),\; \sum_{s'\in\bar{s}'} P_{\hat{s}s'}(a)\right\}\right)\right] \cdot \bar{V}^*(\bar{s}')\right| \;\right\}$$

$$\leq\; \frac{2}{1-\gamma}\left\{\epsilon \;+\; \max_{\bar{s}\in\bar{S},\, a\in A}\;\; \max_{s,\, \hat{s}\,\in\,\bar{s}}\; \gamma\right.$$

$$\left|\sum_{\bar{s}'\in\bar{S}} \left(\sum_{s'\in\bar{s}'} P_{ss'}(a) \;-\; \min\left\{\sum_{s'\in\bar{s}'} P_{ss'}(a),\; \sum_{s'\in\bar{s}'} P_{\hat{s}s'}(a)\right\}\right) \cdot \max_{\bar{s}\in\bar{S}}\; \bar{V}^*(\bar{s}') \;-\right.$$

$$\left.\left.\sum_{\bar{s}'\in\bar{S}} \left(\sum_{s'\in\bar{s}'} P_{\hat{s}s'}(a) \;-\; \min\left\{\sum_{s'\in\bar{s}'} P_{ss'}(a),\; \sum_{s'\in\bar{s}'} P_{\hat{s}s'}(a)\right\}\right) \cdot \min_{\bar{s}'\in\bar{S}}\; \bar{V}^*(\bar{s}')\right| \;\right\}.$$

Since

$$\sum_{\bar{s}'\in\bar{S}} \left(\sum_{s'\in\bar{s}'} P_{ss'}(a) \;-\; \min\left\{\sum_{s'\in\bar{s}'} P_{ss'}(a),\; \sum_{s'\in\bar{s}'} P_{\hat{s}s'}(a)\right\}\right) =$$

$$\sum_{\bar{s}'\in\bar{S}} \left(\sum_{s'\in\bar{s}'} P_{\hat{s}s'}(a) \;-\; \min\left\{\sum_{s'\in\bar{s}'} P_{ss'}(a),\; \sum_{s'\in\bar{s}'} P_{\hat{s}s'}(a)\right\}\right),$$

$$\sum_{\bar{s}'\in\bar{S}} \left(\sum_{s'\in\bar{s}'} P_{ss'}(a) \;-\; \min\left\{\sum_{s'\in\bar{s}'} P_{ss'}(a),\; \sum_{s'\in\bar{s}'} P_{\hat{s}s'}(a)\right\}\right) +$$

$$\sum_{\bar{s}'\in\bar{S}} \left(\sum_{s'\in\bar{s}'} P_{\hat{s}s'}(a) \;-\; \min\left\{\sum_{s'\in\bar{s}'} P_{ss'}(a),\; \sum_{s'\in\bar{s}'} P_{\hat{s}s'}(a)\right\}\right) \;\leq\; \epsilon\; |\bar{S}|$$

and

$$\max_{\bar{s}\in\bar{S}}\; \bar{V}^*(\bar{s}) \;\leq\; \frac{1}{1-\gamma} \quad \text{and} \quad \min_{\bar{s}\in\bar{S}}\; \bar{V}^*(\bar{s}) \;\geq\; 0$$

we have

$$V^*(s) - V^{X[\bar{\pi}^*]}(s) \;\leq\; \frac{2}{1-\gamma}\;\left(\epsilon + \gamma\cdot\epsilon\,\frac{|\bar{S}|}{2}\cdot\frac{1}{1-\gamma}\right).$$

$\square$

Theorem 6. *For a given value vector $\bar{V}$ of the abstract MDP $\bar{M}$*

$$K(\bar{V}) \;\le\; K_r + \gamma \cdot \frac{K_q}{2} \cdot \delta(\bar{V}).$$

Proof.

$$\begin{aligned}
K(\bar{V}) \;&=\; \max_{s\in S,\, a_s\in A_s} \left| Q_{X[\bar{V}]}(s,a_s) - \bar{Q}_{\bar{V}}(\phi(s),\psi_s(a_s)) \right| . \\
&=\; \max_{s\in S,\, a_s\in A_s} \Bigg| R(s,a_s) + \;\gamma \sum_{s'\in S} P_{ss'}(a_s) \cdot X[\bar{V}](s') - \\
&\qquad\qquad \bar{R}(\phi(s),\psi_s(a_s)) - \gamma \sum_{\bar{s}'\in\bar{S}} \bar{P}_{\phi(s)\bar{s}'}(\psi_s(a_s)) \cdot \bar{V}(\bar{s}') \Bigg| . \\
&\le\; \max_{s\in S,\, a_s\in A_s} \left| R(s,a_s) - \bar{R}(\phi(s),\psi_s(a_s)) \right| \;+\; \gamma \\
&\qquad\qquad \max_{s\in S,\, a_s\in A_s} \left| \sum_{\bar{s}'\in\bar{S}} \left(\sum_{s'\in\bar{s}'} P_{ss'}(a_s) - \bar{P}_{\phi(s)\bar{s}'}(\psi_s(a_s)) \right) \cdot \bar{V}(\bar{s}') \right| . \\
&\le\; K_r \;+\; \gamma \;\; \max_{s\in S,\, a_s\in A_s} \Bigg| \sum_{\bar{s}'\in\bar{S}} \\
&\qquad \Bigg(\Bigg(\sum_{s'\in\bar{s}'} P_{ss'}(a_s) - \min\Bigg\{ \sum_{s'\in\bar{s}'} P_{ss'}(a_s),\; \bar{P}_{\phi(s)\bar{s}'}(\psi_s(a_s)) \Bigg\} \Bigg) \cdot \bar{V}(\bar{s}') - \\
&\quad \Bigg(\bar{P}_{\phi(s)\bar{s}'}(\psi_s(a_s)) - \min\Bigg\{ \sum_{s'\in\bar{s}'} P_{ss'}(a_s),\; \bar{P}_{\phi(s)\bar{s}'}(\psi_s(a_s)) \Bigg\} \Bigg) \cdot \bar{V}(\bar{s}') \Bigg) \Bigg| . \qquad (7.1)
\end{aligned}$$

In order to simplify equation (7.1) above, let

$$\begin{aligned}
C_{\bar{s}'}(s,a_s) \;&=\; \min\Bigg\{ \sum_{s'\in\bar{s}'} P_{ss'}(a_s),\; \bar{P}_{\phi(s)\bar{s}'}(\psi_s(a_s)) \Bigg\}, \\
D_{\bar{s}'}(s,a_s) \;&=\; \sum_{s'\in\bar{s}'} P_{ss'}(a_s) \;-\; C_{\bar{s}'}(s,a_s), \\
E_{\bar{s}'}(s,a_s) \;&=\; \bar{P}_{\phi(s)\bar{s}'}(\psi_s(a_s)) \;-\; C_{\bar{s}'}(s,a_s).
\end{aligned}$$

