AN ABSTRACT OF THE DISSERTATION OF

Sean McGregor for the degree of Doctor of Philosophy in Computer Science
presented on June 16, 2017.

Title: Machine Learning Methods for Public Policy: Simulation, Optimization, and Visualization

Abstract approved: ______________________________

Thomas G. Dietterich

Society faces many complex management problems, particularly in the area of shared public resources such as ecosystems. Existing decision making processes are often guided by personal experience and political ideology rather than state-of-the-art scientific understanding. This dissertation envisions a future in which multiple stakeholders are provided with computational tools for formalizing their management preferences and computing optimal solutions based on state-of-the-art computational simulations. To make this vision a reality, advances are required in optimization and visualization, and this dissertation presents research on both topics within the formalism of the Markov decision process (MDP). First, it describes an interactive visualization system for understanding the MDP under user-defined management policies, reward functions, and transition dynamics. Second, it presents a method for optimizing management policies for the user-parameterized

MDPs. The research is illustrated and validated using a combination of benchmark MDPs and an application to the management of wildfire in ponderosa pine forests. For the wildfire problem, an excellent high-fidelity model of forest growth and wildfire behavior is employed. However, this model is extremely slow, which prevents interactive visualization and optimization. To address simulation computational expense, the dissertation also presents a method for creating a fast surrogate model and shows that this model is sufficiently accurate to support policy optimization and visualization.

Machine Learning Methods for Public Policy: Simulation, Optimization, and Visualization

by

Sean McGregor

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented June 16, 2017
Commencement June 2018

ACKNOWLEDGEMENTS

My advisor, Professor Thomas Dietterich, deserves recognition for his unwavering drive to make the world better through research. He was a constant resource and collaborator for solving the hard problems and making a difference. I would also like to thank my committee members, Ronald Metoyer, Prasad Tadepalli, Margaret Burnett, and John Bolte for the time and effort they contributed. Ron's guidance was critical in contributing to the field of visualization. My meetings and classes with Prasad and Margaret always showed new directions of thought that enriched my research immensely.

My collaborators in forestry economics, especially Claire Montgomery and Rachel Houtman, deserve special praise for their efforts at crossing disciplinary lines. Rachel was always keen to learn new things from the world of computing, and was eager to teach me about forestry.

I would like to thank my friends and family. Your love and kindness celebrated my successes and buoyed me through challenges. In particular, I would like to thank both my parents, the Buchanans, and Ashley Bromley for always making me feel at home. Finally, my Great Uncle Ted de Werd's support of the family's education enabled a whole generation of nieces and nephews to prosper and contribute to the world. I thank him for his support of my educational and research efforts.

This material is based upon work supported by the National Science Foundation under grant numbers 1331932 and 0832804. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

CONTRIBUTION OF AUTHORS

This work updates and brings together three manuscripts [38, 41, 40]. Each manuscript benefited from the guidance and editing of the co-authors, including Thomas Dietterich, Ronald Metoyer, Claire Montgomery, Rachel Houtman, and Hailey Buckingham. Additionally, Rachel Houtman deserves particular credit for her work generating data from the computationally expensive wildfire simulator. Similarly, Hailey Buckingham deserves credit for her work with the wildfire simulator tested for the evaluation of the MDP visualization.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF TABLES

LIST OF APPENDIX TABLES

Machine Learning Methods for Public Policy: Simulation, Optimization, and Visualization

1 Introduction

Recent advances in artificial intelligence technology are making it possible to create systems that automatically (or collaboratively) make decisions in natural resource management [27, 16], infrastructure investment [36], law enforcement [67], and other domains of public policy. A theoretical formulation for these and other problems of public policy optimization is the Markov Decision Process (MDP). In an MDP, the state of the world evolves stochastically from one state to another depending on the action chosen at each time step. A scalar reward is received at each time step depending on the simulated state and the chosen action. An MDP is solved by finding a decision making rule (policy) that maximizes the long-term sum of rewards.

Some MDPs are small enough to be solved by exact algorithms [8], but most MDPs of practical interest must be solved using Monte Carlo (MC) methods. In these cases, the standard approach is to implement a software simulation of the MDP and apply a MC optimizer to find a near-optimal policy. When simulators are simplified versions of the real world, it is possible that the specification, implementation, or optimization of the MDP simulator will result in a bad policy recommendation, i.e. one lacking ecological validity. Since inappropriate public policy decisions can bring about a variety of harms, it is necessary to develop methods for testing public policy MDPs and explaining their recommendations. Further, since public policy recommendations need to balance divergent stakeholder interests, such as economic development, ecological diversity, etc., adopting machine-learned policies in the real world requires facilitating compromise. This thesis centers its three contributions around testing public policy MDPs, explaining recommendations, and supporting multi-stakeholder negotiations. We motivate and evaluate our contributions through a wildfire suppression MDP.

Our first study asks, how can we discover problems in the specification or implementation of the MDP (i.e., test for "bugs")? For the simulator defining the wildfire suppression MDP, a single 100 year trajectory is 1.7 gigabytes and the optimization algorithm can process many terabytes of trajectories. Intelligently testing such large datasets requires tools supporting exploration and comparison to user expectations. For this purpose, we introduce a visual analytic system for MDPs: MDPvis. Our first study introduces MDPvis in the context of the wildfire suppression simulator and generalizes its design for the class of all simulator-defined MDPs.

The wildfire simulator highlights the challenge of integrating MDP simulators with visual analytic systems. Generating a single 100-year trajectory of wildfire simulations takes up to 7 hours to complete, which makes the MDP exploration process non-interactive. This thesis addresses the simulation time issue with its second manuscript by extending Model-Free Monte Carlo (MFMC) [23] to exploit independencies between simulator variables. Our algorithm successfully generates complete state-action histories for the wildfire problem in less than a second.

Finally, since different stakeholders in public policy problems will have different reward functions, we desire a method for interactively testing and analyzing how changing the rewards leads to changes in the optimized policy function. Thus we require an optimization algorithm that is both fast and adaptable to a variety of MDPs. Our third manuscript is a method for searching over policy functions quickly after the user issues a reward query. We show the optimization method performs well for a variety of reward functions on the wildfire suppression problem.

1.1 Contributions

This work makes contributions to the fields of forestry and computer science. In forestry, our modeling and optimization results are the first instance of successfully optimizing 100-year, full-fidelity fire suppression policies. Further, our approach of supporting the optimization of multiple reward functions within a visual analytic system provides a novel approach for exploring the effect of rewards for non-market resources (like clean air, ecological diversity, and recreation).

Our computer science contributions address problems within visualization, surrogate modeling, and optimization. MDPvis is the first visualization targeting the MDP formulation. Our extension to the MFMC algorithm gives the first case of state space factorization for MFMC. The MFMC surrogate enables the first case of optimizing a decision tree policy function, which we accomplish with the SMAC global optimization procedure. Finally, our chosen optimization algorithm is the first application of random forests as a direct policy search model. Collectively these contributions define a visual analytic environment supporting the testing and policy analysis of high-dimensional MDPs.

1.2 Organization

These contributions are presented in the following three chapters. First, we introduce MDPvis as our MDP visualization system. Next we introduce MFMCi as our surrogate modeling technique that speeds up simulation by several orders of magnitude. Finally, we introduce an interactive-speed optimization technique with the final manuscript.

2 An Interactive Visualization for Testing Markov Decision Processes: MDPVIS

2.1 Introduction

Policies for Markov Decision Processes (MDPs) are selected by optimization systems that are subject to failures of specification, implementation, integration, and optimization. Since the system stochastically expresses and hides these failures (referred to as "bugs"), testing requires exploration. Visual analytics is a common approach to exploration that combines automated analysis techniques with interactive visualizations for understanding, reasoning, and decision making [30]. We introduce a visual analytic system (see Figure 2.2), MDPVIS, to support the MDP testing process.

The design of MDPVIS tests for bugs whose underlying cause can be ascribed to several different components. For example, Andrew Ng relays an unusual solution to a soccer playing MDP [45]. The developer of the MDP created a reward function that gave the soccer agent a reward for touching the ball under the theory that possession time is associated with scoring goals. A reinforcement learning algorithm was applied to find a decision-making policy to optimize this reward. The resulting policy exhibited a surprising behavior: Instead of using ball possession to advance down the field, the agent stood by the ball and began to "vibrate" to produce the maximum number of ball touches. This bug can be viewed as a problem in specification (the agent should not be rewarded for touching the ball), implementation (the agent should not be able to vibrate next to the ball), integration (the frequency of reward granted by the transition function for ball touches is too high), and optimization (the optimizer may have failed to find the true optimal policy). The multitude of possible MDP bugs and causes give rise to a highly iterative development process.

During our design process for MDPvis, we conducted a series of semi-structured interviews [54] with MDP researchers to elicit current practices for MDP development. We found a variety of ad hoc testing systems in support of an “informed trial and error” [52] process whereby MDP practitioners iteratively explore the parameter space. MDP practitioners generally first write an interactive client to manually execute transitions, followed by a visualization of state evolution as a policy rule is followed. None of the researchers we interviewed use a generic tool supporting this process. We hypothesize this is because researchers have heretofore not had access to a visualization they can easily connect to their MDP simulator and MDP optimizer.

The MDPvis system makes three contributions. First, it introduces a domain independent protocol by which the visualization system can interact with the MDP optimizer and MDP simulator. Second, it provides a visual interface that supports the data analysis tasks that problem domain experts, software developers, and optimization researchers need to perform. Third, it provides an interface by which the users can interact with the MDP simulator and optimizer to define different settings of the MDP parameters and compare their effects on the resulting behavior of the system.

These three contributions relate to the three challenges identified by Sedlmair et al. [52]: the data acquisition gap (getting the data into the visualization tool), the data analysis gap (helping the user visualize the data), and the cognition gap (helping the user uncover important behavior embedded within high-dimensional systems).

Software testing is a subfield of software engineering that includes more precise definitions of testing and bugs. In this paper, we take a high-level view of testing. In particular, we define testing as the "...execution of a program with the intent to produce some problems - especially a failure" [66]. These failures are generally called "bugs."

In other words, developers test software for bugs by comparing the results of execution to the expected results. It is clear from this definition that testing requires the developer to 1) associate program output with program input in order to create test cases and 2) compare actual outputs to expected outputs. Our visualization tool supports these two tasks within the context of MDPs.

Our design process for MDPvis followed Munzner's nested model [44], which includes steps for characterizing the problem domain and designing visual encodings. Our starting point in the design process was a challenging large state space MDP for wildfire suppression policy [27]. In the wildfire problem the goal is to find a policy minimizing the cost of fire suppression and smoke inhalation, while maximizing the rewards from timber harvest, ecological diversity, and recreation.

We evaluate the effectiveness of MDPvis in the context of a simplified version of the wildfire simulator, and we show the generality of MDPvis by integrating it with additional domains common in the MDP researcher community.

In the following sections we formally introduce MDPs with their Data and Task Abstraction, detail the contexts where users test and debug MDPs in section 2.3, review the optimization visualization literature in section 2.4, give our visual encoding for MDPs in section 2.5, and present the MDPvis prototype in section 2.6.

2.2 Data and Task Abstraction

While several different MDP formulations are used in the literature, there is a de facto standard formulation from which other formulations are viewed as specializations. The standard formulation is the infinite horizon discounted Markov Decision Process (MDP) with a designated start state distribution [7, 48] $\mathcal{M} = \langle S, A, P, R, \gamma, P_0 \rangle$. S is a finite set of states of the world; A is a finite set of possible actions that can be taken in each state; $P : S \times A \times S \mapsto [0, 1]$ is the conditional probability of entering state s' when action a is executed in state s; $R(s, a)$ is the reward received after performing action a in state s; $\gamma \in (0, 1)$ is the discount factor, and P_0 is the distribution over starting states. Generally the goal for optimizing an MDP is to find a policy, π, that selects actions maximizing the expected discounted sum of rewards of the MDP. For convenience we also define $P_{n|\pi}$ to be the distribution of states at time n when following policy π.

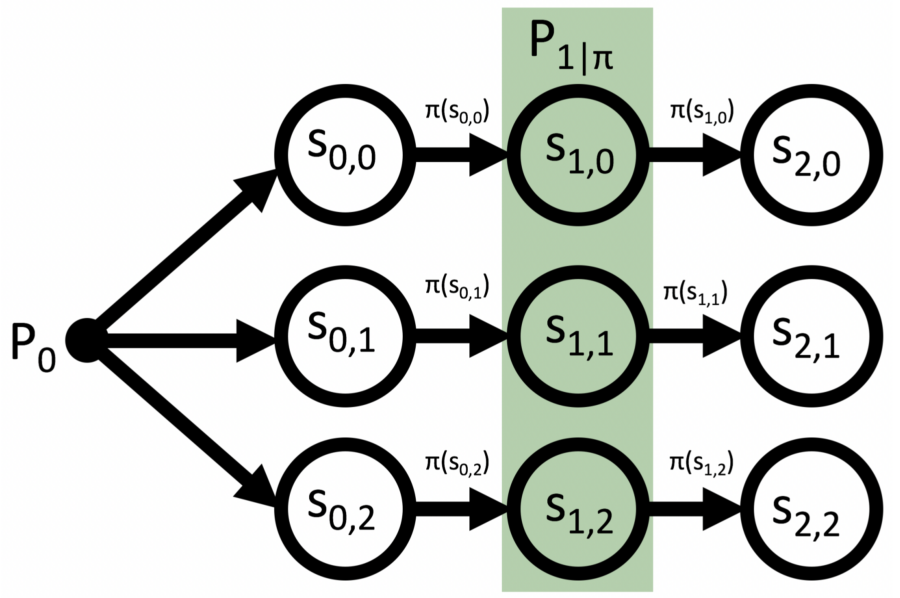


Figure 2.1: A set of three trajectories generated starting at states drawn from the initial state distribution (P_0) and transitioned between states $s_{i,j}$ where i is the time step and j is the trajectory identifier. The current policy ($\pi(s_{i,j})$) selects actions until a trajectory depth of 3. The initial states and all subsequent states are defined on a set of quantitative and categorical variables. Each state transitions to resulting states by evaluating the transition function until reaching the time horizon or terminating state. The transition function draws the resulting states from a distribution that is a function of the current state and the action selected by the policy function. We highlight a set of states drawn from the distribution of states at a particular time horizon under policy π. We label this set $P_{1,\pi}$ for time step 1 under policy π.

When we execute a fixed policy π for T steps starting in a state drawn from P_0, we produce a T-step trajectory through the state space. We will refer to this as a Monte Carlo trajectory. Our approach to visualization generates a collection of T-step trajectories (see Figure 2.1) and then visualizes them over time. These trajectories are the output of the system under test, and their distribution is controlled by the parameters of the MDPs' constituent functions.

Sedlmair et al. [52] label techniques for understanding the relationship between input parameters and outputs as Parameter Space Analysis (PSA), "...the systematic variation of model input parameters, generating outputs for each combination of parameters, and investigating the relation between parameter settings and corresponding outputs." This is a suitable definition for the MDP testing process since trajectories are typically produced by changing the parameters of the policy, transition, and reward functions.

Table A.1 in Appendix A give a series of testing questions derived from experience optimizing for the wildfire suppression policy domain and from interviewing MDP algorithm researchers not involved in the wildfire policy project. The table classifies these questions according to the tasks of Sedlmair et al. for visual parameter space analysis: *fitting, outliers, partitioning, optimization, uncertainty, and sensitivity.* In several cases we broaden the definition from Sedlmair et al. to fit the scope of MDPs.

Fitting, *"Where in the input parameter space would actual measured data occur?"*: In many applications, the MDP simulator simulates real-world phenomena. While we typically do not have access to real-world validated data across the entirety of the state and action space, we do often have state transition data for a subset of the parameter space. Further, while the MDP practitioner may not have access to ground truth data for their system, they can often identify when the system is producing unrealistic outcomes. See Table A.1 for fitting questions.

Outliers, *"What outputs are special?"*: Sedlmair et al. break the outlier task into learning from the outliers of a model and checking the plausibility of the more extreme cases. The outliers are particularly important during the testing phase of MDP development because an optimization algorithm moves through many potential policy functions while searching for an optimal policy. Each of these outliers may push the MDP towards more extreme, and potentially less plausible, states or state transitions. See Table A.2.

Partition, *"How many different types of model behaviors are possible?"*: Here Sedlmair et al. focus on relating partitions of the output space to the inputs. Testing an MDP is less concerned with what types of behaviors are possible, but instead focuses on whether the human tester's mental partition of inputs produces the expected partition of outputs. Thus we expand Sedlmair et al.'s definition to include both the partitioning of inputs and outputs. A tester will generate trajectories for one partition of the parameter space, compare it to a different partition, and check whether the differences in results match expectations. In interactive visualization for MDPs, many partitioning tasks involve filtering trajectories (defined here as interactively subsetting the trajectories) to look at more specific cases or generating additional trajectories under new parameters. These steps are often part of other PSA tasks, as can be seen in the overlap of Table A.3 with other PSA tasks.

Optimization, *"Find the best parameter combination given some objectives."*: We expand on this definition to also include testing the optimization algorithm itself. When testing new optimization algorithms on complex MDPs, it is often not obvious whether the algorithm is stuck in a local optimum, has exploited a bug in the simulator, or got stuck in a local optimum. It is important to have tools that enable the optimization researcher and domain expert to collaborate towards improving the optimizer and the specification of the MDP. See Table A.4.

Uncertainty, *"How reliable is the output?"*: MDPs present aleatoric/statistical uncertainty in the transition and reward functions. The stochasticity may make the optimizer uncertain which actions are optimal. See Table A.5.

Sensitivity, *"What ranges/variations of outputs to expect with changes of input?"*: Sedlmair et al.'s definition of sensitivity shows the fluidity of PSA tasks, since it touches all the other PSA tasks in one respect or another. A particularly important aspect for MDPs is sensitivity of the policy to changes in the reward function to find why a learned policy is deemed near-optimal. If a learned policy is not stable within its neighborhood of similar reward functions, it is likely that the policy is not one that should be relied on in the real-world. See Table A.6 for cross-cutting questions.

Section 2.5.3 has additional MDP testing discussion in the context of visuals from MDPvis. Next we describe the roles that may be filled by users of MDP visualization.

2.3 User Roles

We identify four MDP development roles and give examples of these roles in the context of wildfire suppression and autonomous helicopters.

Policy stakeholders: MDP-derived policies often affect people that are not party to the MDP optimization process. In the wildfire domain, this stakeholder could be a home owner in the region modeled by the MDP. For an autonomous helicopter, this stakeholder could be a person purchasing a finished helicopter product. Our visualization in section 2.6 does not target this population because it limits the design options for the following three groups.