Therefore, from equation (7.1), we get

$$K(\bar{V}) \ \le \ K_r \ + \ \gamma \ \max_{s\in S,\ a_s\in A_s} \left| \sum_{\bar{s}'\in\bar{S}} D_{\bar{s}'}(s,a_s)\cdot \max_{\bar{s}'\in\bar{S}} \ \bar{V}(\bar{s}') - \sum_{\bar{s}'\in\bar{S}} E_{\bar{s}'}(s,a_s)\cdot \min_{\bar{s}'\in\bar{S}} \ \bar{V}(\bar{s}')\right|.$$

Both $D_{\bar{s}'}(s,a_s)$ and $E_{\bar{s}'}(s,a_s)$ are non-negative for all $s \in S,\ a_s \in A_s$ and $\bar{s}' \in \bar{S}$. Since $\sum_{\bar{s}'\in\bar{S}} \ \sum_{s'\in\bar{s}'} P_{ss'}(a_s) = 1$ and $\sum_{\bar{s}'\in\bar{S}} \ \bar{P}_{\phi(s)\bar{s}'}(\psi_s(a_s)) = 1$ for all $s \in S$ and $a_s \in A_s$,

$$\sum_{\bar{s}'\in\bar{S}} D_{\bar{s}'}(s,a_s) - \sum_{\bar{s}'\in\bar{S}} E_{\bar{s}'}(s,a_s) \ = \ 0 \quad \Rightarrow \quad \sum_{\bar{s}'\in\bar{S}} D_{\bar{s}'}(s,a_s) = \sum_{\bar{s}'\in\bar{S}} E_{\bar{s}'}(s,a_s)$$

and also

$$\sum_{\bar{s}'\in\bar{S}} D_{\bar{s}'}(s,a_s) + \sum_{\bar{s}'\in\bar{S}} E_{\bar{s}'}(s,a_s) = \sum_{\bar{s}'\in\bar{S}} \left| \sum_{s'\in\bar{s}'} P_{ss'}(a_s) - \bar{P}_{\bar{s}\bar{s}'}(\psi_s(a_s))\right| = K_q - \Delta(s,a_s)$$

for some $\Delta(s,a_s) \ge 0$. Therefore

$$\begin{aligned} K(\bar{V}) \ &= \ K_r \ + \ \gamma \ \max_{s\in S,\ a_s\in A_s} \left| \frac{K_q - \Delta(s,a_s)}{2} \cdot \max_{\bar{s}'\in\bar{S}} \ \bar{V}(\bar{s}') - \frac{K_q - \Delta(s,a_s)}{2} \cdot \min_{\bar{s}'\in\bar{S}} \ \bar{V}(\bar{s}')\right|. \\ &= \ K_r \ + \ \gamma \ \max_{s\in S,\ a_s\in A_s} \left| \frac{K_q - \Delta(s,a_s)}{2} \cdot \delta(\bar{V})\right|. \\ &\le \ K_r + \gamma \cdot \frac{K_q}{2} \cdot \delta(\bar{V}). \end{aligned}$$

□

Theorem 7. *For any deterministic, stationary policy $\bar{\pi}$ of $\bar{M}$*

$$\|X[\bar{Q}^{\bar{\pi}}] - Q^{X[\bar{\pi}]}\|_\infty \;\le\; \frac{1}{1-\gamma\left(1-\frac{K_q}{2}\right)} \;\left(K_r + \gamma \cdot \frac{K_q}{2} \cdot \frac{R_{max}-R_{min}}{1-\gamma}\right),$$

where $[R_{min}, R_{max}]$ is the range of the reward functions of both the original and abstract MDPs.

Proof. For every state-action pair (s, a_s) of the original MDP M

$$X[\bar{Q}^{\bar{\pi}}](s,a_s) - Q^{X(\bar{\pi})}(s,a_s) \;=\; \bar{Q}^{\bar{\pi}}(\phi(s),\psi_s(a_s)) - Q^{X[\bar{\pi}]}(s,a_s).$$

$$= \;\bar{R}(\phi(s),\psi_s(a_s)) + \gamma \sum_{\bar{s}'\in\bar{S}} \bar{P}_{\phi(s)\bar{s}'}(\psi_s(a_s)) \cdot \bar{V}^{\bar{\pi}}(\bar{s}') \;-\; R(s,a_s) \;-$$

$$\gamma \sum_{s'\in S} P_{ss'}(a) \cdot V^{X[\bar{\pi}]}(s').$$

$$\le\; K_r + \gamma \sum_{\bar{s}'\in\bar{S}} \left(\bar{P}_{\phi(s)\bar{s}'}(\psi_s(a_s)) \cdot \bar{V}^{\bar{\pi}}(\bar{s}') - \sum_{s'\in\bar{s}'} P_{ss'}(a_s) \cdot V^{X[\bar{\pi}]}(s')\right).$$

$$\le\; K_r + \gamma \sum_{\bar{s}'\in\bar{S}} \left(\bar{P}_{\phi(s)\bar{s}'}(\psi_s(a_s)) \cdot \bar{V}^{\bar{\pi}}(\bar{s}') - \left(\sum_{s'\in\bar{s}'} P_{ss'}(a_s)\right) \cdot \min_{s'\in\bar{s}'} \; V^{X[\bar{\pi}]}(s')\right). \tag{7.2}$$

In order to simplify equation (7.2) above, let

$$C_{\bar{s}'}(s,a_s) \;=\; \min\left\{\sum_{s'\in\bar{s}'} P_{ss'}(a_s),\; \bar{P}_{\phi(s)\bar{s}'}(\psi_s(a_s))\right\},$$

$$D_{\bar{s}'}(s,a_s) \;=\; \sum_{s'\in\bar{s}'} P_{ss'}(a_s) \;-\; C_{\bar{s}'}(s,a_s),$$

$$E_{\bar{s}'}(s,a_s) \;=\; \bar{P}_{\phi(s)\bar{s}'}(\psi_s(a_s)) \;-\; C_{\bar{s}'}(s,a_s).$$