Non-programmer domain experts: The MDP formulation can be applied to many problems for which a non-programmer is the authority on the simulated phenomena. A domain expert in the wildfire domain could be a US Forest Service land manager tasked with writing timber harvest and fire suppression policies. In the autonomous helicopter domain, a domain expert could be a professional RC stunt pilot. Visualizations help these domain experts validate the MDP by viewing the output of the system without needing to directly interact with code or terabytes of raw output. Here the domain expert provides input to the software team regarding where the current system is producing unrealistic results.

Software developers: Software developers may develop complex MDPs from a specification written by domain experts. In the wildfire setting, developers collaborate with several domain experts from silviculture, fire modeling, and economics to produce a model integrating all these disciplines. In the autonomous helicopter domain, the developers may collaborate with domain experts from mechanical engineering, computational physics, aviation, and the RC community. Here software developers may use visualization to test against expectations, or use visualization as a collaboration tool to share simulations and policies with domain experts.

Optimization researchers: After constructing an MDP, the optimization researcher finds policies by applying existing optimization algorithms or developing new optimization algorithms. In either case, the complexities of representing a problem domain in a form that can be efficiently optimized requires optimization experience. The optimization researchers may use visualization to test the correctness and performance of their algorithms. In both the wildfire and autonomous helicopter domains, optimization researchers will typically have formal training in optimization methods but little grounding in the problem domain.

In real-world settings, these roles are not well-defined. Each role can be filled by a single person or by a large team of developers and domain experts. MDPvis' target users are domain experts who are willing to spend time learning the MDP formulation, software developers tasked with developing an MDP, and optimization researchers tasked with optimizing a policy for the MDP.

2.4 Related Work

Many problems have been formulated as MDPs, including domains as diverse as RC car control [2, 14], invasive species management [16], and real time strategy games [61].

While no general visualization for large MDPs has heretofore been proposed covering all these domains, there are numerous works that could be viewed as visualization for more restricted classes of MDPs.

Several works present systems for exploring decisions at a single time step. Broeksema et al. [13] give a decision analysis tool to examine recommendations made by an expert system. Decisions are plotted as Voronoi diagrams by means of Multiple Correspondence Analysis (MCA), which is a version of Multidimensional Scaling (MDS). The Voronoi diagrams lack a comprehensible coordinate system in two dimensions, but adjacency of attributes plotted over the diagram show how the decision variable changes as other attributes vary.

MDP policies are often specified via learned classifiers. Effectively testing and debugging classifiers is an active area of research, but published research does not address MDP policy classifiers. Groce et al. [26] explore methods for prioritizing classified data points for user inspection. Once a datapoint is selected the end user can decide whether they agree with its classification. The user debugs the classifier by correcting data labels, which leads to an update of the model. This debugging strategy assumes that the only form of error was an incorrect data label. In MDPs, however, there are no labels on the data, and the central testing question is whether the simulator and policy are generating the right data to begin with.

Kulesza et al. [31] includes a classifier debugging system for email messages. The user provides model correction through an interactive bar chart for a naive Bayes model. Kulesza et al. selected naive Bayes for its interpretability since it has a natural visual representation that end users can understand without formal training in machine learning.

Migut and Worring [42] compose several information visualizations into a visual analytic dashboard for exploring a dichotomous choice as determined by a machine learning classifier. The system does not examine multiple sequential timesteps.

In ensemble visualization, the goal is to provide a compact representation of many predictive models of a singular ground truth [47]. Uncertainty in the predicted result is reduced by viewing a visualization of model agreement. In contrast, the uncertainty in MDP visualization arises from the stochastic responses of the world as the agent acts upon it. Ensemble visualization requires building consensus, but MDP visualization requires exploring the complete set of the world's potential responses to a policy.

Simulation steering is one branch of visualization that attempts to bring the user into a optimization process by allowing the user to select actions at each timestep as the simulator executes. Simulation steering for epidemic response decisions presented by Afzal et al. [3] shows individual outcomes through time. The user can change decisions at various points along a future trajectory to see how the mortality rate responds. This visualization supports user-based optimization for a deterministic MDP.

Waser et al. [58] give another simulation steering visualization, "World Lines," that invites users to control emergency response for flooding events. This visualization generates a small set of alternative futures based on an action in the present. Subsequent versions of World Lines [60, 49, 50, 59] support stochasticity through secondary simulation controls (random levee breach locations) on the probabilistic parameters of the model.

In contrast to the current approaches in the literature that attempt to give the best visualization possible for a particular problem domain, our approach defines the visualization in terms of a formal definition of sequential decision making problems. While our analysis focuses on the core MDP formulation, it is possible to use MDPvis with broader classes of MDPs, including ones with *partial observability*, *continuous-time*, *continuous-actions*, and *multiple agents*.

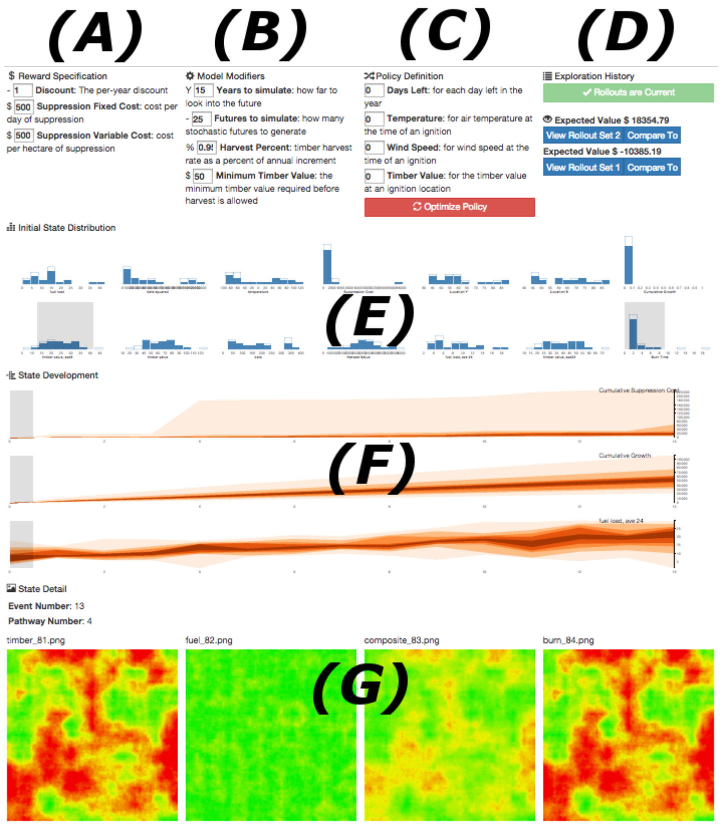
Simulator-defined *Partially observable MDPs* (POMDPs) hide state variables from the optimizer. For example, a helicopter simulator will simulate wind effects on the helicopter's position while not giving the optimizer access to wind data because a physical helicopter cannot observe the velocity of wind impacting its frame. When optimizing POMDPs from Monte Carlo trajectories, the simulator will strip hidden variables after they are simulated or add noise to the observations available to the optimizer. While this is necessary for faithfully simulating the operational conditions of the policy, for purposes of testing and debugging, it is useful to visualize the values of the hidden variables as well. We would also want to be able to visualize the belief state of the POMDP controller, perhaps in terms of a sample of points.

In *continuous-time MDPs*, the action can be selected at any time scale. For instance, in a helicopter domain the rotor angles can change at any fraction of a second. MDPvis assumes actions occur at consistent time scales, so to visualize continuous-time domains it is necessary to discretize time. In the helicopter domain, this would take the form of selecting a sampling frequency for saving the state and action information in the simulator. MDPvis handles *continuous actions*, such as changing the angle of the rotor blade, by treating them as a continuous variable.

Multi-agent MDPs model the interaction of multiple decision makers. Each decision maker can have its own policy function, but the MDP simulator gathers the actions chosen by the multiple agents into a single action vector, simulates the transition to the next state, and computes the reward signal for the agents. In a helicopter domain, this could take the form of several helicopters lifting cut timber after harvest. MDPvis can support multi-agent MDPs by having separate variables and parameters for each agent, but a visualization intended specifically for multi-agent MDPs would be valuable future work.

2.5 Visual Encoding of MDPs

Our visualization for MDPs in Figure 2.2 is shown in four parameter controls and three visualization panels. The controls give the reward, model, and policy parameters that are exposed by the MDP's software. These panels are cached in an exploration history that records the parameters and trajectories computed by the MDP. Here we explain the visual interface of MDPvis, followed by implementation details in section 2.6. Each of the following subsections describes the example in Figure 2.2.



- 1 **Discount**: The per-year discount

$ 500 **Suppression Fixed Cost**: cost per day of suppression

$ 500 **Suppression Variable Cost**: cost per hectare of suppression

Y 15 **Years to simulate**: how far to look into the future

- 25 **Futures to simulate**: how many stochastic futures to generate

% 0.9! **Harvest Percent**: timber harvest rate as a percent of annual increment

$ 50 **Minimum Timber Value**: the minimum timber value required before harvest is allowed

0 **Days Left**: for each day left in the year

0 **Temperature**: for air temperature at the time of an ignition

0 **Wind Speed**: for wind speed at the time of an ignition

0 **Timber Value**: for the timber value at an ignition location

Expected Value $ 18354.79

Expected Value $ -10385.19

Event Number: 13

Pathway Number: 4

Figure 2.2: A high level overview of the Markov Decision Process visualization prototype: MDPvis. The top row has the three parameter controls for (A) the reward specification, (B) the model modifiers, and (C) the policy definition. A fourth panel gives the history of Monte Carlo trajectory sets generated under the parameters of panels (A) through (C). Changes to the parameters enable the optimization button found under the policy definition and the Monte Carlo trajectories button found under the Exploration History section. The visualization has two buttons in the History panel for each set of Monte Carlo trajectories, one for visualizing the associated Monte Carlo trajectories and another for comparing two sets of trajectories. Below the control panels are visualization areas for (E) histograms of the initial state distribution, (F) fan charts for the distribution of variables over time, and (G) a series of individual states rendered by the simulator as images. For a readable version of the visualization we invite users to load the visualization in their browser by visiting *MDPvis.github.io*.

2.5.1 Parameter Panels

Reward Specification: $R(s, a)$

Since the goal of policy optimization is typically to find the policy maximizing the expected sum of discounted rewards, the optimization can be highly sensitive to changes in the parameters of the reward function. To explore these changes we expose the set of real valued reward parameters specified by the MDP as a list of HTML input elements.

Whenever the reward function parameters change, elements of MDPvis related to optimality and expected value are no longer valid. The user interface updates to offer buttons for optimizing a new policy and generating trajectories. Supporting these buttons for computationally expensive MDPs will be explored in chapters 3 and 4.

Model Definition: P

The MDP's simulator may expose model parameters that control the system, or eliminate complexity for testing a specific component. These can include modifiers of the transition function, the total number of transitions to simulate (otherwise known as the horizon or sampling depth), the number of potential futures to sample (otherwise known as the sampling width), modifiers to the initial state distribution, flags for deactivating parts of the model, and fidelity switches for trading execution time for higher fidelity simulations.

A second purpose of the model parameters is to expose elements of the MDP's optimization algorithm to the user. Many MDP optimization algorithms are highly sensitive to parameters for learning rate, search depth, heuristics, convergence tolerance, and optimality requirements. Selecting a reasonable set of these parameters is often an iterative process.

When these parameters change, MDPvis enables buttons for optimizing a new policy and/or generating new trajectories.

Policy Definition: $\pi(s) \mapsto a$

The policy parameters specify the current policy. Just as was the case in the prior sections, it is appropriate to present this control as a set of user-editable real numbers. When the policy is represented by parameters that have no human interpretable meaning as is the case with neural networks and sufficiently large decision trees, then this simplistic policy representation is no longer appropriate. In such cases it is still possible to explore the trajectories produced by the policy.

Policy parameters will update if the user clicks the "optimize" button.

Exploration History

As the user repeatedly generates sets of trajectories under different parameter settings, they may want to return to a prior parameter set to continue exploration or to compare sets of trajectories generated under prior parameter sets. Here we add two buttons for every set of trajectories that have been generated. One button will reload the prior set of parameters and the trajectories that they generated into the visualization. The other button will put the visualization into "comparison mode," which displays the difference between two trajectory sets in the visualization areas.

When in comparison mode, the reward, model, and policy parameters cannot be edited, since they display the differences in the parameter values between the current set of trajectories and the one being applied as a comparator. More information about the comparison mode is included in the following visualization areas.

2.5.2 Visualization Panels

Having specified all the parameters, we can request trajectory sets from the MDP simulator and display them in the visualization areas. The first two visualization areas show sets of trajectories and support applying filters to the current trajectory set. When we apply filters to the trajectory set in one visualization area, the displayed trajectory sets update throughout the visualization, i.e. the visualization is cross-linked.

State Variable Distribution at Event: $P_{n|\pi}$

We show the distribution of states at a particular timestep as a histogram. The user can select a range of values in the histogram to filter trajectories. This supports a global-to-local [52] testing process where exploration starts with an overview of all trajectories before drilling down into specific trajectories. When the user drills down to specific trajectories, the count of filtered trajectories is shown as the unfilled portion of the histogram bar.

When the visualization is in comparison mode, the histograms transform into a a bar chart showing the difference in counts for the bins between the two sets of trajectories.

State Variable Evolution: P

An important question when testing an MDP is "Do the state distributions reflect the real-world?" (See Table A.1). In MDPs, there can be many variables that evolve over time to produce a distribution of outcomes at each time step. In this area, we represent distributions as fan charts giving the deciles of each variable across trajectory timesteps.

To produce the fan chart, we first plot a lightly colored area whose top and bottom represent the largest and smallest value of the variable in each time step. On top of this lightest color, we plot a series of colors increasing in darkness with nearness to the trajectory set's median value for the timestep.

By giving the percentiles, we are able to show both the diversity of outcomes and the probability of particular ranges of values. If the number of trajectories is small enough to avoid the visual clutter of many intersecting lines, we render the trajectories as a line chart.

The histograms and fan charts both support the visualization of conditional distributions, where the user specifies a set of filters that subset the trajectories shown in the fan charts and histograms. The filters between the fan charts and histograms are cross-linked. When filtering the fan chart, the filters in the corresponding histogram updates, and vice versa. Changing the timestep of the fan chart's filter updates the histograms to the newly-selected timestep.

When in comparison mode, the color extents are plotted by subtracting a color's maximum (minimum) extent from the corresponding maximum (minimum) extent of the other fan chart. Figures 2.6, 2.7, and 2.15 show fan charts rendered in comparison mode.

Once the user filters out enough trajectories, the fan charts switch to line charts. Clicking on one of the lines in the line charts requests the state detail.

State Detail:$S_{i,j}$

We considered but ultimately rejected rendering the details of individual trajectories with a scatterplot matrix, parallel coordinates, Multi-Dimensional Scaling (MDS), or Multiple Correspondence Analysis (MCA). We found all these approaches would unnecessarily tie the MDP practitioner to an inadequate representation. Instead, we allow the MDP simulator to render a two dimensional array of images or videos that will be shown at the bottom of MDPvis.

For example, in the wildfire domain the state of the world is captured by four images of the landscape's timber and fuel values. Our co-authors in forestry were already rendering these landscapes, as shown in Figure 2.2, so we integrated MDPvis with these standard visualizations. Non-spatial MDPs would not render landscapes, but they could return a visualization relevant to their practitioners.

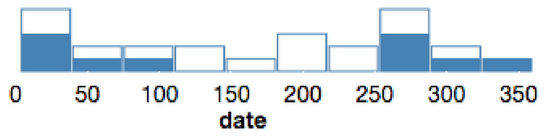
2.5.3 Parameter Space Analysis (PSA) Examples

Here we demonstrate the functionality of MDPvis for the PSA categories given by Sedlmair et al. [52] with fictitious examples from a wildfire domain. The intent of these examples is to illustrate how visualization with MDPvis is analogous to unit testing with visual expectations. In some cases, these expectations could be written in closed form as traditional unit tests, but traditional unit tests presume the developer has the resources, expertise, and foreknowledge to codify the expectation. We expand on the examples of this section with real-world bugs in Section 2.7.

Testing ***Sensitivity*** *with Interaction*: Testing Question 26 in Table A.6 asks "What are the most important drivers of policy?" Policies are typically functions mapping states to actions. This is an instance of a function visualization problem with a domain of $|S|$ inputs and a codomain of $|A|$ outputs. However, a policy's tendency to inhabit a particular region of the state space means we do not need to examine how the policy function maps *all* states to actions. We can focus on the states produced by Monte Carlo trajectories under the policy function. Testing a policy's sensitivity to the state variables involves filtering (excluding) actions to see which state variables in the trajectory set mapped to the unfiltered actions. Figure 2.3 shows a visual expectation next to a buggy result.

Testing ***Optimization*** *by Optimizing*: Question 17 in Table A.4 asks, "Is the policy converged to an acceptable local optimum?" The expected reward of MDPs are typically non-convex functions, meaning the optimization can terminate with policies that are clearly suboptimal when viewing how the states map to actions. For bad policies we want to know if the optimizer received an unlucky set of Monte Carlo trajectories, started from a bad initial policy, or broke due to a buggy implementation. To detect these three cases, we can change the parameters of the MDP to predict how the optimizer should improve the policy and compare to the prediction.

For example, let's use optimization to test the optimizer of a wildfire MDP. Specifically, let's start with a landscape covered by a species whose timber is not



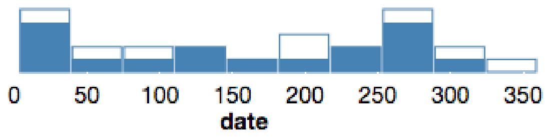


Figure 2.3: We generated two histograms in MDPvis showing the dates a fire started on a landscape, then filtered the trajectories based on the action taken in this time step. In both these charts the filled portion represents fires that were allowed to burn unsuppressed, and the unfilled portion represents fires that were suppressed. Since ignition date is related to weather, we expect (top) to allow more fires to burn at the start and end of the season, but not in the middle of the season when the landscape is much drier. A buggy result (bottom) shows no apparent relationship.

valuable at harvest time. We would expect suppression expenses to dominate timber harvests and produce a policy that never suppresses fires as shown at the top of Figure 2.4. If we change the parameters of the MDP so that harvests are worth billions of dollars, we would expect a newly optimized policy to begin suppressing fires (the bottom of Figure 2.4).

If the optimized policy does not change in response to these large reward function changes, then there is likely a bug.

Testing ***Outliers*** *with State Detail*: Question 8 in Table A.2 asks, “Are the most extreme state transitions realistic?” With optimization algorithms, it is important to validate the most extreme trajectories, since they have disproportionate effect.