Therefore equation (7.2) can be written as

$$X[\bar{Q}^{\bar{\pi}}](s,a_s) - Q^{X(\bar{\pi})}(s,a_s) \;\le\; K_r + \gamma \sum_{\bar{s}'\in\bar{S}} \left(\left(E_{\bar{s}'}(s,a_s) + C_{\bar{s}'}(s,a_s)\right) \cdot \bar{V}^{\bar{\pi}}(\bar{s}') \;-\right.$$

$$\left(D_{\bar{s}'}(s,a_s) + C_{\bar{s}'}(s,a_s)\right) \cdot \min_{s' \in \bar{s}'} V^{X[\bar{\pi}]}(s')\Bigg).$$

$$\leq K_r + \gamma \sum_{\bar{s}' \in \bar{S}} \Bigg(C_{\bar{s}'}(s,a_s) \cdot \|V^{X[\bar{\pi}]} - X[\bar{V}^{\bar{\pi}}]\|_\infty +$$

$$E_{\bar{s}'}(s,a_s) \cdot \frac{R_{max}}{1-\gamma} - D_{\bar{s}'}(s,a_s) \cdot \frac{R_{min}}{1-\gamma} \Bigg).$$

Both $D_{\bar{s}'}(s,a_s)$ and $E_{\bar{s}'}(s,a_s)$ are non-negative for all $s \in S,\ a_s \in A_s$ and $\bar{s}' \in \bar{S}$. Since $\sum_{\bar{s}' \in \bar{S}} \sum_{s' \in \bar{s}'} P_{ss'}(a_s) = 1$ and $\sum_{\bar{s}' \in \bar{S}} \bar{P}_{\phi(s)\bar{s}'}(\psi_s(a_s)) = 1$ for all $s \in S$ and $a_s \in A_s$,

$$\sum_{\bar{s}' \in \bar{S}} D_{\bar{s}'}(s,a_s) - \sum_{\bar{s}' \in \bar{S}} E_{\bar{s}'}(s,a_s) \ = \ 0 \quad \Rightarrow \quad \sum_{\bar{s}' \in \bar{S}} D_{\bar{s}'}(s,a_s) = \sum_{\bar{s}' \in \bar{S}} E_{\bar{s}'}(s,a_s)$$

and also

$$\sum_{\bar{s}' \in \bar{S}} D_{\bar{s}'}(s,a_s) + \sum_{\bar{s}' \in \bar{S}} E_{\bar{s}'}(s,a_s) = \sum_{\bar{s}' \in \bar{S}} \left| \sum_{s' \in \bar{s}'} P_{ss'}(a_s) - \bar{P}_{\bar{s}\bar{s}'}(\psi_s(a_s)) \right| = K_q - \Delta(s,a_s)$$

for some $\Delta(s,a_s) \geq 0$, which implies

$$\sum_{\bar{s}' \in \bar{S}} C_{\bar{s}'}(s,a_s) \ = \ 1 - \sum_{\bar{s}' \in \bar{S}} D_{\bar{s}'}(s,a_s) \ = \ 1 - \frac{K_q - \Delta(s,a_s)}{2}.$$

Therefore

$$X[\bar{Q}^{\bar{\pi}}](s,a_s) - Q^{X(\bar{\pi})}(s,a_s) \ \leq \ K_r + \gamma \left(\left(1 - \frac{K_q - \Delta(s,a_s)}{2} \right) \cdot \|V^{X[\bar{\pi}]} - X[\bar{V}^{\bar{\pi}}]\|_\infty + \right.$$

$$\left. \frac{K_q - \Delta(s,a_s)}{2} \cdot \left(\frac{R_{max} - R_{min}}{1-\gamma} \right) \right).$$

$$\leq \ K_r + \gamma \left(\left(1 - \frac{K_q}{2} \right) \cdot \|V^{X[\bar{\pi}]} - X[\bar{V}^{\bar{\pi}}]\|_\infty \ + \ \frac{K_q}{2} \cdot \left(\frac{R_{max} - R_{min}}{1-\gamma} \right) \right).$$

$$\leq \ K_r + \gamma \left(1 - \frac{K_q}{2} \right) \cdot \|Q^{X[\bar{\pi}]} - X[\bar{Q}^{\bar{\pi}}]\|_\infty \ + \ \gamma \cdot \frac{K_q}{2} \cdot \left(\frac{R_{max} - R_{min}}{1-\gamma} \right).$$

Similarly, we can derive

$$Q^{X(\bar{\pi})}(s,a_s) - X[\bar{Q}^{\bar{\pi}}](s,a_s) \;\leq\; K_r + \gamma\left(1-\frac{K_q}{2}\right)\cdot\|Q^{X[\bar{\pi}]} - X[\bar{Q}^{\bar{\pi}}]\|_\infty + \gamma\cdot\frac{K_q}{2}\cdot\left(\frac{R_{max}-R_{min}}{1-\gamma}\right).$$

Therefore

$$\|X[\bar{Q}^{\bar{\pi}}] - Q^{X(\bar{\pi})}\|_\infty \leq K_r + \gamma\left(1-\frac{K_q}{2}\right)\|Q^{X[\bar{\pi}]} - X[\bar{Q}^{\bar{\pi}}]\|_\infty \;+\; \gamma\cdot\frac{K_q}{2}\cdot\left(\frac{R_{max}-R_{min}}{1-\gamma}\right).$$

which implies

$$\|X[\bar{Q}^{\bar{\pi}}] - Q^{X(\bar{\pi})}\|_\infty \;\leq\; \frac{1}{1-\gamma\left(1-\frac{K_q}{2}\right)}\left(K_r + \gamma\cdot\frac{K_q}{2}\cdot\frac{R_{max}-R_{min}}{1-\gamma}\right).$$

□

Corollary 2. *For any deterministic, stationary policy $\bar{\pi}$ of $\bar{M}$*

$$\|X[\bar{V}^{\bar{\pi}}] - V^{X[\bar{\pi}]}\|_\infty \;\le\; \frac{1}{1-\gamma\left(1-\frac{K_q}{2}\right)}\;\left(K_r + \gamma\cdot\frac{K_q}{2}\cdot\frac{R_{max}-R_{min}}{1-\gamma}\right),$$

where $[R_{min}, R_{max}]$ is the range of the reward functions of both the original and abstract MDPs.

Proof.

$$\begin{aligned}
\|X[\bar{V}^{\bar{\pi}}] - V^{X[\bar{\pi}]}\|_\infty \;&=\; \max_{s\in S}\;\left|X[\bar{V}^{\bar{\pi}}](s) - V^{X[\bar{\pi}]}(s)\right| \\
&=\; \max_{s\in S}\;\left|X[\bar{Q}^{\bar{\pi}}](s, X[\bar{\pi}](s)) - Q^{X[\bar{\pi}]}(s, X[\bar{\pi}](s))\right| \\
&\le\; \max_{s\in S,\, a\in A_s}\;\left|X[\bar{Q}^{\bar{\pi}}](s, a_s) - Q^{X[\bar{\pi}]}(s, a_s)\right| \\
&=\; \|X[\bar{Q}^{\bar{\pi}}] - Q^{X[\bar{\pi}]}\|_\infty \\
&\le\; \frac{1}{1-\gamma\left(1-\frac{K_q}{2}\right)}\;\left(K_r + \gamma\cdot\frac{K_q}{2}\cdot\frac{R_{max}-R_{min}}{1-\gamma}\right).
\end{aligned}$$

□

Proposition 4. *For an MDP $M = \langle S, A, P, R \rangle$ with discount factor γ, the max-norm distance between a given vector V and the optimal value vector V^* of M is*

$$\|V^* - V\|_\infty \;\leq\; \frac{1}{1-\gamma}\;\|B[V] - V\|_\infty,$$

where B is the Bellman optimality operator.

Proof.

$$\begin{aligned}
\|V^* - V\|_\infty &= \left\|\lim_{n\to\infty} B^n[V] - V\right\|_\infty \\
&= \lim_{n\to\infty} \|B^n[V] - V\|_\infty \\
&= \lim_{n\to\infty} \left\|\sum_{k=0}^{n-1} \Big(B^{k+1}[V] - B^k[V]\Big)\right\|_\infty \\
&\leq \lim_{n\to\infty} \sum_{k=0}^{n-1} \left\|B^{k+1}[V] - B^k[V]\right\|_\infty \\
&\leq \lim_{n\to\infty} \sum_{k=0}^{n-1} \gamma^k \;\|B[V] - V\|_\infty\,, \qquad \text{by the contraction property of } B \\
&\leq \frac{1}{1-\gamma}\;\|B[V] - V\|_\infty
\end{aligned}$$

□

Theorem 8. *For any deterministic, stationary policy* $\bar{\pi}$ *of* $\bar{M}$

$$\|X[\bar{V}^{\bar{\pi}}] - V^{X[\bar{\pi}]}\|_\infty \ \leq \ \frac{1}{1-\gamma} \cdot K(\bar{V}^{\bar{\pi}}).$$

Proof. The proof follows from a bound on $\|B_{X[\bar{\pi}]}[X[\bar{V}^{\bar{\pi}}]] - X[\bar{V}^{\bar{\pi}}]\|_\infty$. For any state $s \in S$

$$\begin{aligned}
\left|B_{X[\bar{\pi}]}[X[\bar{V}^{\bar{\pi}}]](s) - X[\bar{V}^{\bar{\pi}}](s)\right| \ &= \ \left|Q_{X[\bar{V}^{\bar{\pi}}]}(s, X[\bar{\pi}](s)) - \bar{V}^{\bar{\pi}}(\phi(s))\right| \\
&= \ \left|Q_{X[\bar{V}^{\bar{\pi}}]}(s, X[\bar{\pi}](s)) - \bar{Q}_{\bar{V}^{\bar{\pi}}}(\phi(s), \bar{\pi}(\phi(s)))\right| \\
&= \ \left|Q_{X[\bar{V}^{\bar{\pi}}]}(s, X[\bar{\pi}](s)) - \bar{Q}_{\bar{V}^{\bar{\pi}}}(\phi(s), \psi_s(X[\bar{\pi}](s)))\right| \\
&\leq \ \max_{a_s \in A_s} \left|Q_{X[\bar{V}^{\bar{\pi}}]}(s, a_s) - \bar{Q}_{\bar{V}^{\bar{\pi}}}(\phi(s), \psi_s(a_s))\right| \\
&\leq \ \max_{s \in S,\ a_s \in A_s} \left|Q_{X[\bar{V}^{\bar{\pi}}]}(s, a_s) - \bar{Q}_{\bar{V}^{\bar{\pi}}}(\phi(s), \psi_s(a_s))\right| \\
&\leq \ K(\bar{V}^{\bar{\pi}}).
\end{aligned}$$

Therefore $\|B_{X[\bar{\pi}]}[X[\bar{V}^{\bar{\pi}}]] - X[\bar{V}^{\bar{\pi}}]\|_\infty \ \leq \ K(\bar{V}^{\bar{\pi}})$ and by proposition 4

$$\|X[\bar{V}^{\bar{\pi}}] - V^{X[\bar{\pi}]}\|_\infty \ \leq \ \frac{1}{1-\gamma} \cdot K(\bar{V}^{\bar{\pi}}).$$

□

Corollary 3. *For any deterministic, stationary policy* $\bar{\pi}$ *of* $\bar{M}$ *and its stochastic extension* $\mathcal{X}[\bar{\pi}]$ *given by*

$$\mathcal{X}[\bar{\pi}](s, a_s) \;=\; \frac{\bar{\pi}(\phi(s), \psi_s(a_s))}{|\{a' \in A_s : \psi_s(a') = \psi_s(a_s)\}|}$$

for all $s \in S$ *and* $a_s \in A_s$*, we have*

$$\|X[\bar{V}^{\bar{\pi}}] - V^{\mathcal{X}[\bar{\pi}]}\|_\infty \;\leq\; \frac{1}{1-\gamma} \cdot K(\bar{V}^{\bar{\pi}}).$$

Proof. The proof follows from a bound on $\|B_{\mathcal{X}[\bar{\pi}]}[X[\bar{V}^{\bar{\pi}}]] - X[\bar{V}^{\bar{\pi}}]\|_\infty$. For any state $s \in S$

$$\begin{aligned}
\left|B_{\mathcal{X}[\bar{\pi}]}[X[\bar{V}^{\bar{\pi}}]](s) - X[\bar{V}^{\bar{\pi}}](s)\right| \;&=\; \left|\sum_{a_s \in A_s} \mathcal{X}[\bar{\pi}](s, a_s) \cdot Q_{X[\bar{V}^{\bar{\pi}}]}(s, a_s) - \bar{V}^{\bar{\pi}}(\phi(s))\right| \\
&=\; \left|\sum_{a_s \in A_s} \mathcal{X}[\bar{\pi}](s, a_s) \cdot Q_{X[\bar{V}^{\bar{\pi}}]}(s, a_s) - \bar{Q}_{\bar{V}^{\bar{\pi}}}(\phi(s), \bar{\pi}(\phi(s)))\right| \\
&=\; \sum_{a_s \in A_s} \mathcal{X}[\bar{\pi}](s, a_s) \left|Q_{X[\bar{V}^{\bar{\pi}}]}(s, a_s) - \bar{Q}_{\bar{V}^{\bar{\pi}}}(\phi(s), \psi_s(a_s))\right| \\
&\leq \sum_{a_s \in A_s} \mathcal{X}[\bar{\pi}](s, a_s) \cdot K(\bar{V}^{\bar{\pi}}) \\
&=\; K(\bar{V}^{\bar{\pi}}).
\end{aligned}$$