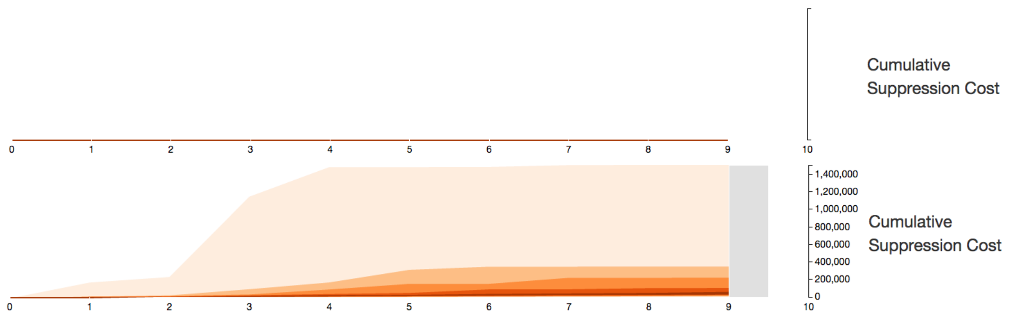


Figure 2.4: When we make timber harvest more valuable, we would expect an optimized policy to suppress more fires. Here we optimize a policy after changing the reward function to increase timber value. We expect the suppression expenses of the top chart to update to the bottom chart after optimization.

In the wildfire domain, we may ask whether the most extreme wildfires respect firebreaks established by previous fires. We can generate state details for the largest fires by filtering the time step with the largest fires in the fan chart. Once fewer trajectories are displayed, we can generate the state snapshots whose expectations and buggy results are shown in 2.5.

Testing ***Partition*** *by Comparing*: Question 14 in Table A.3 asks, "Do policies differ in their resultant state distributions as expected?" Domain experts may have mental models of MDP behavior subject to various policies. In MDPVIS the domain expert can compare state partitions produced by different policies. Figure 2.6 shows a comparison's expectation and a buggy result.

Testing ***Uncertainty*** *by Re-Parameterizing*: Question 20 of Table A.5, "How certain is the policy function about a specific choice?" We don't want the policy to change if we change the simulator's parameters within the range consistent with real-world data.

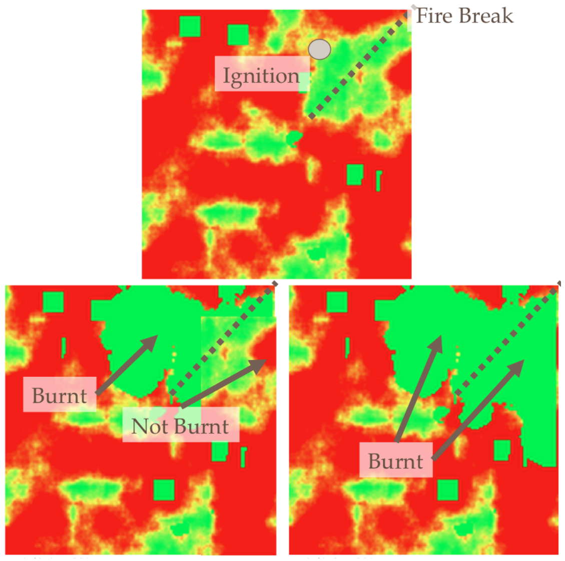


Figure 2.5: The top row contains a single starting state and the bottom row contains an expected result state (left) next to a buggy result (right). The dotted lines show a "fire break" formed by a previous fire that should prevent future fires from spreading directly across the landscape. By filtering to this outlier we can check whether the largest wildfires are respecting fire breaks.

For example, fire management practices are more effective today than in the year 1900, but what if fire fighting practices continue to improve (a form of structural uncertainty)? Fire managers will want to know if the policy optimized for a future with better firefighting technology will be different from the policy optimized for a continued status quo. MDPvis supports exploration of structural uncertainty by permitting the user to change parameters, such as "suppression effect," and re-optimize. Figure 2.7 shows two potential results of this analysis.

Testing ***Fit*** *by Augmenting*: Question 4 of Table A.1 asks, "Do simulated transitions result in realistic state distributions?" Many domains will have datasets

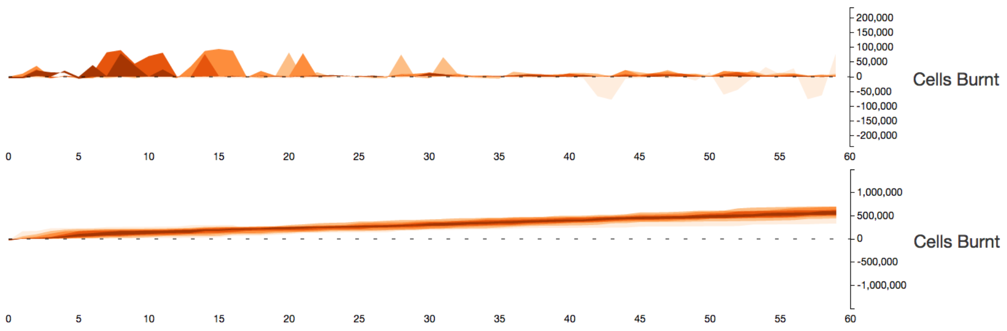


Figure 2.6: These charts show two comparisons of the number of cells burnt for policies that allow all fires to burn or suppress all fires. When the color in the chart is above the center line, the let-burn-all policy has more burnt pixels in that time step. We would expect (top) the number of cells burnt to be greater initially in the let-burn-all policy, but for the number of cells burnt to decline as fires reduce fuel levels. A buggy result (bottom) shows a steady increase in cells burnt relative to the suppress-all policy.

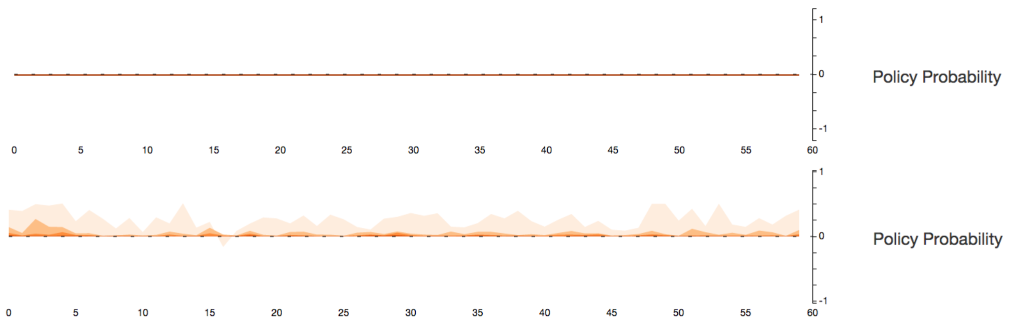


Figure 2.7: Since we are uncertain how fire suppression effectiveness will change through time, we want policies optimized under different effectiveness levels to be similar. We can explore policy similarity by comparing policies optimized under two different effectiveness parameters. The top chart is a comparison fan chart showing a single line straight across at zero. This means there is no difference in the policy probabilities between the policies optimized under different suppression effectiveness parameters. The second chart gives a buggy result, showing many differences in the policy probability between the two parameterizations.

for historical policies. These data give distributions of state variables that can be compared against the simulated distributions. We can add historical distributions by augmenting the dataset with a derived variable for each state variable having historical data. This technique is best demonstrated with an example.

In the wildfire domain, we take a variable for which we have many real-world samples, such as "Growth Percentile." Based on historical data, we can assign a "Historical Growth Percentile" as the percentile value of each Growth Percentile value within the historical dataset. Figure 2.8 shows the visual expectation in the fan chart, which has consistent percentiles across all time steps. A buggy result given in Figure 2.8 shows the percentiles of the simulated dataset departing from the historical distribution.

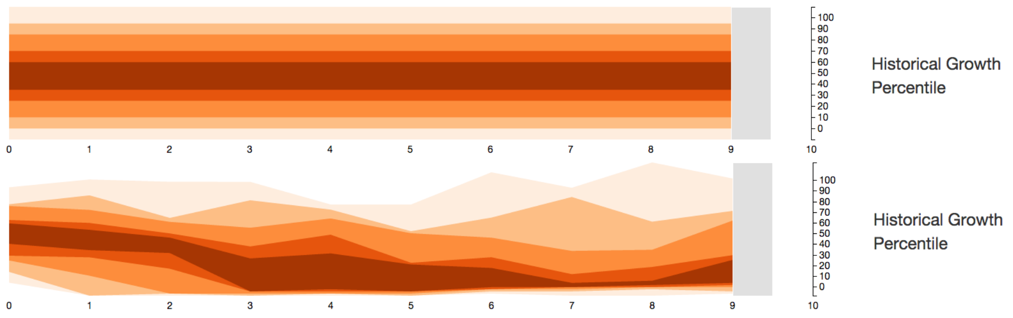


Figure 2.8: These charts check how well the distribution of simulated vegetative growth matches the distribution of historical vegetative growth. We produce the data for these charts by adding a variables to each state that maps the vegetative growth to the percentile the growth represents in the historical dataset. Thus we expect the median value (the dark red) to surround the 50th growth percentile, and the lighter colors to match up with the percentiles as is appropriate. The top chart shows the simulated distribution exactly matches the historical distribution, and the bottom chart is comparatively very buggy.

2.6 MDPvis Implementation

We built MDPvis as a data-driven web application. Building on the web application stack affords two primary benefits. First, Brehmer and Munzner [10] identify sharing as an important feature to implement, and the ubiquity of web browsers makes it an ideal platform for collaboration. Second, the web stack emphasizes standard data interchange formats that ease integration with MDP simulators and optimization algorithms. We identified four HTTP requests *(initialize, trajectories, optimize,* and *state)* that are answered by the MDP simulator. These requests do not assume a particular domain or implementation language. In most cases, the requests should be able to interface with the MDP simulator and optimizer using the same command-line client they would typically implement for testing a domain.

MDPvis issues the following HTTP requests to the MDP simulator and optimizer:

1. ***/initialize*** – Ask for the parameters that should be displayed to the user. The parameters are a list of tuples, each containing the name, description, minimum value, maximum value, and current value of a parameter. These parameters are then rendered as HTML input elements for the user to modify. Following initialization, MDPvis automatically requests trajectories for the initial parameters.

2. ***/trajectories?QUERY*** – Get trajectories for the parameters that are currently defined in the user interface. The server returns an array of arrays

containing the state variables that should be rendered for each time step.

3. ***/optimize?QUERY*** – Get a newly optimized policy. This sends all the parameters defined in the user interface for the MDP and the MDP's optimization algorithm returns a newly optimized policy as a new set of policy parameters.

4. ***/state?IDENTIFIER*** – Get the full state details and images associated with a state selected by the user.

All relevant languages have web server libraries that can be integrated with the MDP's code base for serializing Monte Carlo trajectories. We integrated the wildfire domain with the visualization by adding a serialization library (one line of code) and modifying a dummy data file to initialize the domain. The most challenging integration task was parsing the HTTP parameters into the expected data types for the simulation code and writing the loop to invoke the simulator multiple times.

2.6.1 Integration with Reinforcement Learning Frameworks

Machine learning researchers typically evaluate new algorithms on interesting challenge domains or toy domains that illustrate some capability or failure of an algorithm. The problem domain collections of Geramifard et al. [25], Bellemare et al. [6], Duan et al. [18], and Brockman et al. [12] aim to unify the reinforcement learning research community behind shared implementations with consistent APIs.

We integrated MDPvis with two reinforcement learning frameworks to test our claims of implementation generality.

RLPy (Reinforcement Learning Python) is a collection of 26 problem domains. We integrated three of RLPy's 26 problem domains, including the toy domains of Mountain Car and grid world and the more challenging domain of HIV Treatment introduced by Ernst et al. [19]. The web server and data formatting code are available online [35] and can be adapted to the 23 additional domains in RLPy. We now discuss our experience with three of the domains.

Mountain Car models an underpowered car trapped between two hills. Successful policies in Mountain Car store energy by rocking the car back and forth between the hills until it has enough momentum to escape. We created a policy function for Mountain Car with a parameter controlling the probability of reinforcing the current direction of travel by accelerating in that direction. We exposed this parameter to MDPvis to explore different policy outcomes. The domain has a parameter for "noise," which gives the probability of replacing the action chosen by the policy with a randomly chosen action.

After integrating RLPy's Mountain Car implementation with MDPvis, we discovered the capitalization of the noise parameter in the constructor differed from the default noise level. The result was a failure to update the domain to the selected noise value. We discovered this bug when testing RLPy's integration with MDPvis. Figure 2.9 shows two fan charts for the car's x position. The bug would not be apparent to Mountain Car's developers, because their visualization tools do not show the absence of variance. We subsequently submitted a bug fix to the

RLPy maintainers, who merged the fix a day later. For more details, see the pull request on GitHub [33].

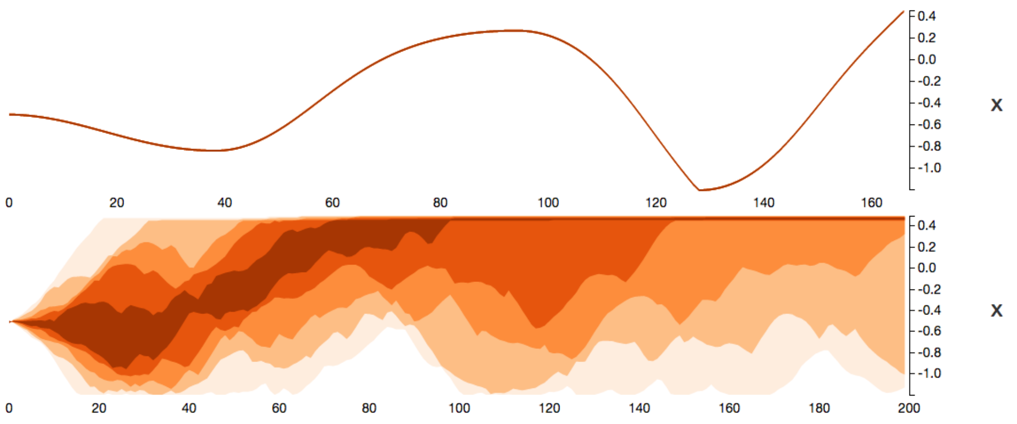


Figure 2.9: The two charts show the Mountain Car domain before (top) and after (bottom) fixing the noise parameter, which is set to 0.2 in both cases. The top chart shows no variation across trajectories despite the expectation that the actions would stochastically change in proportion to the noise parameter.

Grid World models navigation tasks in two dimensions. Researchers often use it to explore an algorithm's ability to avoid obstacles (pits) and learn increasingly long series of actions in order to reach a terminal reward. Unlike Mountain Car and our wildfire domain, which have only two actions, the Grid World domains have an action for each of the four cardinal directions. To accommodate filtering trajectories based on action selection, we defined an indicator variable for each of the four actions which is equal to 1 when that action is chosen by the policy and 0 otherwise. Figure 2.10 shows the resulting fan charts for the actions as drawn by a random policy.

Grid World domains also show the importance of allowing the simulator to

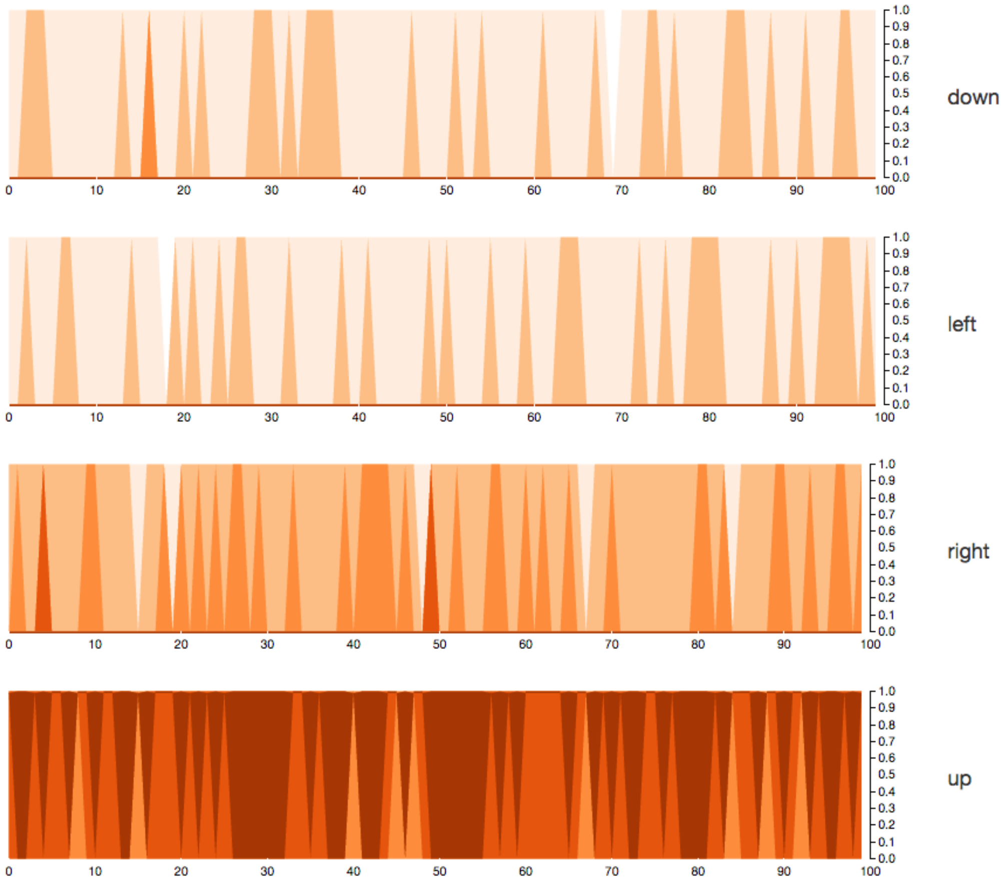


Figure 2.10: Four fan charts represent each of the actions taken for a Grid World domain. In this grid world map the agent starts in the lower left corner with a pit immediately to its right. Successful policies will move up, right, then down to reach the goal grid cell. We created a stochastic policy that more frequently moves "up," followed by less frequent moves in other directions. The fan charts show our simplistic policy does not change action selection probability through time. If we want to know what actions lead to or from a particular state, we can apply filters to state variables and view the action fan charts.

return images and videos for the state detail view. Grid Worlds have a natural 2-dimensional representation (a grid) that is easier to interpret than the fan chart representation.

HIV Treatment is a medical treatment domain with six immune system state variables that are measured every 5 days over 1000 days:

- T_1: the number of healthy $CD4^+$ T-lymphocytes
- T_2: the number of healthy macrophages
- T_1^*: the number of infected $CD4^+$ T-lymphocytes
- T_2^*: the number of infected macrophages
- V: the number of free virus particles
- E: the number of HIV-specific cytotoxic T-cells

HIV Treatment policies dynamically select one of four drug therapy options, including reverse transcriptase inhibitor (RTI), protease inhibitor (PI), a combination of these drugs, or no treatment. Each drug affects a different step of the virus's replication process. Successful policies produce a drug schedule that jointly minimizes virus count and medication side effects. A typical real world policy includes "Structured Treatment Interruption" (STI), which cycles patients between treatment and non-treatment periods.

We created a parameterized policy class that mimicked our interpretation of STI. The parameters are independent probabilities of administering RTI and PI. We automatically administer both HIV drugs if the patient's infected cells or free viruses are "spiking."

Figure 2.11 shows fan charts under three different policies. The top chart is for a policy with high probabilities of administering both drugs, and the bottom chart only administers drugs under the “spiking” condition. Since the domain is implemented as a deterministic integration of differential equations, we can see the outcomes converge to a single patient history as the policy becomes more deterministic. The scale of the spikes on the top chart is also greater than the spikes on the charts administering less medication. This suggests that the medication carries risks requiring further study.

The second set of domains we integrated are implemented in the **OpenAI “Gym”** framework of Brockman et al. [12]. OpenAI Gym is both a collection of problem domains for researchers and a leaderboard website showing the relative performance of optimization algorithms developed by various researchers. It includes eight “environments” that range from the simplistic classical control problems to the Atari 2600 games from Bellemare et al. [6]. The environments provide a total of 176 domains.

OpenAI Gym includes custom trajectory animations for domains. Developers can upload videos of these animations to OpenAI’s servers for display with their leaderboard entries. We integrated these videos with MDPvis through the state detail panel.

OpenAI Gym problem domains can be classified as either classic control problems, or raw perception problems. Optimization in classic control problems use relatively short state vectors where each entry in the vector is a “feature.” These vectors typically have fewer dimensions than the complete state space, as is the

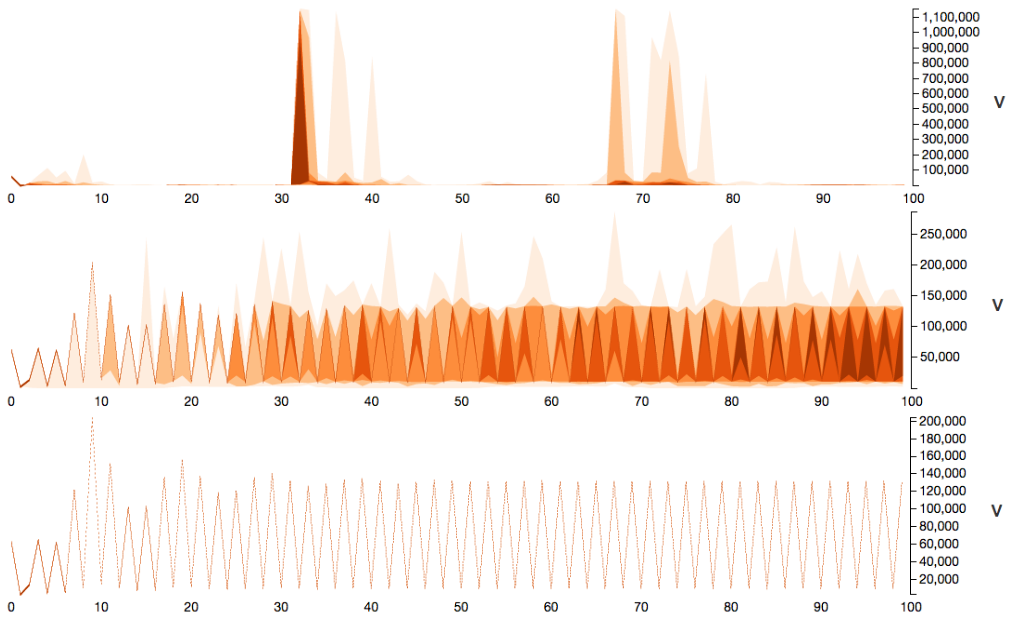


Figure 2.11: The virus counts (V) for patients under three policies. The top fan chart was generated by a policy with a probability of administering RTI and PI of 0.4. The middle chart has the probabilities of 0.01 and the bottom chart only administers the drug when the virus count is spiking. You can see the top chart largely keeps the virus to a minimum. The middle chart is dominated by the automatic administration of RTI and PI when the patient is spiking, but the stochastic administration of the drugs in other time steps produces some variation. The last chart shows the determinism of the HIV domain. All the patients have the same patient histories when the policy and transition dynamics are both deterministic.

case in our wildfire example where we employ summary statistics of a landscape rather than the full landscape. In raw perception problems, the optimization algorithm is given the complete state information in the form of an image, similar to the way a human would perceive the domain. Solving these problems typically requires the application of deep learning methods.

Optimization in raw perception domains blends selecting actions and learning what combinations of pixel values are important. Machine learning research calls this process "feature learning." The advantage of feature learning is that it eliminates the need for the programmer to define features. A drawback is that the resulting features are often difficult to understand and do not correspond to the features a human programmer would create. Developing techniques for understanding complex learned features is an active area of research (e.g. Zahavy et al. [65]).

We integrated all four of OpenAI Gym's classical control domains, which include Pendulum, Acrobot, Cartpole, and Mountain Car. We also integrated all 55 Atari games bundled into OpenAI Gym via the Arcade Learning Environment. The web server and data presentation code developed for both these domain collections share a single lightweight web server found at GitHub [34]. The server integrates with each domain by specifying the name of the domain and the names of the visualized state variables when starting the server. This makes it possible to switch between the domains without touching the web server's implementation. Next we highlight two of the domains we integrated.

Pendulum is a domain concerned with balancing a pendulum by applying torque either left or right so as to maintain the position of the pendulum. Unlike all domains presented to this point, Inverted Pendulum has a continuously valued action space bounded by the maximum and minimum torque values. Continuous action spaces are typically more challenging to optimize than discrete action spaces, but they have an intuitive mapping within MDPvis. The histograms and fan charts both vary continuously and can thus render continuous actions.

OpenAI Gym includes a visualization for pendulum that shows the position of the pendulum and the force applied. Figure 2.12 shows the video for a selected trajectory in the State Detail area of MDPvis.

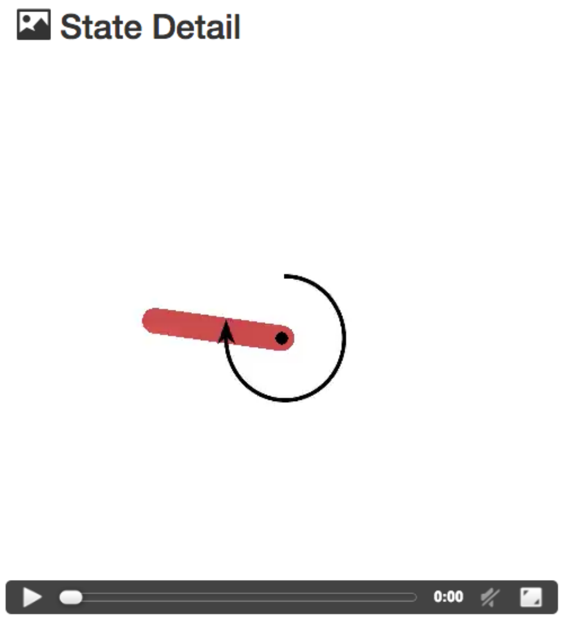


Figure 2.12: MDPvis displays a media player for the video generated by the Pendulum domain included in OpenAI Gym. This video is displayed by clicking the trajectory associated with it in the Fan Charts.

Ms. Pac-Man is a perception domain based on the popular 1982 title of the same name. The goal is to produce a series of actions that avoid ghosts while consuming all the dots on the screen. To support perception domains like Ms. Pac-Man, we extended MDPvis with the ability to label axes with an image. Figure 2.13 shows a time series defined by an image saved from the initial game state. The image defines a similarity score for each game frame. The similarity score displays in the fan chart next to the image.

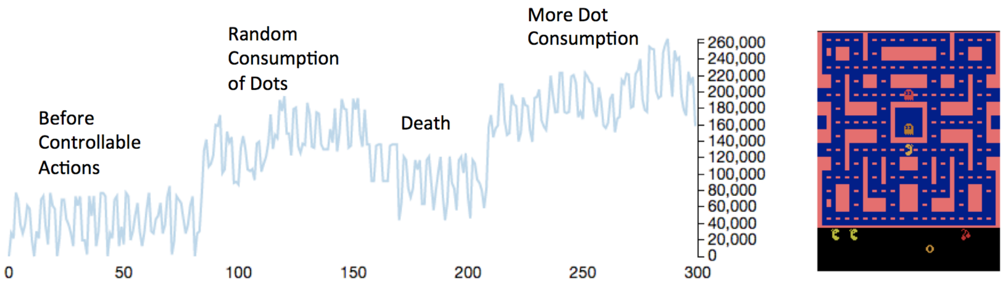


Figure 2.13: An annotated time series displayed within MDPvis for the Ms. Pac-Man Atari domain. The graphic on the right side is the image used to generate similarity scores by summing the difference in each pixel between the displayed image and the image generated within the trajectory. Many Atari games render different game state every other frame, which accounts for the jitter in the time series. The text annotations (from left to right) show the initial phase of the game, followed by a phase in which the ghosts are hunting the dot-consuming player, before a ghost catches the character and the character and ghost positions reset to the position found in the image on the right.

Our similarity metric comparing pixels at consistent screen coordinates is not tied to the visual features that the optimization algorithm determines are important. The work of Zahavy et al. [65] show we can use components of neural networks for both selecting the images and determining the similarity among images based on what the network is paying attention to. We leave this effort to future work.

2.6.2 Debugging Capabilities

Bugs may result from bad implementation (code) or from bad parameters. In the case of parameter values, it is possible to fix (debug) the problem without leaving MDPvis. However, non-programmer domain experts cannot fix bad implementation without sharing a bug report with the software developer. More robust integration of MDPvis with the software developer's Integrated Development Environment (IDE) and version control would expand MDPvis's debugging capabilities. In particular, MDPvis is not currently aware of versions of the MDP simulator or optimizer. As code changes, it would be useful to treat each build as a selectable parameter in the interface. This would allow for more robust collaboration between team members as implementations change. Further, selecting versions would allow for visual regression testing.

2.7 Use-Case Study: Wildfire Suppression Policies

We arrived at our generalized visualization by following Munzner's nested model [44] for the problem domain of Wildfire Suppression and then generalized the results to all MDPs. Since the Wildfire Domain has high-dimensional states, it is representative of a particularly challenging class of MDPs.

We used MDPvis in a use-case study to provide anecdotal evidence of the utility of MDPvis. This case study is based on user sessions with our forestry economics collaborators who formulated fire suppression policies as an MDP optimization problem. Throughout the design and development process, we worked closely with these domain experts, to identify their needs as developers of MDP solutions. The analyses in this section were performed by these experts during their first use of MDPvis, in conjunction with the author (a PhD student from computer science who implemented MDPvis but did not contribute to the development of the MDP).

The wildfire suppression domain tested in this section combines models for fire spread, vegetative growth, weather, suppression effectiveness, suppression cost, and harvest. The domain is an idealized version of the wildfire suppression domain explored in Chapters 3 and 4, meaning it is both simpler and faster to evaluate than the full-fidelity version it is based on. We evaluate on the idealized simulator because it generates trajectory sets within a few seconds of user time, whereas the full-fidelity version takes 7 hours to generate a single 100-year trajectory.

We expressed the suppression policy as a differentiable function with interpretable coefficients. Each coefficient resembles a weight that would be generated by a logistic regression. Increasing a coefficient's value makes it more likely that a wildfire will be suppressed for increasing values of the corresponding state variable. We applied policy gradient methods [55, 5, 46] to optimize new policies, which have the advantage of being both fast and likely to improve the policy from an uninformed policy for all sample sizes.

We detected bugs for most of the questions of A.1 and highlight interesting cases with their interactions under their corresponding analysis tasks below.

Fitting: Several failures to simulate real conditions were detected. Upon filtering the trajectories to the ones containing the most extreme fires, we examined the state details and found that the fires were not spreading east or south from the ignition site (see Figure 2.14). We could only see this on the larger fires, because the rectangular timber harvests were masking the unusual shape of the fire spread.

Outliers: A common real-world wildfire suppression policy suppresses the vast

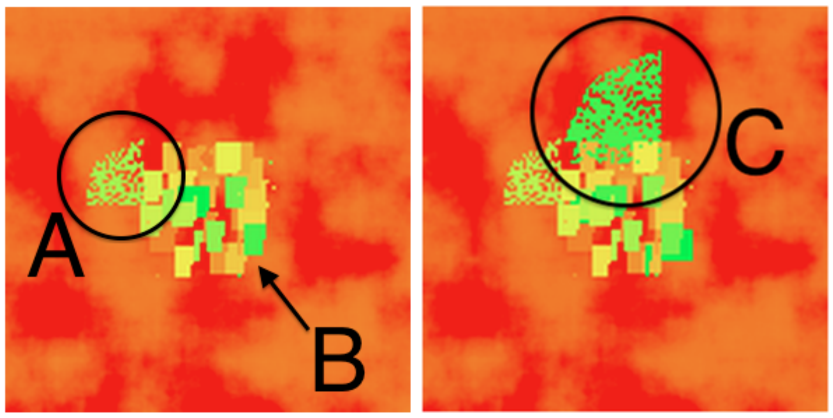


Figure 2.14: Two sequential spatial snapshots of timber values for a state transition that includes one of the largest fire loss events from 200 trajectories of 60 years. The management area is visible in the center due to the rectangular timber harvests. These rectangular harvests obscure the irregular boundary of small fires. (A) shows a medium size fire obscured by a mixture of small fires and timber harvests in (B). The straight edge of the largest fire introduced in (C) clearly shows the fire is not spreading in all directions.

majority of wildfires, so it forms a natural baseline for comparisons. To better illustrate the outcomes of a suppress-all policy, we compared it to a let-burn-all policy and found a surprising fact: the let-burn-all policy has higher expected reward than the suppress-all policy. This shows that either the models do not balance the various rewards of fire suppression properly or a policy that is completely opposite from current forestry practices produces better outcomes. We found the reason for this counterintuitive result after filtering trajectories to only display outlier fires. Large fires immediately increase the harvest reward, because the maximum allowable harvest is a function of tree growth. The harvest level is depressed without large fires creating the conditions for explosive post-fire growth.

Partition: When comparing two different policies (see Figure 2.15) under otherwise consistent parameters, we observed a slight difference in the percentiles of the weather events. Since these weather events are exogenously determined by a random number generator with a consistent seed, this difference indicates that the random number generator is called differently depending on the action that is selected. Without fixing this bug, we cannot compare a landscape's response to different policies under the same set of ignition events. Further, this effectively allows optimization algorithms to choose the weather by running experiments under different policies until they experience the best possible weather conditions.

Optimization: Although the policies reported by our policy gradient algorithm improved upon our naive baselines, we found it easy to improve upon the machine optimized policies by making small changes to the policy parameters. This shows that the optimizer is failing to find a local optimum.

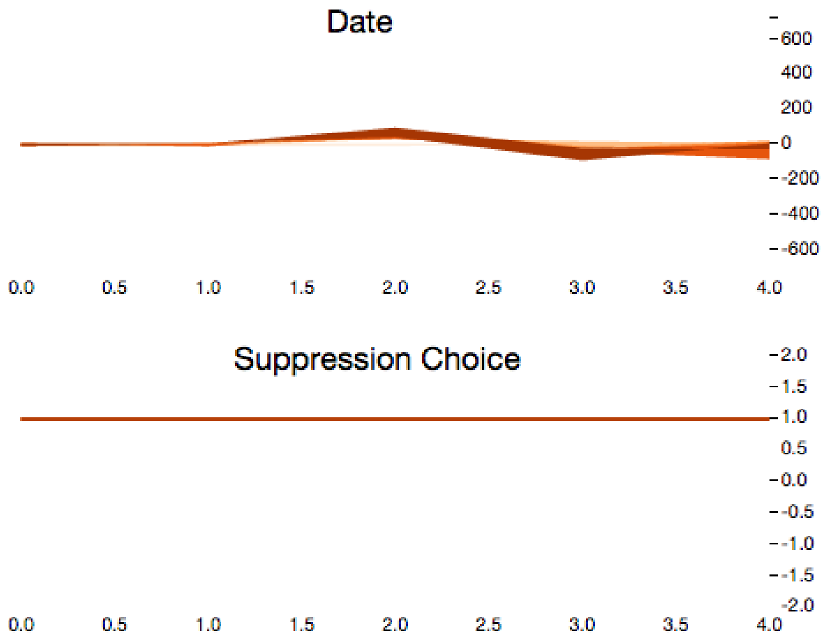


Figure 2.15: Fan charts for the suppression choice and the ignition date shown in comparison mode for two trajectory sets. We generated one trajectory set under a suppress-all policy and a second trajectory set under a let-burn-all policy. We confirmed that the proper action is being selected for each state by observing the differences in suppression choices is always 1. However, there is an unexpected difference in the dates, which should be consistent between the two trajectory sets.

We were able to identify the most likely source of the problem: when we optimized policies for increasing trajectory depths, we found a runtime bug with our implementation of importance sampling that produces a division by zero. We hypothesize that this runtime fault is causing our optimization function's other anomalous results.

Sensitivity: When viewing the timber harvest chart in comparison mode, the lack of substantial differences in harvest volumes for different policies indicates that the harvest volume is not sensitive to the policy. Additional comparisons after fixing bugs showed harvests are only significantly impacted when old growth forest dominates the landscape and trees stop growing. Since the model rarely reaches 100 percent old growth, the harvest level is not sensitive to the policy.

Uncertainty: The invariant harvest reward means it is always better to let a wildfire burn. After using MDPvis with other US federal land wildfire simulators, we found this lack of uncertainty to be faithful to real world tradeoffs. This poses an epistemological problem to our team of wildfire suppression optimization researchers. Finding interesting research questions requires changing to a privately managed landscape or including a case analysis of policies optimized for different harvest levels.

2.8 Conclusion and Availability

This chapter presented MDPvis, a domain-independent tool supporting the testing and debugging of MDP simulation and optimization software. MDPvis employs a simple web service protocol to interact with the MDP simulator and optimizer and supports many visual analysis tasks related to MDP testing. MDPvis allows for viewing trajectory distributions over time and making temporal comparisons between policies (either policies produced by the optimizer or policies designed by the user). When we integrated MDPvis with a simple reinforcement learning domain (Mountain Car), we unexpectedly found a bug, which we reported to the framework authors. We presented a use-case study in which our users immediately discovered several serious bugs. We also discovered interesting behavior that is either a bug or an indication that real-world policies diverge significantly from the optimal policy. Our users report that MDPvis is already greatly accelerating their testing and debugging processes, and they are looking forward to applying it to other MDP simulators.

A live version of MDPvis, the source code, and integration instructions are available at *MDPvis.github.io*.