Therefore $\|B_{\mathcal{X}[\bar{\pi}]}[X[\bar{V}^{\bar{\pi}}]] - X[\bar{V}^{\bar{\pi}}]\|_\infty \;\leq\; K(\bar{V}^{\bar{\pi}})$ and by proposition 4

$$\|X[\bar{V}^{\bar{\pi}}] - V^{\mathcal{X}[\bar{\pi}]}\|_\infty \;\leq\; \frac{1}{1-\gamma} \cdot K(\bar{V}^{\bar{\pi}}).$$

□

Theorem 9. *For the optimal value vectors V^* and $\bar{V}^*$ of the original MDP M and the abstract MDP $\bar{M}$, we have*

$$\|V^* - X[\bar{V}^*]\|_\infty \;\le\; \frac{1}{1-\gamma} \cdot K(\bar{V}^*).$$

Proof. The proof follows from a bound on $\|X[\bar{V}^*] - B[X[\bar{V}^*]]\|_\infty$. For any state $s \in S$

$$\begin{aligned}
B[X[\bar{V}^*]](s) - X[\bar{V}^*](s) \;&=\; \max_{a_s \in A_s} \left\{ R(s, a_s) + \gamma \sum_{s' \in S} P_{ss'}(a_s) \cdot X[\bar{V}^*](s') \right\} - \bar{V}^*(\phi(s)) \\
&= \max_{a_s \in A_s} \left\{ R(s, a_s) + \gamma \sum_{s' \in S} P_{ss'}(a_s) \cdot X[\bar{V}^*](s') - \bar{V}^*(\phi(s)) \right\} \\
&= \max_{a_s \in A_s} \left\{ R(s, a_s) + \gamma \sum_{s' \in S} P_{ss'}(a_s) \cdot X[\bar{V}^*](s') - \bar{B}_{\bar{\pi}^*}[\bar{V}^*](\phi(s)) \right\} \\
&= \max_{a_s \in A_s} \Bigg\{ R(s, a_s) + \gamma \sum_{s' \in S} P_{ss'}(a_s) \cdot X[\bar{V}^*](s') - \Bigg(\bar{R}(\phi(s), \bar{\pi}^*(\phi(s))) + \\
&\qquad\qquad \gamma \sum_{\bar{s}' \in \bar{S}} \bar{P}_{\phi(s)\bar{s}'}(\bar{\pi}^*(\phi(s))) \cdot \bar{V}^*(\bar{s}') \Bigg) \Bigg\} \\
&\le \max_{a_s \in A_s} \Bigg\{ R(s, a_s) + \gamma \sum_{s' \in S} P_{ss'}(a_s) \cdot X[\bar{V}^*](s') - \Bigg(\bar{R}(\phi(s), \psi_s(a_s)) + \\
&\qquad\qquad \gamma \sum_{\bar{s}' \in \bar{S}} \bar{P}_{\phi(s)\bar{s}'}(\psi_s(a_s)) \cdot \bar{V}^*(\bar{s}') \Bigg) \Bigg\} \\
&= \max_{a_s \in A_s} \left\{ Q_{X(\bar{V}^*)}(s, a_s) - \bar{Q}_{\bar{V}^*}(\phi(s), \psi_s(a_s)) \right\} \\
&\le \max_{s \in S,\, a_s \in A_s} \left| Q_{X(\bar{V}^*)}(s, a_s) - \bar{Q}_{\bar{V}^*}(\phi(s), \psi_s(a_s)) \right| \\
&= K(\bar{V}^*)
\end{aligned}$$

and

$$
\begin{aligned}
X[\bar{V}^*](s) - B[X[\bar{V}^*]](s) \;&=\; \bar{V}^*(\phi(s)) - \max_{a_s \in A_s} \left\{ R(s, a_s) + \gamma \sum_{s' \in S} P_{ss'}(a_s) \cdot X[\bar{V}^*](s') \right\} \\
&=\; \bar{B}_{\bar{\pi}^*}[\bar{V}^*](\phi(s)) - \max_{a_s \in A_s} \left\{ R(s, a_s) + \gamma \sum_{s' \in S} P_{ss'}(a_s) \cdot X[\bar{V}^*](s') \right\} \\
&=\; \bar{R}(\phi(s), \bar{\pi}^*(\phi(s))) \;+\; \gamma \sum_{\bar{s}' \in \bar{S}} \bar{P}_{\phi(s)\bar{s}'}(\bar{\pi}^*(\phi(s))) \cdot \bar{V}^*(\bar{s}') \,- \\
&\qquad \max_{a_s \in A_s} \left\{ R(s, a_s) + \gamma \sum_{s' \in S} P_{ss'}(a_s) \cdot X[\bar{V}^*](s') \right\} \\
&\leq\; \bar{R}(\phi(s), \bar{\pi}^*(\phi(s))) \;+\; \gamma \sum_{\bar{s}' \in \bar{S}} \bar{P}_{\phi(s)\bar{s}'}(\bar{\pi}^*(\phi(s))) \cdot \bar{V}^*(\bar{s}') \,- \\
&\qquad \left\{ R(s, X[\bar{\pi}^*(\phi(s))]) + \gamma \sum_{s' \in S} P_{ss'}(X[\bar{\pi}^*(\phi(s))]) \cdot X[\bar{V}^*](s') \right\} \\
&=\; \bar{Q}_{\bar{V}^*}(\phi(s), \psi_s(X[\bar{\pi}^*(\phi(s))])) - Q_{X(\bar{V}^*)}(s, X[\bar{\pi}^*(\phi(s))]) \\
&\leq\; \left| Q_{X(\bar{V}^*)}(s, X[\bar{\pi}^*(\phi(s))]) - \bar{Q}_{\bar{V}^*}(\phi(s), \psi_s(X[\bar{\pi}^*(\phi(s))])) \right| \\
&\leq\; \max_{s \in S,\; a_s \in A_s} \left| Q_{X(\bar{V}^*)}(s, a_s) - \bar{Q}_{\bar{V}^*}(\phi(s), \psi_s(a_s)) \right| \\
&=\; K(\bar{V}^*)
\end{aligned}
$$