The wildfire simulator tested in the use-case study of this chapter is a simplified reimplementation of the computationally expensive wildfire simulator explored in the next two chapters. While the simplified simulator is ideal for prototyping and evaluating MDPvis, it is not sufficiently faithful to the real world to make policy recommendations. In the next chapter we show how to create a fast surrogate model based on data from the expensive simulator. The surrogate model allows users to interactively test, explain, and configure MDP simulators.

3 Fast Visualization with Model-Free Monte Carlo

3.1 Introduction

Visual analytics is particularly useful for problem domains with high-dimensional state spaces, but these high-dimensional problems are also the ones that tend to be computationally expensive to simulate. The transition dynamics of our fire management MDP are defined by a simulator that can take up to 7 hours to simulate a single 100-year trajectory of fire ignitions and resulting landscapes. How can we support interactive policy analysis when the simulator is so expensive?

Our approach is to develop a surrogate model that can substitute for the simulator. We start by designing a small set of "seed policies" and invoking the slow simulator to generate several 100-year trajectories for each policy. This gives us a database of state transitions of the form (s_t, a_t, r_t, s_{t+1}), where s_t is the state at time t, a_t is the selected action, r_t is the resulting reward, and s_{t+1} is the resulting state. Given a new policy π to visualize, we apply the method of Model-Free Monte Carlo (MFMC) developed by Fonteneau et al. [23] to simulate trajectories for π by stitching together state transitions according to a specified distance metric Δ. Given a current state s and desired action $a = \pi(s)$, MFMC searches the database to find a tuple $(\tilde{s}, \tilde{a}, r, s')$ that minimizes the distance $\Delta((s, a), (\tilde{s}, \tilde{a}))$. It then uses s' as the resulting state and r as the corresponding one-step reward.

We call this operation "stitching" (s, a) to $(\tilde{s}, \tilde{a})$. MFMC is guaranteed to give reasonable simulated trajectories under assumptions about the smoothness of the transition dynamics and reward function and provided that each matched tuple is removed from the database when it is used. Algorithm 1 provides the pseudocode for MFMC generating a single trajectory.

Fonteneau et al. [22] apply MFMC to estimate the expected cumulative return of a new policy π by calling MFMC n times and computing the average cumulative reward of the resulting trajectories. We will refer to this as the MFMC estimate $V^{\pi}_{MFMC}(s_0)$ of $V^{\pi}(s_0)$.

Algorithm 1: MFMC for a single trajectory. When generating multiple trajectories for a single trajectory set, the state transitions from D must not be reused [22].

```
Input Parameters: Policy $\pi$, horizon $h$, starting state $s_0$, distance metric
  $\Delta(.,.)$, database $D$;
Returns: $(s, a, r)_1, ..., (s', a', r')_h$;
$t \leftarrow \emptyset$;
$s \leftarrow s_0$;
while length(t)$< h$ do
    $a \leftarrow \pi(s)$;
    $\mathcal{H} \leftarrow \underset{(\widetilde{s}, \widetilde{a}, r, s') \in D}{\text{argmin}} \Delta((s, a), (\widetilde{s}, \widetilde{a}))$;
    $r \leftarrow \mathcal{H}^r$;
    append$(t, (s, a, r))$;
    $s \leftarrow \mathcal{H}^{s'}$;
    $D \leftarrow D \setminus \mathcal{H}$;
end
return$(t)$;
```

In high-dimensional spaces (i.e., where the states and actions are described by many features), MFMC breaks because of two related problems. First, distances become less informative in high-dimensional spaces. Second, the required number of seed-policy trajectories grows exponentially in the dimensionality of the space. The main technical contribution of this chapter is to introduce a modified algorithm, MFMCi, that reduces the dimensionality of the distance matching process by factoring out certain exogenous state variables and removing the features describing the action. In many applications, this can very substantially reduce the dimensionality of the matching process to the point that MFMC is again practical.

This chapter is organized as follows. First, we briefly review previous research in surrogate modeling. Second, we introduce our method for factoring out exogenous variables. The method requires a modification to the way that trajectories are generated from the seed policies. With this modification, we prove that MFMCi gives sound results and that it has lower bias and variance than MFMC. Third, we conduct an experimental evaluation of MFMCi on our fire management problem. We show that MFMCi gives good performance for three different classes of policies and that for a fixed database size, it gives much more accurate visualizations than MFMC.

3.2 Related Work

Surrogate modeling is the construction of a fast simulator that can substitute for a slow simulator. When designing a surrogate model for our wildfire suppression problem, we can consider several possible approaches.

First, we could write our own simulator for fire spread, timber harvest, weather, and vegetative growth that computes the state transitions more efficiently. For instance, Arca et al. [4] use a custom-built model running on GPUs to calculate fire risk maps and mitigation strategies. However, developing a new simulator requires additional work to design, implement, and (especially) validate the simulator. This cost can easily overwhelm the resulting time savings.

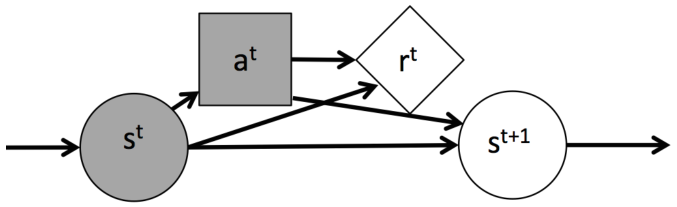
A second approach would be to learn a parametric surrogate model from data generated by the slow simulator. For instance, Abbeel et al. [1] learn helicopter dynamics by updating the parameters of a function designed specifically for helicopter flight. Designing a suitable parametric model that can capture weather, vegetation, fire spread, and the effect of fire suppression would require a major modeling effort.

Instead of pursuing these two approaches, we adopted the method of Model-Free Monte Carlo (MFMC). In MFMC, the model is replaced by a database of transitions computed from the slow simulator. MFMC is "model-free" in the sense that it does not learn an explicit model of the transition probabilities. In effect, the database constitutes the transition model (c.f., Dyna; [56]).

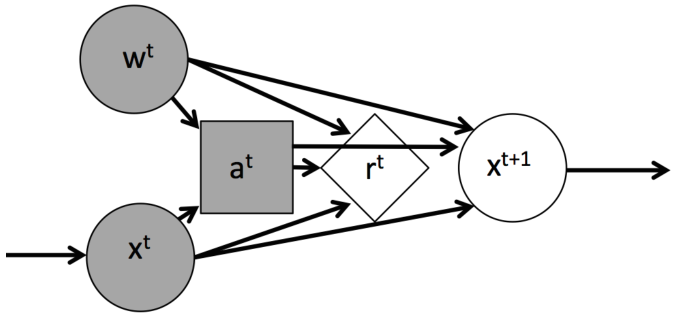
3.3 Notation

We work with the standard finite horizon undiscounted MDP, which is consistent with the notation given in the previous chapter after assigning the discount factor (γ) to 1.

In this chapter, we focus on two queries about a given MDP. First, given a



(a) The standard MDP transition.



(b) MDP transition with *exogenous* (w) and *Markovian* variables (x).

Figure 3.1: MDP probabilistic graphical models.

policy π, we wish to estimate the expected cumulative reward of executing that policy starting in state s_0: $V^{\pi}(s_0) = \mathbb{E}[\sum_{t=0}^{h} R(s_t, \pi(s_t))|s_0, \pi]$. Second, we are interested in visualizing the distribution of the states visited at time t: $P(s_t|s_0, \pi)$. In particular, let $v^1, \ldots, v^m$ be functions that compute interesting properties of a state. For example, in our fire domain, $v^1(s)$ might compute the total area of old growth Douglas fir and $v^2(s)$ might compute the total volume of harvestable wood. Visualizing the distribution of these properties over time gives policy makers insight into how the system will evolve when it is controlled by policy π.

3.4 Factoring State to Improve MFMC

We now describe how we can factor the state variables of an MDP in order to reduce the dimensionality of the MFMC stitching computation. State variables can be divided into Markovian and Time-Independent random variables. A time-independent random variable x_t is exchangeable over time t and does not depend on any other random variable (including its own previous values). A (first-order) Markovian random variable x_{t+1} depends on its value x_t at the previous time step. In particular, the state variable s_{t+1} depends on s_t and the chosen action a_t. Variables can also be classified as endogenous and exogenous. The variable x_t is exogenous if its distribution is independent of $a_{t'}$ and $s_{t'} \setminus \{x'_t\}$ for all $t' \leq t$. Non-exogenous variables are endogenous. The key insight of this chapter is that if a variable is time-independent and exogenous, then it can be removed from the MFMC stitching calculation as follows.

Let us factor the MDP state s into two vectors of random variables: w, which contains the time-independent, exogenous state variables and x, which contains all of the other state variables (see Figure 3.1). In our wildfire suppression domain, the state of the forest from one time step to another is Markovian, but our policy decisions also depend on exogenous weather events such as rain, wind, and lightning.

We can formalize this factorization as follows.

Definition 3.4.1. A Factored Exogenous MDP is an MDP such that the state (x, w) and next state (x', w') are related according to

$$Pr(x', w'|x, w, a) = Pr(w')Pr(x'|x, w, a). \tag{3.1}$$

This factorization allows us to avoid computing similarity in the complete state space s. Instead we only need to compute the similarity of the Markov state x. Without the factorization, MFMC stitches (s, a) to the $(\tilde{s}, \tilde{a})$ in the database D that minimizes a distance metric Δ, where Δ has the form $\Delta((s, a), (\tilde{s}, \tilde{a})) \mapsto \mathbb{R}^+$. Our new algorithm, MFMCi, makes its stitching decisions using only the Markov state. It stitches the current state x by finding the tuple $(\tilde{x}, \tilde{w}, a, r, x')$ that minimizes the lower-dimensional distance metric $\Delta_i(x, \tilde{x})$. MFMCi then adopts $(\tilde{x}, \tilde{w})$ as the current state, computes the policy action $\tilde{a} = \pi(\tilde{x}, \tilde{w})$, and then makes a transition to x' with reward r. The rationale for replacing x by $\tilde{x}$ is the same as in MFMC, namely that it is the nearest state from the database D. The rationale for replacing w by $\tilde{w}$ is that both w and $\tilde{w}$ are exchangeable draws from the exogenous time-independent distribution $P(w)$, so they can be swapped without changing the distribution of simulated paths.

There is one subtlety that can introduce bias into the simulated trajectories. What happens when the action $\tilde{a} = \pi(\tilde{x}, \tilde{w})$ is not equal to the action a in the database tuple $(\tilde{x}, \tilde{w}, a, r, x', w')$? One approach would be to require that $a = \tilde{a}$ and keep rejecting candidate tuples until we find one that satisfies this constraint. We call this method, "Biased MFMCi", because doing this introduces a bias. Consider again the graphical model in Figure 3.1. When we use the value of a to decide whether to accept $\tilde{w}$, this couples $\tilde{w}$ and $\tilde{x}$ so that they are no longer

Algorithm 2: Populating D for MFMCi by sampling whole trajectories.

```
Input Parameters: Policy $\pi$, horizon $h$, trajectory count $n$, transition
  simulator $f_x$, reward simulator $f_r$, exogenous distribution $P(w)$,
  stochasticity distribution $P(z)$
Returns: $nh$ transition sets $B$
$D \leftarrow \emptyset$
while $|D| < nh$ do
    $x = f_x(\cdot, \cdot, \cdot, \cdot)$ // Draw initial Markov state
    $l = 0$
    while $l < h$ do
        $B \leftarrow \emptyset$
        $w \sim P(w)$
        $z \sim P(z)$
        for $a \in A$ do
            $r \leftarrow f_r(x, a, w, z)$
            $x' \leftarrow f_x(x, a, w, z)$
            $B \leftarrow B \cup \{(x, w, a, r, x')\}$
        end
        append($D$,$B$)
        $x \leftarrow B^{x'}_{\pi(x,w)}$
        $l \leftarrow l + 1$
    end
end
return($D$)
```

independent.

An alternative to Biased MFMCi is to change how we generate the database D to ensure that for every state $(\tilde{x}, \tilde{w})$, there is always a tuple $(\tilde{x}, \tilde{w}, a, r, x', w')$ for every possible action a. To do this, as we execute a trajectory following policy π, we simulate the result state (x', w') and reward r for each possible action a and not just the action $a = \pi(x, w)$ dictated by the policy. We call this method "Debiased MFMCi". This requires drawing more samples during database construction, but it restores the independence of $\tilde{w}$ from $\tilde{x}$.

3.4.1 MFMC with independencies (MFMCi)

For purposes of analyzing MFMCi, it is helpful to make the stochasticity of $P(s'|s,a)$ explicit. To do this, let z be a time-independent random variable distributed according to $P(z)$. Then we can "implement" the stochastic transition $P(s'|s,a)$ in terms of a random draw of z and a state transition *function* f_x as follows. To make a state transition from state $s = (w,x)$ and action a, we draw samples of both the exogenous variable $w' \sim P(w')$ and from $z \sim P(z)$ and then evaluate the function $x' = f_x(x,w,a,z)$. The result state s' is then (x',w'). Similarly, to model stochastic rewards, we can define the function f_r such that $r := f_r(x,w,a,z)$. This encapsulates all of the randomness in $P(s'|s,a)$ and $R(s,a)$ in the variables w' and z.

As noted in the previous section, stitching only on x can introduce bias unless we simulate the effect of every action a for every $x \in D$. It is convenient to collect together all of these simulated successor states and rewards into *transition sets*. Let $B(x,w)$ denote the transition set of tuples $\{(x,w,a,x',r)\}$ generated by simulating each action a in state (x,w). Given a transition set B, it is useful to define selector notation as follows. Subscripts of B constrain the set of matching tuples and superscripts indicate which variable is extracted from the matching tuples. Hence, $B_a^{x'}$ denotes the result state x' for the tuple in B that matches action a. With this notation, Algorithm 2 describes the process of populating the database with transition sets.

Algorithm 3: MFMCi

```
**Input Parameters:** Policy $\pi$, horizon $h$, starting state $x_0$, distance metric $\Delta_i(\cdot,\cdot)$, database $D$
**Returns:** $(x_0, w, a, r)_1, \ldots, (x', w', a', r')_h$
$t \leftarrow \emptyset$
$x \leftarrow x_0$
while length(t)$< h$ do
    $\hat{B} \leftarrow \operatorname{argmin}_{B \in D} \Delta_i(x, B^{x'})$
    $\hat{w} \leftarrow \hat{B}^w$
    $a \leftarrow \pi(x, \hat{w})$
    $r \leftarrow \hat{B}^r_a$
    append$(t, (x, w, a, r))$
    $D \leftarrow D \setminus \hat{B}$
    $x \leftarrow B^{x'}_a$
end
return$(t)$
```

Algorithm 3 gives the pseudo-code for MFMCi. Note that when generating multiple trajectories with Algorithm 3 for a single policy query, the transition sets are drawn without replacement. To estimate the cumulative return of policy π, we call MFMCi n times and compute the mean of the cumulative rewards of the resulting trajectories. We refer to this as the MFMCi estimate $V^{\pi}_{MFMCi}(s_0)$ of $V^{\pi}(s_0)$.

3.4.2 Bias and Variance Bound on $V^{\pi}_{MFMCi}(s_0)$

Fonteneau et al. [23, 24, 22] derived bias and variance bounds on the MFMC value estimate $V^{\pi}_{MFMC}(s_0)$. Here we rework this derivation to provide analogous bounds for $V^{\pi}_{MFMCi}(s_0)$. The Fonteneau et al. bounds depend on assuming Lipschitz smoothness of the functions f_s, f_r and the policy π. To do this, they require that the action space be continuous in a metric space A. We will impose the same requirement for purposes of analysis. Let L_f, L_R, and L_π be Lipschitz constants for the chosen norms $\|\cdot\|_S$ and $\|\cdot\|_A$ over the S and A spaces, as follows:

$$\|f_s(s,a,z) - f_s(s',a',z)\|_S \;\leq\; L_f(\|s-s'\|_S + \|a-a'\|_A) \tag{3.2}$$

$$|f_r(s,a,z) - f_r(s',a',z)| \;\leq\; L_R(\|s-s'\|_S + \|a-a'\|_A) \tag{3.3}$$

$$\|\pi(s) - \pi(s')\|_A \leq L_\pi(\|a-a'\|_A). \tag{3.4}$$

To characterize the database's coverage of the state-action space, let $\alpha_k(D)$ be the maximum distance from any state-action pair (s,a) to its k-th nearest neighbor in database D. Fonteneau, et al. call this the k-dispersion of the database.

Theorem 1. *[22] For any Lipschitz continuous policy π, let V^{π}_{MFMC} be the MFMC estimate of the value of π in s_0 based on n MFMC trajectories of length h drawn from database D. Under the Lipschitz continuity assumptions of Equations 3.2, 3.3, and 3.4, the bias and variance of V^{π}_{MFMC} are*

$$|V^{\pi}_{MFMC}(s_0) - V^{\pi}(s_0)| \leq C\alpha_{nh}(D) \tag{3.5}$$

$$Var^{\pi}_{n,D}(s_0) \leq \Big(\frac{\sigma^{\pi}_h(s_0)}{\sqrt{n}} + 2C\alpha_{nh}(D)\Big)^2 \tag{3.6}$$

where $\sigma^{\pi}_h(s_0)$ is the variance of the total reward for h-step trajectories under π when executed on the true MDP and C is defined in terms of the Lipschitz constants as

$$C = L_R \sum_{i=0}^{h-1} \sum_{j=0}^{h-i-1} [L_f(1 + L_\pi)]^j. \tag{3.7}$$

Now we derive analogous bias and variance bounds for $V^{\pi}_{MFMCi}(s_0)$. To this end, define two Lipschitz constants L_{F_i} and L_{R_i} such that the following conditions hold for the MDP:

$$\|f_x(x,a,w,z) - f_x(x',a,w,z)\|_X \;\leq\; L_{f_i}(\|x - x'\|_X) \tag{3.8}$$

$$|f_r(x,a,w,z) - f_r(x',a,w,z)| \;\leq\; L_{R_i}(\|x - x'\|_X). \tag{3.9}$$

Where $\|\cdot\|_X$ is the chosen norm over the X space.

Let $\alpha_{i,k}(D)$ be the maximum distance from any Markov state x to its k-th nearest neighbor in database D for the distance metric Δ_i. Further, let x_0 be the initial Markov state. Then we have

Corollary 1. *For any Lipschitz continuous policy π, let V^{π}_{MFMCi} be the MFMCi estimate of the value of π in x_0 based on n MFMCi trajectories of length h drawn from database D. Under the Lipschitz continuity assumptions of Equations 3.8* *and 3.9, the bias and variance of* V^{π}_{MFMCi} *are*

$$|V^{\pi}_{MFMCi}(s_0) - V^{\pi}(s_0)| \leq C_i \alpha_{i,nh}(D) \tag{3.10}$$

$$Var^{\pi}_{i,n,D}(x_0) \leq \Big(\frac{\sigma^{\pi}_{h}(x_0)}{\sqrt{n}} + 2C_i \alpha_{i,nh}(D)\Big)^2 \tag{3.11}$$

where C_i *is defined as*

$$C_i = L_{R_i} \sum_{b=0}^{h-1} \sum_{j=0}^{h-b-1} [L_{f_i}]^j. \tag{3.12}$$

Proof. (Sketch) The result follows by observing that because there is always a matching action for each transition set, a will equal a' and $\|a - a'\|_A$ will be zero, so we can eliminate L_{π}. Similarly, because we can factor out w, we only match on x, so we can replace L_f with L_{f_i} and replace the norms with respect to S by the norms with respect to X. Finally, as we argued above, by using transition sets we do not introduce any added bias by adopting w instead of matching against it. Formally, we can view this as converting w from being an observable exogenous variable to being part of the unobserved exogenous source of stochasticity z. With these changes, the proof of Fonteneau, et al., holds. $\square$

We believe that similar proof techniques can bound the bias and variance of estimates of the quantiles of $P(v^j(s_t))$ for properties $v^j(s_t)$ of the state at time step t. We leave this to future work.

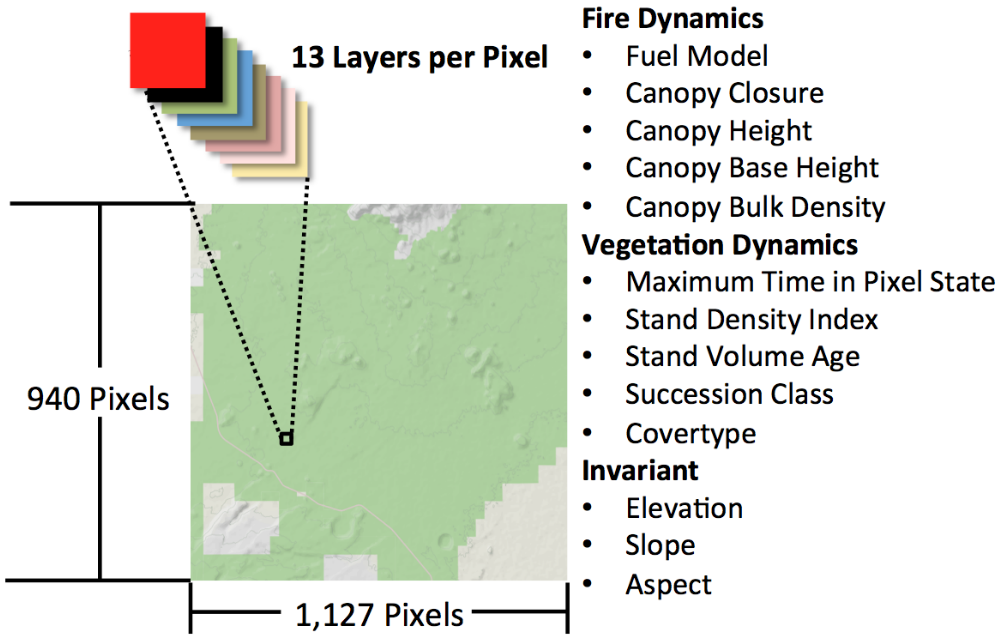


Figure 3.2: The landscape totals approximately one million pixels, each of which has 13 state variables that influence the spread of wildfire on the landscape. We use summary statistics of the dynamic state variables in MFMC's distance metric. (Map is copyright of OpenStreetMap contributors)

3.5 Experimental Evaluation

In our experiments we test whether we can generate accurate trajectory visualizations for a wildfire, timber, vegetation, and weather simulator [27]. The aim of the wildfire management simulator is to help US Forest Service land managers decide whether suppress a wildfire on National Forest lands. Each 100-year trajectory takes up to 7 hours to simulate.

Figure 3.2 shows a snapshot of the landscape as generated by the simulator. The landscape is comprised of approximately one million pixels, each with 13 state variables. When a fire is ignited by lightning, the policy must choose between two actions: *Suppress* (fight the fire) and *Let Burn* (do nothing). Hence, $|A| = 2$.

The simulator spreads wildfires with the FARSITE fire model [21] according to the surrounding pixel variables (X) and the hourly weather. Weather variables include *hourly wind speed, hourly wind direction, hourly cloud cover, daily maximum/minimum temperature, daily maximum/minimum humidity, daily hour of maximum/minimum temperature, daily precipitation, and daily precipitation duration.* These are generated by resampling from 25 years of observed weather [62]. MFMCi can treat the weather variables and the ignition location as exogenous variables because the decision to fight (or not fight) a fire has no influence on weather or ignition locations. Further, changes in the Markov state do not influence the weather or the spatial probability of lightning strikes.

After computing the extent of the wildfire on the landscape, the simulator applies a cached version of the Forest Vegetation Simulator [17] to update the vegetation of the individual pixels. Finally, a harvest scheduler selects pixels to harvest for timber value.

We constructed three policy classes that map fire ignitions to fire suppression decisions. We label these policies INTENSITY, FUEL, and LOCATION. The INTENSITY policy suppresses fires based on the weather conditions at the time of the ignition and the number of days remaining in the fire season. The FUEL policy suppresses fires when the landscape accumulates sufficient high-fuel pixels. The LOCATION policy suppresses fires that are ignited in the top half of the landscape, but allows fires on the bottom half of the landscape to burn (which mimics the situation that arises when houses and other buildings occupy part of the landscape).

We selected these policy classes because they are functions of different components of the Markov and exogenous state. The INTENSITY policies are a function of the exogenous variables and should be difficult for MFMC, because the sequence of actions along a trajectory will be driven primarily by the stochasticity of the weather circumstances. This contrasts with the FUEL policy, which should follow the accumulation of vegetation between time steps in the Markov state. Finally, the LOCATION policy should produce landscapes that are very different from the other two policy classes as fuels become spatially imbalanced in the Markov state.

The analysis of Fonteneau et al. [23] assumes the database is populated with state-action transitions covering the entire state-action space. The dimensionality of the wildfire state space makes it impossible to satisfy this assumption. We focus sampling on states likely to be entered by future policy queries by seeding the database with one trajectory for each of 360 policies whose parameters are sampled according to a grid over the INTENSITY policy space. The INTENSITY policy parameters include a measure of the weather conditions at the time of ignition known as the Energy Release Component (ERC) and a measure of the seasonal risk in the form of the calendar day. These measures are drawn from $\big[0, 100\big]$ and $\big[0, 180\big]$, respectively.

The three policy classes are very different from each other. One of our goals is to determine whether MFMCi can use state transitions generated from the INTENSITY policy to accurately simulate state transitions under the FUEL and LOCATION policies. We evaluate MFMCi by generating 30 trajectories for each policy from the ground truth simulator.

For our distance metric Δ_i, we use a weighted Euclidean distance computed over the mean/variance standardized values of the following landscape features: *Canopy Closure, Canopy Height, Canopy Base Height, Canopy Bulk Density, Stand Density Index, High Fuel Count, and Stand Volume Age.* All of these variables are given a weight of 1. An additional feature, the time step (*Year*), is added to the distance metric with a very large weight to ensure that MFMCi will only stitch from one state to another if the time steps match. Introducing this non-stationarity ensures we exactly capture landscape growth stages for all pixels that do not experience fire.

Our choice of distance metric features is motivated by the observation that the risk profile (the likely size of a wildfire) and the vegetation profile (the total tree cover) are easy to capture in low dimensions. If we instead attempt to capture the likely size of a specific fire, we need a distance metric that accounts for the exact spatial distribution of fuels on the landscape. Our distance metric successfully avoids directly modeling spatial complexity.

To visualize the trajectories, we employ the visualization tool MDPvis ([37] and Chapter 2). The key visualization in MDPvis is the fan chart, which depicts the time evolution of various state, action, and reward variables for the set of trajectories as a function of time (see Figure 3.3). Each fan chart plots the distribution of the value of one variable in terms of a set of quantiles.

To evaluate the quality of the fan charts generated using surrogate trajectories, we define visual fidelity error in terms of the difference in vertical position between the true median and its position under the surrogate. Specifically, we

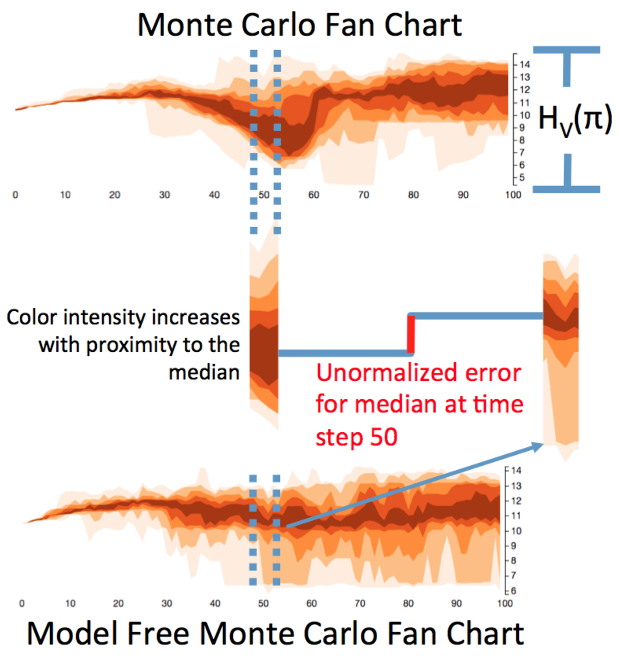


Figure 3.3: Top: A fan chart generated by Monte Carlo simulations from the expensive simulator. Bottom: A fan chart generated from the MFMC surrogate model. x axis is the time step and y axis is the value of the state variable at each time step. Each change in color shows a quantile boundary for a set of trajectories generated under policy π. Middle: Error measure is the distance between the median of the Monte Carlo simulations (left) and the median of the MFMC/MFMCi surrogate simulations (right). The error is normalized across fan charts according to $H_v(\pi)$, which is the Monte Carlo fan chart height for policy π and variable v.

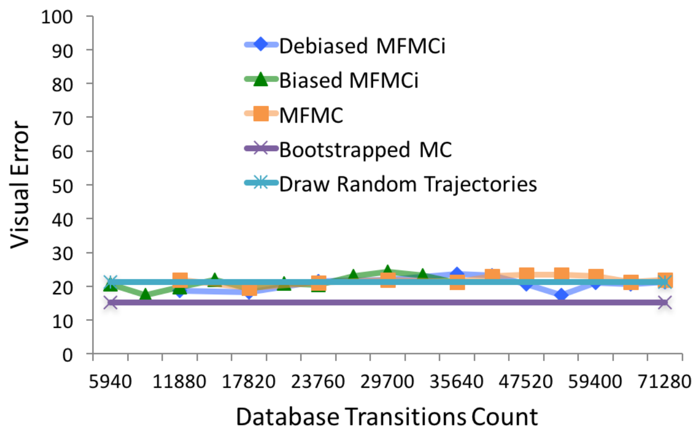


Figure 3.4: Visual fidelity errors for a weather *intensity* policy class. Fires are suppressed based on a combination of the weather and how much time is left in the fire season.

define error(v, t) as the offset between the correct location of the median and its MFMCi-modeled location for state variable v in time step t. We normalize the error by the height of the fan chart for the rendered policy $(H_v(\pi))$. The weighted error is thus $\sum_{v \in S} \sum_{t=0}^{h} \frac{\text{error}(v,t)}{H_v(\pi)}$.

This error is measured for 20 variables related to the counts of burned pixels, fire suppression expenses, timber loss, timber harvest, and landscape ecology.

3.5.1 Experimental Results

We evaluated the visual fidelity under three settings: (a) debiased MFMCi (exogenous variables excluded from the distance metric Δ_i; debiasing tuples included in the database D), (b) MFMC (exogenous variables included in Δ), and (c) biased

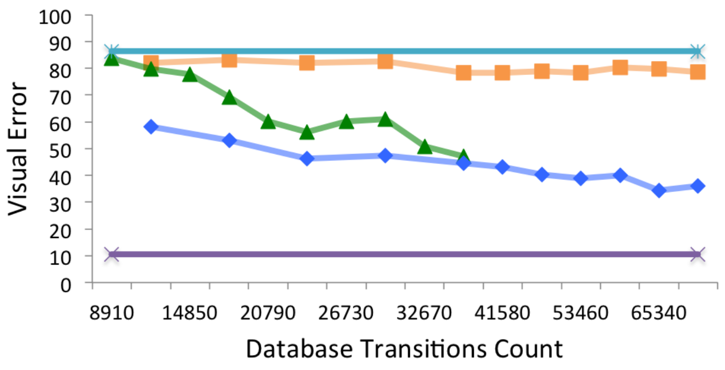


Figure 3.5: Visual fidelity errors for a ignition *location* policy class. Fires are always suppressed if they start on the top half of the landscape, otherwise they are always allowed to burn.

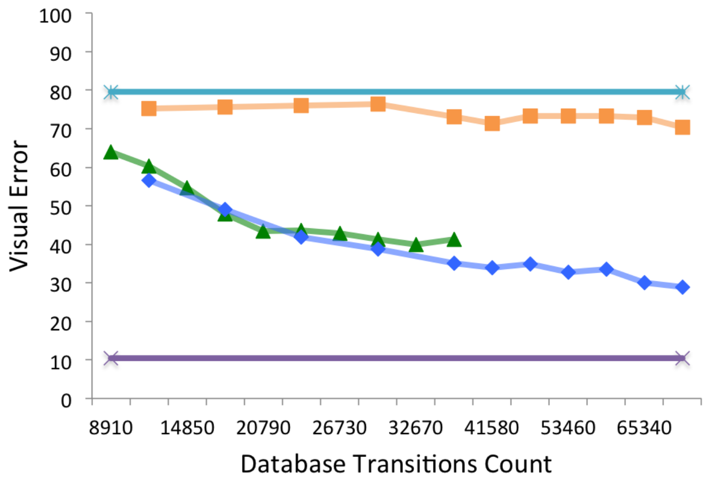


Figure 3.6: Visual fidelity errors for a *fuel* accumulation policy class. Fires are always suppressed if the landscape is at least 30 percent in high fuels, otherwise the fire is allowed to burn.

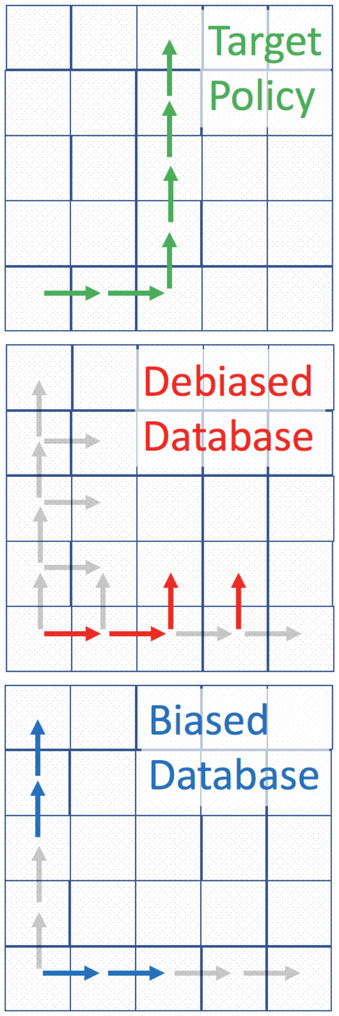


Figure 3.7: Example of MFMC's autoregressive tendency for a grid world domain where the only available actions are "up" and "right". The green arrows show a trajectory that we would like to synthesize from two different MFMC databases where the distance metric is Euclidean with arbitrarily large weight given to the time step and action. The gray arrows show the grid world transitions in the two databases. In the debiased database the stitching operation will stay on the rightward trajectory despite there being transitions that more closely track the target trajectory. The biased database forces the stitching operation to hop to the policy more consistent with the target policy. In some instances it is better to bias the exogenous variables than repeatedly stitch to the same trajectories.

MFMCi (exogenous variables excluded from Δ_i and the extra debiasing tuples removed from D). We also compare against two baselines that explore the upper and lower bounds of the visual error. First, we show that the lower bound on visual error is not zero. Although each policy has true quantile values at every time step, estimating these quantiles with 30 trajectories is inherently noisy. We estimate the achievable visual fidelity by bootstrap resampling the 30 ground truth trajectories and report the average visual fidelity error. Second, we check whether the error introduced by stitching is worse than visualizing a set of random database trajectories. Thus the bootstrap resample forms a lower bound on the error, and comparison to the random trajectories detects stitching failure. Figures 3.4, 3.5, 3.6 plot "learning curves" showing the visualization error as a function of the size of the database D. The ideal learning curve should show a rapid decrease in visual fidelity error as $|D|$ grows.

3.6 Discussion

For each policy class, we chose one target policy from that class and measured how well the behavior of that policy could be simulated by our MFMC variants. Recall that the database of transitions was generated using a range of INTENSITY policies. When we apply the MFMC variants to generate trajectories for an INTENSITY policy, all methods (including random trajectory sampling) produce an accurate representation of the median for MDPVIS. When the database trajectories do not match the target policy, MFMCi outperforms MFMC. For some policies, the debiased database outperforms the biased databases, but the difference decreases with additional database samples. Next we explore these findings in more depth.

INTENSITY **Policy.** Figure 3.4 shows the results of simulating an INTENSITY policy that suppresses all fires that have an ERC between 75 and 95, and ignite after day 120. This policy suppresses approximately 60 percent of fires. There are many trajectories in the database that agree with the target policy on the majority of fires. Thus, to simulate the target policy it is sufficient to find a policy with a high level of agreement and then sample the entire trajectory. This is exactly what MFMC, MFMCi, and Biased MFMCi do. All of them stitch to a good matching trajectory and then follow it, so they all give accurate visualizations as indicated by the low error rate in Figure 3.4. Unsurprisingly, we can approximate INTENSITY policies from a very small database D built from other INTENSITY policies.

LOCATION **Policy.** Figure 3.5 plots the visual fidelity error when simulating a LOCATION policy from the database of INTENSITY policy trajectories. When D is small, the error is very high. MFMC is unable to reduce this error as D grows, because its distance metric does not find matching fire conditions for similar landscapes. In contrast, because the MFMCi methods are matching on the smaller set of Markov state variables, they are able to find good matching trajectories. The debiased version of MFMCi outperforms the biased version for the smaller database sizes. In the biased version, the matching operation repeatedly stitches over long distances to find a database trajectory with a matching action. Debiased MFMCi avoids this mistake. This explains why debiased MFMCi rapidly decreases the error while biased MFMCi takes a bit longer but then catches up at roughly

$|D| =$40,000.