Therefore $\|X[\bar{V}^*] - B[X[\bar{V}^*]]\|_\infty \;\leq\; K(\bar{V}^*)$ and by proposition 4

$$
\|V^* - X[\bar{V}^*]\|_\infty \;\leq\; \frac{1}{1-\gamma} \cdot K(\bar{V}^*).
$$

$\square$

Theorem 10. *For the optimal value vector V^* of the original MDP M and the optimal value vector $\bar{V}^*$ and optimal policy $\bar{\pi}^*$ of the abstract MDP $\bar{M}$, we have*

$$\|V^* - V^{X[\bar{\pi}^*]}\|_\infty \;\leq\; \frac{2}{1-\gamma} \cdot K(\bar{V}^*).$$

Proof.

$$\begin{aligned} \|V^* - V^{X[\bar{\pi}^*]}\|_\infty \;&=\; \|V^* - X[\bar{V}^*] \;+\; X[\bar{V}^*] - V^{X[\bar{\pi}^*]}\|_\infty \\ &\leq\; \|V^* - X[\bar{V}^*]\|_\infty \;+\; \|X[\bar{V}^*] - V^{X[\bar{\pi}^*]}\|_\infty \end{aligned}$$

From theorems 8 and 9 we have

$$\|V^* - X[\bar{V}^*]\|_\infty \leq \frac{1}{1-\gamma} \cdot K(\bar{V}^*)$$

and

$$\|X[\bar{V}^*] - V^{X[\bar{\pi}^*]}\|_\infty \;=\; \|X[\bar{V}^{\bar{\pi}^*}] - V^{X[\bar{\pi}^*]}\|_\infty \;\leq\; \frac{1}{1-\gamma} \cdot K(\bar{V}^{\bar{\pi}^*}) \;=\; \frac{1}{1-\gamma} \cdot K(\bar{V}^*).$$

Therefore

$$\|V^* - V^{X[\bar{\pi}^*]}\|_\infty \;\leq\; \frac{2}{1-\gamma} \cdot K(\bar{V}^*).$$

□

Theorem 11. *For a given* $\epsilon \geq 0$*, if*

$$\phi(s) = \phi(\dot{s}) \;\Rightarrow\; |V^*(s) - V^*(\dot{s})| \;\leq\; \epsilon$$

for all $s, \dot{s} \in S$*, then* $\|V^* - X[\bar{V}^*]\|_\infty \;\leq\; \dfrac{\epsilon}{1-\gamma}$.

Proof. Based on the proof of theorem 1 (part b) in [58].

Let $\bar{V}$ be such that for all $\bar{s} \in \bar{S}$

$$\bar{V}(\bar{s}) = \max_{s \in \bar{s}} \; V^*(s) \; - \; \frac{\epsilon}{2}.$$

Then $\|V^* - X[\bar{V}]\|_\infty \;\leq\; \dfrac{\epsilon}{2}$.

The first step is to derive a bound on $\|\bar{B}[\bar{V}] - \bar{V}\|_\infty$. For all $\bar{s} \in \bar{S}$

$$\begin{aligned}
\left|\bar{B}[\bar{V}](\bar{s}) - \bar{V}(\bar{s})\right| \;&=\; \left|\sum_{s \in \bar{s}} W_{\bar{s}}(s) \left(\max_{a \in A_s} R(s,a) + \gamma \sum_{s' \in S} P_{ss'}(a) \cdot X[\bar{V}](s')\right) - \bar{V}(\bar{s})\right|. \\
&=\; \left|\sum_{s \in \bar{s}} W_{\bar{s}}(s) \left(B[X[\bar{V}]](s) - X[\bar{V}](s)\right)\right|. \\
&\leq\; \sum_{s \in \bar{s}} W_{\bar{s}}(s) \cdot \left\|B[X[\bar{V}]] - X[\bar{V}]\right\|_\infty. \\
&=\; \left\|B[X[\bar{V}]] - X[\bar{V}]\right\|_\infty. \\
&=\; \left\|B[X[\bar{V}]] - B[V^*] + B[V^*] - X[\bar{V}]\right\|_\infty. \\
&\leq\; \left\|B[X[\bar{V}]] - B[V^*]\right\|_\infty + \left\|V^* - X[\bar{V}]\right\|_\infty. \\
&\leq\; \gamma\; \left\|X[\bar{V}] - V^*\right\|_\infty + \left\|V^* - X[\bar{V}]\right\|_\infty. \\
&\leq\; (1+\gamma) \cdot \frac{\epsilon}{2}.
\end{aligned}$$

Combining this with proposition 4 in the appendix, we get

$$\|\bar{V}^* - \bar{V}\|_\infty \;\leq\; \frac{1}{1-\gamma} \cdot \|\bar{B}[\bar{V}] - \bar{V}\|_\infty \;\leq\; \frac{1}{1-\gamma} \cdot (1+\gamma) \cdot \frac{\epsilon}{2},$$

which implies

$$\begin{aligned}
\left\|V^* - X[\bar{V}^*]\right\|_\infty &= \left\|V^* - X[\bar{V}] + X[\bar{V}] - X[\bar{V}^*]\right\|_\infty . \\
&\leq \left\|V^* - X[\bar{V}]\right\|_\infty + \left\|X[\bar{V}] - X[\bar{V}^*]\right\|_\infty . \\
&= \left\|V^* - X[\bar{V}]\right\|_\infty + \left\|\bar{V} - \bar{V}^*\right\|_\infty . \\
&\leq \frac{\epsilon}{2} + \frac{1+\gamma}{1-\gamma} \cdot \frac{\epsilon}{2}. \\
&= \frac{\epsilon}{1-\gamma}.
\end{aligned}$$

□

Theorem 12. *For a given $\epsilon \geq 0$, if*

$$\phi(s) = \phi(\dot{s}) \;\Rightarrow\; |V^*(s) - V^*(\dot{s})| \;\leq\; \epsilon$$

for all $s, \dot{s} \in S$, then

$$\|V^* - V^{\pi_{X[\bar{V}^*]}}\|_\infty \;\leq\; \frac{2\epsilon\gamma}{(1-\gamma)^2}.$$

Proof. Based on the proof of theorem 1 (part c) in [58].

The definition of $\pi_{X[\bar{V}^*]}$ is

$$\pi_{X[\bar{V}^*]}(s) \;\in\; \arg\max_{a \in A_s} \; R(s,a) + \gamma \sum_{s' \in S} P_{ss'}(a_s) \cdot X[\bar{V}^*](s').$$

for all $s \in S$.

$$\begin{aligned}
\|V^* - V^{\pi_{X[\bar{V}^*]}}\|_\infty \;&=\; \left\|V^* - B[X[\bar{V}^*]] + B[X[\bar{V}^*]] - V^{\pi_{X[\bar{V}^*]}}\right\|_\infty . \\
&=\; \left\|B[V^*] - B[X[\bar{V}^*]] + B_{\pi_{X[\bar{V}^*]}}[X[\bar{V}^*]] - B_{\pi_{X[\bar{V}^*]}}[V^{\pi_{X[\bar{V}^*]}}]\right\|_\infty . \\
&\leq\; \left\|B[V^*] - B[X[\bar{V}^*]]\right\|_\infty + \left\|B_{\pi_{X[\bar{V}^*]}}[X[\bar{V}^*]] - B_{\pi_{X[\bar{V}^*]}}[V^{\pi_{X[\bar{V}^*]}}]\right\|_\infty . \\
&\leq\; \gamma \;\left\|V^* - X[\bar{V}^*]\right\|_\infty + \;\gamma\; \left\|X[\bar{V}^*] - V^{\pi_{X[\bar{V}^*]}}\right\|_\infty . \\
&\leq\; \gamma \;\left\|V^* - X[\bar{V}^*]\right\|_\infty + \;\gamma\; \left\|X[\bar{V}^*] - V^* + V^* - V^{\pi_{X[\bar{V}^*]}}\right\|_\infty . \\
&\leq\; \gamma \;\left\|V^* - X[\bar{V}^*]\right\|_\infty + \;\gamma\; \left\|X[\bar{V}^*] - V^*\right\|_\infty + \;\|V^* - V^{\pi_{X[\bar{V}^*]}}\|_\infty . \\
&\leq\; \frac{2\epsilon\gamma}{1-\gamma} + \gamma \;\|V^* - V^{\pi_{X[\bar{V}^*]}}\|_\infty .
\end{aligned}$$

Therefore

$$\|V^* - V^{\pi_{X[\bar{V}^*]}}\|_\infty \;\leq\; \frac{2\epsilon\gamma}{(1-\gamma)^2}.$$

□

Bibliography

[1] David Abel, Ellis D. Hershkowitz, and Michael L. Littman. Near Optimal Behavior via Approximate State Abstraction. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning*, volume 48, pages 2915–2923, 2016.

[2] Ankit Anand, Ritesh Noothigattu, Parag Singla, et al. OGA-UCT: On-the-go Abstractions in UCT. In *Proceedings of the 26th International Conference on Automated Planning and Scheduling*, 2016.

[3] Thomas Anthony, Zheng Tian, and David Barber. Thinking Fast and Slow with Deep Learning and Tree Search. In *Advances in Neural Information Processing Systems 30*, pages 5364–5374, 2017.

[4] Dimitri P. Bertsekas. Feature-based Aggregation and Deep Reinforcement Learning: A Survey and some new Implementations. *IEEE CAA Journal of Automatica Sinica*, 6(1):1–31, 2019.

[5] Dimitri P. Bertsekas and David A. Castanon. Rollout Algorithms for Stochastic Scheduling Problems. *Journal of Heuristics*, 5(1):89–108, 1999.

[6] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[7] Dimitri P Bertsekas, John N Tsitsiklis, and Cynara Wu. Rollout Algorithms for Combinatorial Optimization. *Journal of Heuristics*, 3(3):245–262, 1997.

[8] Blai Bonet and Hector Geffner. Action selection for MDPs: Anytime AO* versus UCT. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2012.

[9] 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.

[10] Olivier Buffet and Douglas Aberdeen. The Factored Policy-gradient Planner. *Artificial Intelligence*, 173(5):722–747, 2009.

[11] Tristan Cazenave. Nested Monte-Carlo Search. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, 2009.

[12] Hyeong Soo Chang, Robert Givan, and Edwin K. P. Chong. Parallel Rollout for Online Solution of Partially Observable Markov Decision Processes. *Discrete Event Dynamic Systems*, 14(3):309–341, 2004.

[13] Hao Cui, Roni Khardon, Alan Fern, and Prasad Tadepalli. Factored MCTS for Large Scale Stochastic Planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 3261–3267, 2015.

[14] Tomás De La Rosa, Sergio Jiménez Celorrio, and Daniel Borrajo. Learning Relational Decision Trees for Guiding Heuristic Planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pages 60–67, 2008.

[15] Janardhan Rao Doppa, Alan Fern, and Prasad Tadepalli. Structured Prediction via Output Space Search. *The Journal of Machine Learning Research*, 15(1):1317–1350, 2014.

[16] Alan Fern, Sung Wook Yoon, and Robert Givan. Approximate Policy Iteration with a Policy Language Bias: Solving Relational Markov Decision Processes. *Journal of Artificial Intelligence Research*, 25:75–118, 2006.

[17] William D Harvey and Matthew L Ginsberg. Limited Discrepancy Search. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 607–615, 1995.

[18] Jesse Hoey, Robert St-Aubin, Alan Hu, and Craig Boutilier. SPUDD: Stochastic Planning using Decision Diagrams. In *Proceedings of the Conference in Uncertainty in Artificial Intelligence*, 1999.

[19] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation Learning: A Survey of Learning Methods. *ACM Computing Surveys*, 50(2):21, 2017.

[20] Ilse CF Ipsen and Carl D Meyer. The Idea behind Krylov Methods. *American Mathematical Monthly*, pages 889–899, 1998.

[21] Murugeswari Issakkimuthu and Alan Fern. Online Policy Improvement for Probabilistic Planning: Benchmarks and Baselines. In *Workshop on the International Planning Competition (WIPC) at the 31st International Conference on Planning and Scheduling*, 2021.

[22] Murugeswari Issakkimuthu, Alan Fern, Roni Khardon, Prasad Tadepalli, and Shan Xue. Hindsight Optimization for Probabilistic Planning with Factored Actions. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pages 120–128, 2015.

[23] Murugeswari Issakkimuthu, Alan Fern, and Prasad Tadepalli. Training Deep Reactive Policies for Probabilistic Planning Problems. In *Proceedings of the 28th International Conference on Automated Planning and Scheduling*, 2018.