FUEL **Policy.** The FUEL policy shows a best case scenario for the biased database. Within 7 time steps, fuel accumulation causes the policy action to switch from let-burn-all to suppress-all. Since all of the trajectories in the database have a consistent probability of suppressing fires throughout all 100 years, the ideal algorithm will select a trajectory that allows all wildfires to burn for 7 years (to reduce fuel accumulation), then stitch to the most similar trajectory in year 8 that will suppress all future fires. The biased database will perform this "policy switching" by jumping between trajectories to find one that always performs an action consistent with the current policy.

Policy switching is preferable to the debiased database in some cases. To illustrate this, consider the grid world example in Figure 3.7. It shows that debiased samples can offer stitching opportunities that prevent policy switching and hurt the results.

In summary, our experiments show that MFMCi is able to generalize across policy classes and that it requires only a small number of database trajectories to accurately reproduce the median of each state variable at each future time step. In general, it appears to be better to create a debiased database than a biased database having the same number of tuples.

Our results allow our forestry stakeholders to interactively explore a range of policies within MDPVIS. We further show how these results support interactive policy optimization for user-specified reward functions in Chapter 4.

4 Fast Policy Optimization with SMAC

4.1 Introduction

When lightning ignites a fire in the US National Forest system, the forest manager must decide whether to suppress that fire or allow it to burn itself out. This decision has immediate costs in terms of fire fighting expenses and smoke pollution and long-term benefits, including increased timber harvest revenue and reduced severity of future fires. Different stakeholders place different values on these various outcomes, and this leads to contentious and difficult policy debates. In the US Pacific Northwest, a period in the 1990s is referred to as the "Timber Wars" because of the troubling and occasionally violent conflicts that arose between stakeholder groups over forest management policies during that period. This is typical of many ecosystem management problems—the complexity of ecosystem dynamics and the broad array of interested parties makes it difficult to identify politically-feasible policies.

One way that visual analytics can help is to provide a high-fidelity simulation environment in which stakeholders can explore the policy space, experiment with different reward functions, compute the resulting optimal policies, and visualize the behavior of the ecosystem when it is managed according to those policies. This process can elicit missing aspects of the reward function, and it can help the stakeholders reach a policy consensus that is informed by the best available ecosystem models. To create such a simulation environment, we need a tool that meets the following requirements:

i. **Modifiability**: users should be able to modify the reward function to represent the interests of various stakeholders.

ii. **Automatic Optimization**: users should be able to optimize policies for the updated reward functions without the involvement of a machine learning researcher.

iii. **Visualization**: users should be able to explore the behavior of the system when it is controlled by the optimized policies.

iv. **Interactivity**: all these tasks should be performed at interactive speeds.

MDPvis supports requirements i and iii, but it does not support the optimization capability needed for requirement ii. Furthermore, the full-fidelity wildfire simulator is very slow, so even if we had an optimization algorithm for this noisy, high-dimensional problem, the optimization could not meet the interactive speeds needed for requirement iv.

This chapter simultaneously addresses requirements ii and iv by applying the SMAC [28] black-box function optimization algorithm to the surrogate developed in Chapter 3. SMAC is similar to Bayesian methods for black box optimization. However, unlike those methods, SMAC does not employ a Gaussian process to model the black box function. Instead, it fits a random forest ensemble [11]. This has three important benefits. First, it does not impose any smoothness assumption on the black box function. We will see below that wildfire policies are naturally expressed in the form of decision trees, so they are highly non-smooth. Second, SMAC does not require the design of a Gaussian process kernel. This makes it more suitable for application by end users such as our policy stakeholders. Third, the CPU time required for SMAC scales as $O(n \log n)$ where n is the number of evaluations of the black box function, whereas standard GP methods scale as $O(n^3)$ because they must compute and invert the kernel matrix.

This chapter makes two contributions. First, it shows that SMAC can rapidly find high-scoring policies for a range of different reward functions that incorporate both short-term and long-term rewards. Second, it confirms that this is possible even though SMAC is using an approximate surrogate for the high-fidelity simulator. Taken together, these contributions mark the first successful optimization a wildfire suppression policy for a full 100-year planning horizon. Since SMAC is also fast, these contributions also produce an algorithm for optimizing policies within interactive time frames.

The chapter is structured as follows. First, we provide an overview of direct policy search and SMAC. This is followed by a description of the fire management problem including a review of the different components of the reward function and the relative weight that different stakeholder constituencies place on these components. We then describe the parameterized policy representation for wildfire management policies. The results of applying SMAC to optimize these policies are shown next. Finally, the surrogate estimates of the values of these policies are checked by running them on the full-fidelity simulator. The results confirm the accuracy of the surrogate estimates.

4.2 Direct Policy Search Methods

We work with the standard finite horizon discounted MDP introduced in Chapter 2. Let Π be a class of deterministic policies with an associated parameter space Θ. Each parameter vector $\theta \in \Theta$ defines a policy $\pi_\theta : S \mapsto A$ that specifies what action to take in each state. Let $\tau = \langle s_0, s_1, \ldots, s_h \rangle$ be a trajectory generated by drawing a state $s_0 \sim P_0(s_0)$ according to the starting state distribution and then following policy π_θ for h steps. Let $\rho = \langle r_0, \ldots, r_{h-1} \rangle$ be the corresponding sequence of rewards. Both τ and ρ are random variables because they reflect the stochasticity of the starting state and the probability transition function. Let V_θ define the expected cumulative discounted return of applying policy π_θ starting in a state $s_0 \sim P_0(s_0)$ and executing it for h steps:

$$V_\theta = \mathbb{E}_\rho[r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots + \gamma^{h-1} r_{h-1}]$$

The goal of direct policy search is to find θ^* that maximizes the value of the corresponding policy:

$$\theta^* = \underset{\theta \in \Theta}{\operatorname{argmax}}\, V_\theta.$$

Two prominent forms of policy search are policy gradient methods and sequential model-based optimization. Policy gradient methods [63, 55, 15, 51] estimate the gradient of V_θ with respect to θ and then take steps in parameter space to ascend the gradient. This is often challenging because the estimate is based on Monte Carlo samples of τ and because gradient search only guarantees to find a local optimum.

Sequential model-based optimization methods [32, 43, 68, 28, 53, 64, 57] construct a model of V_θ called a Policy Value Model and denoted $PVM(\theta)$. The PVM estimates both the value of V_θ and a measure of the uncertainty of that value. The most popular form of PVM is the Gaussian Process, which models V_θ as the GP mean and the uncertainty as the GP variance. The basic operation of sequential model-based optimization methods is to select a new point θ at which to query the PVM, observe the value of V_θ at that point (e.g., by simulating a trajectory τ using π_θ), and then update the PVM to reflect this new information. In Bayesian Optimization, the PVM is initialized with a prior distribution over possible policy value functions and then updated after each observation by applying Bayes rule. The new points θ are selected by invoking an *acquisition function*.

SMAC [28] is a sequential model-based optimization method in which the PVM is a random forest of regression trees. The estimated value of V_θ is obtained by "dropping" θ through each of the regression trees until it reaches a leaf in each tree and then computing the mean and the variance of the training data points stored in all of those leaves. In each iteration, SMAC evaluates V_θ at 10 different values of θ, adds the observed values to its database R of (θ, V_θ) pairs, and then rebuilds the random forest.

SMAC chooses 5 of the 10 θ values with the goal of finding points that have high "generalized expected improvement". The (ordinary) expected improvement at point θ is the expected increase in the maximum value of the PVM that will be observed when we measure V_θ under the assumption that V_θ has a normal distribution whose mean is μ_θ (the current PVM estimate of the mean at θ) and whose variance is σ_θ^2 (the PVM estimate of the variance at θ). The expected improvement at θ can be computed as

$$EI(\theta) := \mathbb{E}\big[I(\theta)\big] = \sigma_\theta\big[z\cdot\Phi(z) + \phi(z)\big], \tag{4.1}$$

where $z := \frac{\mu_\theta - f_{max}}{\sigma_\theta}$, f_{max} is the largest known value of the current PVM, Φ denotes the cumulative distribution function of the standard normal distribution, and ϕ denotes the probability density function of the standard normal distribution [29].

The generalized expected improvement (GEI) is obtained by computing the expected value of $I(\theta)$ raised to the g-th power. In SMAC, g is set to 2. Hutter et al. [28] show that this can be computed as

$$GEI(\theta) = \mathbb{E}\big[I^2(\theta)\big] = \sigma_\theta^2\big[(z^2+1)\cdot\Phi(z) + z\cdot\phi(z)\big]. \tag{4.2}$$

Ideally, SMAC would find the value of θ that maximizes $GEI(\theta)$ and then evaluate V_θ at that point. However, this would require a search in the high-dimensional space of Θ, and it would also tend to focus on a small region of Θ. Instead, SMAC employs the following heuristic strategy to find 10 candidate values of θ. First, it performs a local search in the neighborhood of the 10 best known values of θ in the PVM. This provides 10 candidate values. Next, it randomly generates another 10,000 candidate θ vectors from Θ and evaluates the GEI of each of them. Finally, it chooses the 5 best points from these 10,010 candidates and 5 points sampled at random from the 10,000 random candidates, and evaluates V_θ at each of these 10 points. This procedure mixes "exploration" (the 5 random points) with "exploitation" (the 5 points with maximum GEI), and it has been found empirically to work well.

Hutter et al. [28] prove that the SMAC PVM is a consistent estimator of V and that given an unbounded number of evaluations of V, it finds the optimal value θ^*.

4.3 The Wildfire Management Domain

To evaluate our methods, we have selected a portion of the Deschutes National Forest in Eastern Oregon. This area is being managed with the goal of restoring the landscape to the condition it was believed to be in prior to the arrival of Europeans. Figure 3.2 shows a map of the study site. It is comprised of approximately one million pixels, each described by 13 state variables.

We employ the high-fidelity simulator described in Houtman et al. [27], which combines a simple model of the spatial distribution of lightning strikes (based on historical data) with the state-of-the-art Farsite fire spread simulator [21], a fire duration model [20], and the high-resolution FVS forest growth simulator [17]. Weather is simulated by resampling from the historical weather time series observed at a nearby weather station.

The MDP advances in a sequence of decision points. Each decision point corresponds to a lightning-caused ignition, and the MDP policy must decide between two possible actions: *Suppress* (fight the fire) and *Let Burn* (do nothing). Hence, $|A| = 2$. Based on the chosen action and the (simulated) weather, the intensity, spread, and duration of the fire is determined by the simulator. In order to capture the long-term impacts of our policy decisions, we employ a planning horizon of $h = 100$ years.

Unfortunately, the simulator is very expensive. Simulating a 100-year trajectory of fires can take up to 7 hours of clock time. This is obviously too slow for interactive use. We therefore have adopted the surrogate modeling approach of Chapter 3 during the optimization.

Wildfires produce a variety of immediate and long term losses and benefits. For market-based rewards, such as suppression costs and timber revenues, there is a single defensible reward function. Since wildfire decisions also affect many outcomes for which there is no financial market, such as air, water, ecology, and recreation, there are potentially many different composite reward functions. A benefit of the random forest method is the ability to change the reward function and re-optimize without making any assumptions about the character of the response surface V. In our experimental evaluation, we optimize and validate policies for four different reward functions as an approximation of different stakeholder interests. Table 4.1 details each of these reward function constituencies.

The reward functions are compositions of five different reward components. The *Suppression* component gives the expenses incurred for suppressing a fire.

Constituency	Suppression Costs	Timber Revenues	Ecology Target	Air Quality	Recreation Target
Composite	✓	✓	✓	✓	✓
Politics	-	✓	✓	✓	✓
Home Owners	-	-	-	✓	✓
Timber	✓	✓	-	-	-

Table 4.1: Components of each reward function. The "politics" constituency approximates a decision maker that is not responsible for funding firefighting operations. The "home owner" constituency only cares about air quality and recreation. The "timber" companies only care about how much timber they harvest, and how much money they spend protecting that timber. The "composite" reward function takes an unweighted sum of all the costs and revenues produced for the constituencies. Additional reward functions can be specified by users interactively within MDPvis.

Fire suppression expenses increase with fire size and the number of days the fire burns. Without fire suppression effort, the fire suppression costs are zero, but the fire generally takes longer to self-extinguish. *Timber* harvest values are determined by the number of board feet harvested from the landscape. A harvest scheduler included in the simulator determines the board feet based on existing forest practice regulations. Generally we would expect timber harvest to increase with suppression efforts, but complex interactions between the harvest scheduler and tree properties (size, age, species) often results in high timber harvests following a fire. *Ecological* value is a function of the squared deviation from an officially-specified target distribution of vegetation on the landscape known as the "restoration target." Since our landscape begins in a state that has a recent history of fire suppression efforts, there is much more vegetation than the target. A good way to reach the target is to allow wildfires to burn, but increased timber harvest can also contribute to this goal. *Air Quality* is a function of the number of days a wildfire burns. When fires burn, the smoke results in a large number of home-owner complaints. We encode this as a negative reward for each smoky day. Finally, the *recreation* component penalizes the squared deviation from a second vegetation target distribution—namely, one preferred by people hiking and camping. This distribution consists of old, low-density ponderosa pine trees. Frequent, low-intensity fires produce this kind of distribution, because they burn out the undergrowth while leaving the fire-resilient ponderosa pine unharmed. If we optimize for any single reward component, the optimal policy will tend to be one of the trivial policies "suppress-all" or "letburn-all". When multiple reward components are included, the optimal policy still tends to either suppress or let burn most fires by default, but it tries to identify exceptional fires where the default should be overridden. See Houtman et al. [27] for a discussion of this phenomenon.

A natural policy class in this domain takes the form of a binary decision tree as shown in Figure 4.1. At each level of the tree, the variable to split on is fixed in this policy class. With the exception of the very first split at the root, which has a hard-coded threshold, the splitting thresholds $\theta_1, \ldots, \theta_{14}$ are all adjusted by the policy optimization process. Moving from the top layer of the tree to the bottom, the tree splits first on whether the fire will be extinguished within the next 8 days by a "fire-ending weather event" (i.e., substantial rain or snowfall). The threshold value of 8 is fixed (based on discussions with land managers and on the predictive scope of weather forecasts). The next layer splits on the state of fuel accumulation on the landscape. The fuel level is compared either to θ_1 (left branch, no weather event predicted within 8 days) or θ_2 (right branch; weather predicted within 8 days). When the fuel level is larger than the corresponding threshold, the right branch of the tree is taken. The next layer splits on the intensity of the fire at the time of the ignition. In this fire model, the fire intensity is quantified in terms of the Energy Release Component (ERC), which is a composite measure of dryness in fuels. Finally, the last layer of the tree asks whether the current date is close to the start or end of the fire season. Our study region in Eastern Oregon is prone to late spring and early fall rains, which means fires near the boundary of the fire season are less likely burn very long. We note that this policy function is difficult for gradient-based policy search, because it is not differentiable and exhibits complex responses to parameter changes.

4.4 Experiments

To train the MFMCi surrogate model, we sampled 360 policies from a policy class that suppresses wildfires based on the ERC at the time of the ignition and the day of the ignition (see Chapter 3 for details). Every query to the MFMCi surrogate generates 30 trajectories, and the mean cumulative discounted reward is returned as the observed value of V_θ.

We apply SMAC with its default configuration. When SMAC grows a random regression tree for its PVM, there is a parameter that specifies the fraction of the parameter dimensions (i.e., $\theta_1, \ldots, \theta_{14}$) that should be considered when splitting a

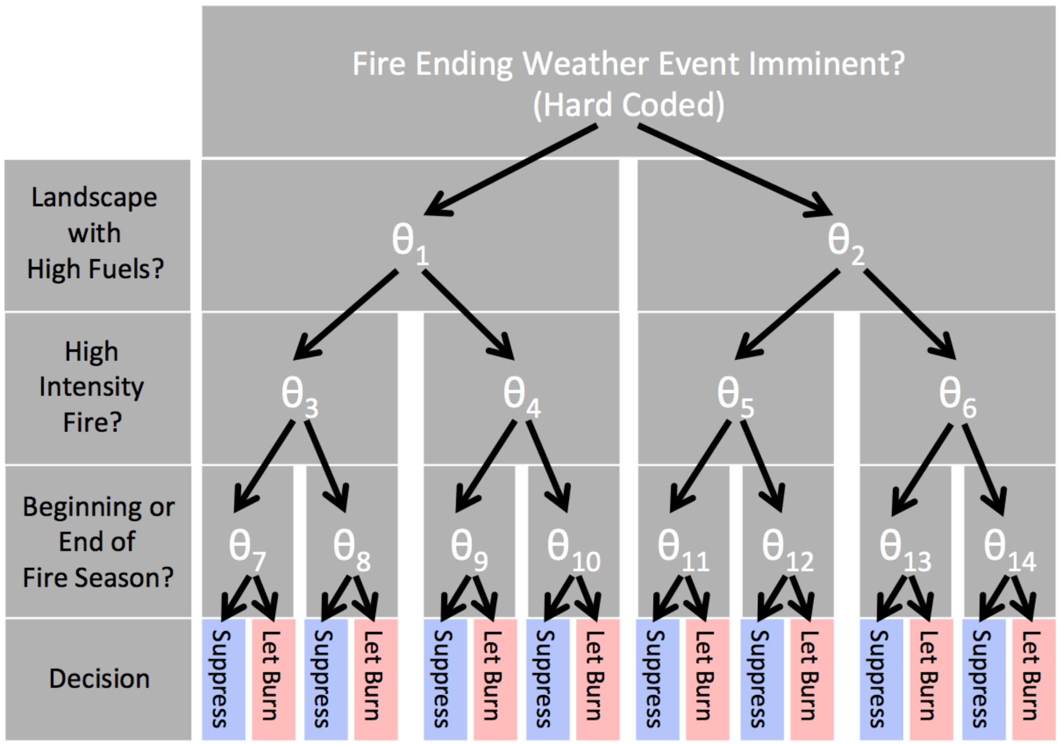

Fire Ending Weather Event Imminent?
(Hard Coded)
Landscape with High Fuels?
High Intensity Fire?
Beginning or End of Fire Season?
Decision
θ1
θ2
θ3
θ4
θ5
θ6
θ7
θ8
θ9
θ10
θ11
θ12
θ13
θ14
Suppress
Let Burn

Figure 4.1: The layers of the decision tree used to select wildfires for suppression. The top layer splits on whether the fire will likely be extinguished by a storm in the next 8 days regardless of the suppression decision. The next layers then have 14 parameters for the number of pixels that are in high fuel (parameters 1 and 2, $\big[0, 1000000\big]$), the intensity of the weather conditions at the time of the fire (3 through 6, $\big[0, 100\big]$), and a threshold that determines whether the fire is close to either the start or end of the fire season (7 through 14, $\big[0, 90\big]$).

regression tree node. We set this parameter to 5/6. A second parameter determines when to stop splitting, namely: a random forest node can only be split if it contains at least 10 examples. Finally, the size of the random forest is set to 10 trees.