[24] Murugeswari Issakkimuthu, Alan Fern, and Prasad Tadepalli. The Choice Function Framework for Online Policy Improvement. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence*, 2020.

[25] Nan Jiang, Alex Kulesza, and Satinder P. Singh. Abstraction Selection in Model-based Reinforcement Learning. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37, pages 179–188, 2015.

[26] Sergio Jiménez, Tomás De la Rosa, Susana Fernández, Fernando Fernández, and Daniel Borrajo. A Review of Machine Learning for Automated Planning. *The Knowledge Engineering Review*, 27(4):433–467, 2012.

[27] Michael Kearns, Yishay Mansour, and Andrew Y. Ng. A Sparse Sampling Algorithm for near-optimal Planning in Large Markov Decision Processes. *Machine Learning*, 49(2-3):193–208, 2002.

[28] Thomas Keller and Patrick Eyerich. PROST : Probabilistic Planning based on UCT. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling*, 2012.

[29] Roni Khardon. Learning Action Strategies for Planning Domains. *Artificial Intelligence*, 113(1-2):125–148, 1999.

[30] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement Learning in Robotics: A Survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[31] Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo Planning. In *Machine Learning: ECML 2006*, pages 282–293, 2006.

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

[33] Francisco Javier Larrosa Bondia, Emma Rollón Rico, and Rina Dechter. Limited Discrepancy AND/OR Search and its Application to Optimization Tasks in Graphical Models. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence*, pages 617–623, 2016.

[34] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning Hand-eye Coordination for Robotic Grasping with Deep Learning and Large-scale Data Collection. *The International Journal of Robotics Research*, 2016.

[35] Mario Martin and Hector Geffner. Learning Generalized Policies in Planning Domains using Concept Languages. In *Principles of Knowledge Representation and Reasoning*, 2000.

[36] S. Minton. *Machine Learning Methods for Planning.* Morgan Kaufmann, 1993.

[37] S. Minton, J. Carbonell, C. A. Knoblock, D. R. Kuokka, O. Etzioni, and Y. Gil. Explanation-based Learning: A Problem Solving Perspective. *Artifical Intelligence Journal*, 40:63–118, 1989.

[38] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level Control through Deep Reinforcement Learning. *Nature*, 518(7540):529–533, 2015.

[39] Truong-Huy Dinh Nguyen, Tomi Silander, Wee-Sun Lee, and Tze-Yun Leong. Bootstrapping Simulation-based Algorithms with a Suboptimal Policy. In *Proceedings of the 24th International Conference on Planning and Scheduling*, 2014.

[40] Marek Petrik. An Analysis of Laplacian Methods for Value Function Approximation in MDPs. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 2574–2579, 2007.

[41] Jervis Pinto and Alan Fern. Learning Partial Policies to Speedup MDP Tree Search via Reduction to IID Learning. *The Journal of Machine Learning Research*, 18(1):2179–2213, 2017.

[42] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley & Sons, Inc., 1st edition, 1994.

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

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

[45] Aswin Raghavan, Roni Khardon, Prasad Tadepalli, and Alan Fern. Memory-Effcient Symbolic Online Planning for Factored MDPs. In *Conference on Uncertainty in Artificial Intelligence*, pages 732–741, 2015.

[46] Balaraman Ravindran and Andrew G. Barto. Approximate Homomorphisms: A Framework for Non-Exact Minimization in Markov Decision Processes. In *International Conference on Knowledge based Computer Systems*, 2004.

[47] Stéphane Ross and Drew Bagnell. Efficient Reductions for Imitation Learning. In *International Conference on Artificial Intelligence and Statistics*, pages 661–668, 2010.

[48] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd edition, 1994.

[49] Scott Sanner. Relational Dynamic Influence Diagram Language (RDDL) : Language Description. 2010.

[50] Jost Schatzmann, Karl Weilhammer, Matt Stuttle, and Steve Young. A Survey of Statistical User Simulation Techniques for Reinforcement-Learning of Dialogue Management Strategies. *The Knowledge Engineering Review*, 21(2):97–126, 2006.

[51] I. Serban, Chinnadhurai Sankar, M. Pieper, Joelle Pineau, and Yoshua Bengio. The Bottleneck Simulator: A Model-based Deep Reinforcement Learning Approach. *Journal of Artificial Intelligence Research*, 69:571–612, 2020.

[52] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature*, 529(7587):484–489, 2016.

[53] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A General Reinforcement Learning Algorithm that Masters Chess, Shogi, and Go through Self-play. *Science*, 362(6419):1140–1144, 2018.

[54] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy P. Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the Game of Go without Human Knowledge. *Nature*, 550(7676):354–359, 2017.

[55] Alexander L. Strehl and Michael L. Littman. An Analysis of Model-based Interval Estimation for Markov Decision Processes. *Journal of Computer and System Sciences*, 74(8):1309–1331, 2008.

[56] Gerald Tesauro and Gregory R. Galperin. Online Policy Improvement using Monte-Carlo Search. In *Advances in Neural Information Processing Systems*, 1997.

[57] Sam Toyer, Felipe Trevizan, Sylvie Thiébaux, and Lexing Xie. Action Schema Networks: Generalised Policies with Deep Learning. In *AAAI Conference on Artificial Intelligence*, 2018.

[58] John N. Tsitsiklis and Benjamin Van Roy. Feature-based Methods for Large Scale Dynamic Programming. *Machine Learning*, 22(1-3):59–94, 1996.

[59] Thomas J. Walsh, Sergiu Goschin, and Michael L. Littman. Integrating Sample-based Planning and Model-based Reinforcement Learning. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence*, pages 612–617, 2010.

[60] Ward Whitt. Approximations of Dynamic Programs, I. *Mathematics of Operations Research*, 3(3):231–243, 1978.

[61] Xiang Yan, Persi Diaconis, Paat Rusmevichientong, and Benjamin V Roy. Solitaire: Man versus Machine. In *Advances in Neural Information Processing Systems*, pages 1553–1560, 2005.

[62] SungWook Yoon, Alan Fern, and Robert Givan. Inductive Policy Selection for First-Order MDPs. In *Conference on Uncertainty in Artificial Intelligence*, 2002.

[63] Terry Zimmerman and Subbarao Kambhampati. Learning-Assisted Automated Planning: Looking back, Taking stock, Going forward. *AI Magazine*, 24(2)(2):73–96, 2003.