Figure 4.2 shows the results of applying SMAC to find optimal policies for the four reward function constituencies. The left column of plots show the order in which SMAC explores the policy parameter space. The vertical axis is the estimated cumulative discounted reward, and the point that is highest denotes the final policy output by SMAC. Blue points are policy parameter vectors chosen by the GEI acquisition function whereas red points are parameter vectors chosen by SMAC's random sampling process. Notice that in all cases, SMAC rapidly finds a fairly good policy. The right column of plots gives us some sense of how the different policies behave. Each plot sorts the evaluated policy parameter vectors according to the percentage of fires that each policy suppresses. In 4.2(b), we see that the highest-scoring policies allow almost all fires to burn, whereas in 4.2(f), the highest-scoring policies suppress about 80% of the fires.

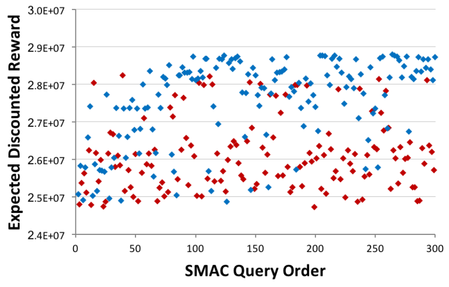
Let us examine these policies in more detail. The optimal policies for the *politics* and *timber* reward constituencies allow most wildfires to burn, but for different reasons. For the *politics* constituency, it is the Ecological reward that encourages this choice, whereas for the *timber* constituency, it is the increased harvest levels that result. The *composite* reward function produces a very similar optimal policy, presumably because it contains both the Ecological and Harvest reward components. These results indicate that timber company and political interests largely coincide insofar as fire policy is concerned.

The most interesting case is the *home owner* constituency reward function, which seeks to minimize smoke (which suggests suppressing all fires) and maximize recreation value (which suggests allowing fires to burn the understory occasionally). We can see in 4.2(f) that the best policy found by SMAC allows 20% of fires to burn and suppresses the remaining 80%.

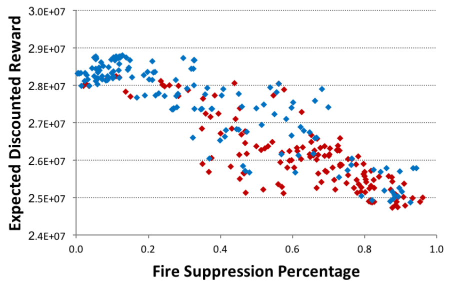
These results agree with our intuition for each reward constituency and provide evidence that SMAC is succeeding in optimizing these policies. However, the expected discounted rewards in Figure 4.2 are estimates obtained from the modified MFMC surrogate model. To check that these estimates are valid, we invoked each optimal policy on the full simulator at least 50 times and measured the cumulative discounted reward under each of the reward functions. We also evaluated the Suppress All and Let Burn All policies for comparison. The results are shown in Figure 4.3.

Each panel of boxplots depicts the range of cumulative returns for each policy on one reward function. For the policy that SMAC constructed, we also plot a dashed red line showing the MFMC estimate of the return. In all cases, the estimates are systematically biased high. This is to be expected, because any optimization against a noisy function will tend to fit the noise and, hence, over-estimate the true value of the function. Nonetheless, the MFMC estimates all fall within the inter-quartile range of the full simulator estimates. This confirms that the MFMC estimates are giving an accurate picture of the true rewards.

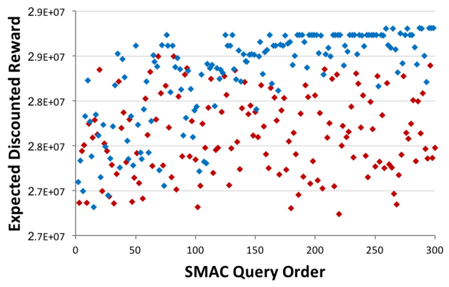
Note that because of the stochasticity of this domain, using only 50 trajectories from the full simulator is in general not sufficient to determine which policy is



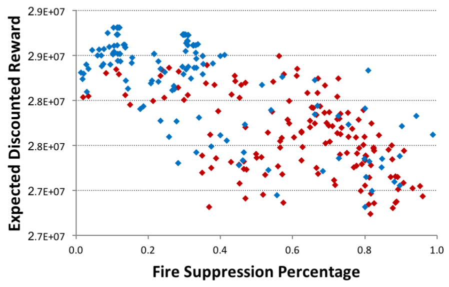
(a) Composite reward function.



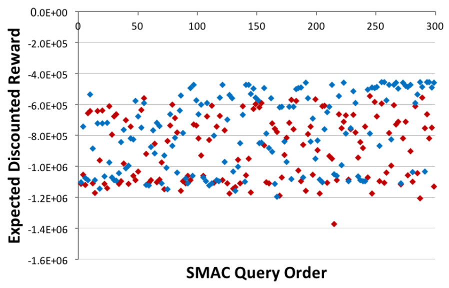
(b) Composite reward function.



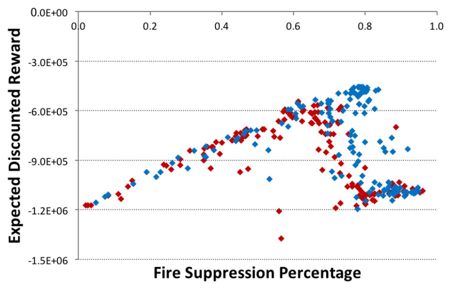
(c) Politics reward function.



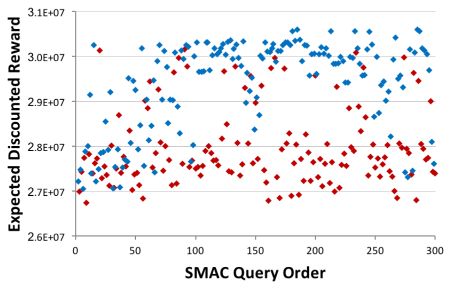
(d) Politics reward function.



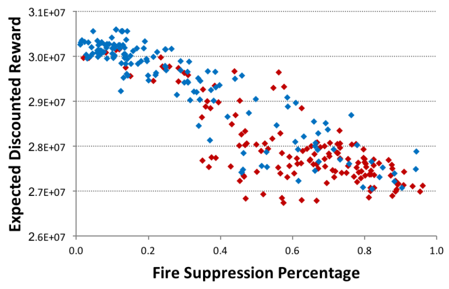
(e) Home owner reward function.



(f) Home owners reward function.

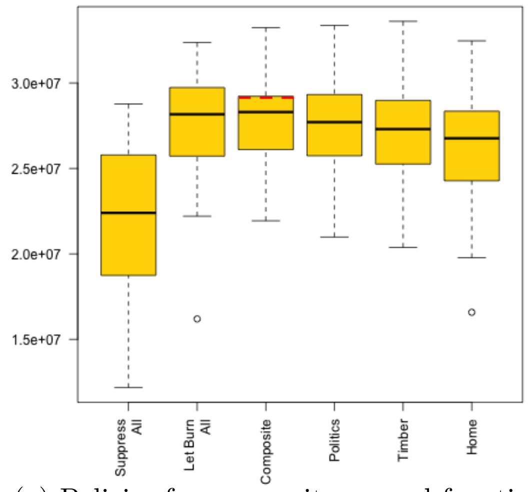


(g) Timber reward function.

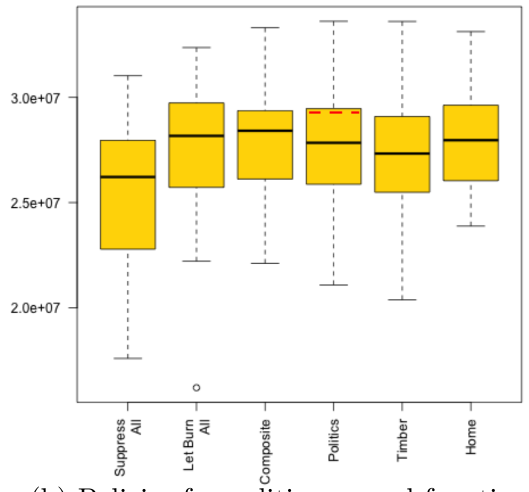


(h) Timber reward function.

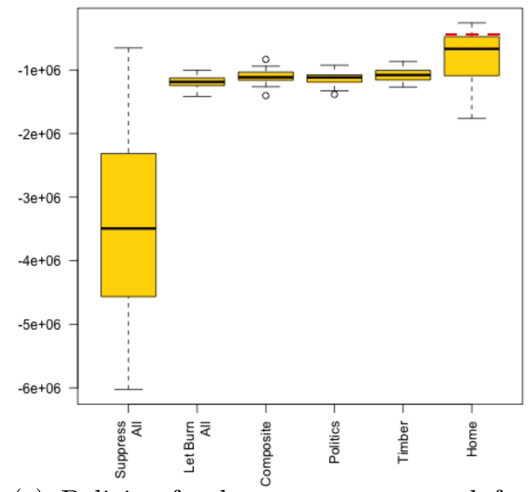
Figure 4.2: Average reward achieved for 30 trajectories. Blue diamonds are selected by the EI heuristic and red diamonds are randomly sampled points.



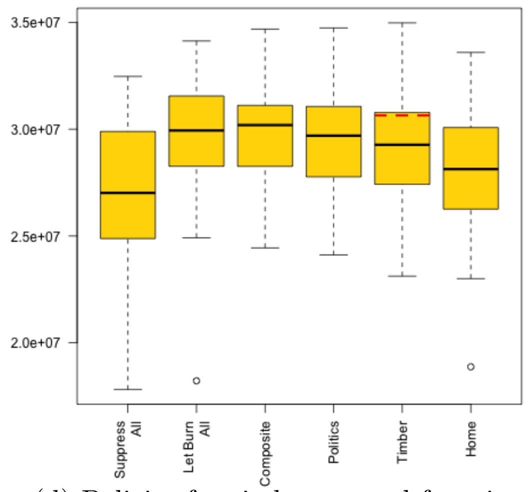
(a) Policies for composite reward function.



(b) Policies for politics reward function.



(c) Policies for home owner reward function.



(d) Policies for timber reward function.

Figure 4.3: Each set of box charts show the performance of various policies for a constituency. The individual box plots show the expected discounted reward for each of the policies optimized for a constituency, as well as the hard-coded policies of suppress-all and let-burn-all. The red dashed lines indicate the expected discounted reward estimated by MFMCi.

optimal for each reward function. The only clear case is for the *home owner* reward function where the SMAC-optimized policy is clearly superior to all of the other policies.

4.5 Discussion

Previous work on high-fidelity wildfire management [27] has focused only on *policy evaluation*, in which the full simulator was applied to evaluate a handful of alternative policies. This chapter reports the first successful *policy optimization* for wildfire suppression at scale. This chapter demonstrates that SMAC applied to our MFMC-based surrogate model is able to find high-quality policies for four different reward functions and do so at interactive speeds. This combination of optimization efficiency and robust ease of use has the potential to provide a basis for interactive decision support that can help diverse groups of stakeholders explore the policy ramifications of different reward functions and perhaps reach consensus on wildfire management policies.

5 Conclusion

The three chapters of this thesis explore three questions surrounding the application of machine learning methods to public policy problems. First, we developed a visual analytics environment for testing MDPs. Such testing methods are required if the field of machine learning is to be trusted for public policy decision making. Our evaluation of the visualization on a simplified version of our wildfire simulator illustrated this point when we found a large number of bugs showing the simulator lacked ecological validity.

Having shown the utility of the visualization for a simplified simulator, we then turned to dealing with the full-fidelity simulator. However, the computational expense of the simulator prevented adequate exploration of the policy parameter space. Consequently, we developed an extension to Model-Free Monte Carlo that supports the rapid visualization of state variables for a changing policy function. The resulting algorithm is the first successful application of Model-Free Monte Carlo to a large state space MDP.

In our final manuscript, we provided an optimization method that is both fast enough to run interactively and capable of handling a variety of reward and policy function spaces. Collectively, the contributions of fast simulation, optimization, and visualization allow domain experts and researchers to detect bugs and explore why the optimization system makes a particular recommendation. Further, we note that our motivating domain of wildfire suppression policy has heretofore never successfully been optimized over 100 year time horizons with a high fidelity simulator.

6 Future Work

Machine learning public policy recommendations should be testable for correctness, explainable, and negotiable to reconcile stakeholder preferences. Our completed research focuses on developing and evaluating a visual analytic system for the testing use case. The immediate next step is to formally address the use cases of explainability and negotiability. Future work in these areas potentially includes changes to the visual interface, additional algorithms, and a formal evaluation with stakeholders.

Supporting the negotiation use case involves exploring the decisions selected by policies optimized for different constituencies. In the current visualization design, it is the user's responsibility to select reward parameters that may be informative. A potential algorithmic improvement for the visualization would provide a model predicting the policy response surface for changes to the reward function. With a model predicting the optimized policy, it is possible to more quickly identify where stakeholders may disagree in the behavior of the policy, and seek out instances where a composite reward function may produce a policy that is mutually agreeable.

We also identify research opportunities building on the findings of our surrogate wildfire simulator. The surprisingly strong performance of the biased version of MFMCi suggests that a distance metric that is aware of trajectory sampling biases may substantially improve performance. Current distance metrics can repeatedly incur small errors even when a more carefully stitched state could reduce the accumulation of error by finding a policy in the database that largely agrees with the target policy.

Bibliography

[1] Pieter Abbeel, Varun Ganapathi, and Andrew Y. Ng. Learning Vehicular Dynamics, with Application to Modeling Helicopters. *Advances in Neural Information Processing Systems (NIPS)*, pages 1–8, 2005.

[2] Pieter Abbeel, Morgan Quigley, and Andrew Y. Ng. Using inaccurate models in reinforcement learning. *International Conference on Machine Learning*, pages 1–8, 2006.

[3] Shehzad Afzal, Ross Maciejewski, and Davis S. Ebert. Visual Analytics Decision Support Environment for Epidemic Modeling and Response Evaluation. In *Visual Analytics Science and Technology (VAST), 2011 IEEE Conference on*, pages 191–200. IEEE, 2011.

[4] Bachisio Arca, Tiziano Ghisu, William Spataro, and Giuseppe a. Trunfio. GPU-accelerated Optimization of Fuel Treatments for Mitigating Wildfire Hazard. *Procedia Computer Science*, 18:966–975, 2013.

[5] Jonathan Baxter, Peter L Bartlett, and Lex Weaver. Experiments with Infinite-Horizon, Policy-Gradient Estimation. *Journal of Artificial Intelligence Research*, 15:351–381, 2001.

[6] Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The Arcade Learning Environment: An Evaluation Platform for General Agents. 47(IJCAI):253–279, 2012.

[7] Richard Bellman. *Dynamic Programming.* Princeton University Press, New Jersey, 1957.

[8] Dimitri P Bertsekas, Dimitri P Bertsekas, Dimitri P Bertsekas, and Dimitri P Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena Scientific Belmont, MA, 1995.

[9] Maryam Booshehrian, Torsten Möller, Randall M. Peterman, and Tamara Munzner. Vismon: Facilitating Analysis of Trade-Offs, Uncertainty, and Sensitivity In Fisheries Management Decision Making. In *Proceedings of Euro-*

graphics Conference on Visualization 2012 (EuroVis 2012), pages 1235–1244. Computer Graphics Forum, 2012.

[10] Matthew Brehmer and Tamara Munzner. A multi-level typology of abstract visualization tasks. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2376–2385, 2013.

[11] Leo Breiman. Random Forests. *Machine learning*, 45(1):5–32, 2001.

[12] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym, 2016.

[13] Bertjan Broeksema, Thomas Baudel, Alex Telea, and Paolo Crisafulli. Decision Exploration Lab: A Visual Analytics Solution for Decision Management. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):1972–1981, 2013.

[14] Mark Cutler, Thomas J. Walsh, and Jonathan P. How. Reinforcement learning with multi-fidelity simulators. *2014 IEEE International Conference on Robotics and Automation (ICRA)*, (1):3888–3895, 2014.

[15] Marc Peter Deisenroth. A Survey on Policy Search for Robotics. *Foundations and Trends in Robotics*, 2(2011):1–142, 2011.

[16] TG Dietterich, MA Taleghan, and Mark Crowley. PAC Optimal Planning for Invasive Species Management: Improved Exploration for Reinforcement Learning from Simulator-Defined MDPs. *Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

[17] G. Dixon. *Essential FVS : A User's Guide to the Forest Vegetation Simulator*. USDA Forest Service, Fort Collins, CO, 2002.

[18] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking Deep Reinforcement Learning for Continuous Control. In *International Conference on Machine Learning (ICML-16)*, volume 48, page 14, 2016.

[19] Damien Ernst, Guy-Bart Stan, Jorge Goncalves, and Louis Wehenkel. Clinical data based optimal STI strategies for HIV: a reinforcement learning approach. *Proceedings of the 45th IEEE Conference on Decision and Control*, pages 667–672, 2006.

[20] Mark Finney, Isaac C Grenfell, and Charles W McHugh. Modeling containment of large wildfires using generalized linear mixed-model analysis. *Forest Science*, 55(3):249–255, 2009.

[21] Mark A. Finney. *FARSITE: fire area simulator model development and evaluation.* USDA Forest Service, Rocky Mountain Research Station, Missoula, MT, 1998.

[22] Raphael Fonteneau, Susan A Murphy, Louis Wehenkel, and Damien Ernst. Model-Free Monte Carlo-like Policy Evaluation. *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, pages 217–224, 2010.

[23] Raphael Fonteneau, Susan a Murphy, Louis Wehenkel, and Damien Ernst. Batch Mode Reinforcement Learning based on the Synthesis of Artificial Trajectories. *Annals of Operations Research*, 208(1):383–416, Sep 2013.

[24] Raphael Fonteneau and L A Prashanth. Simultaneous Perturbation Algorithms for Batch Off-Policy Search. In *53rd IEEE Conference on Conference on Decision and Control*, 2014.

[25] Alborz Geramifard, Robert H Klein, Christoph Dann, William Dabney, and Jonathan P How. RLPy: The Reinforcement Learning Library for Education and Research. http://acl.mit.edu/RLPy, 2013.

[26] Alex Groce, Todd Kulesza, Chaoqiang Zhang, Shalini Shamasunder, Margaret Burnett, Weng-Keen Wong, Simone Stumpf, Shubhomoy Das, Amber Shinsel, Forrest Bice, and Kevin McIntosh. You Are the Only Possible Oracle: Effective Test Selection for End Users of Interactive Machine Learning Systems. *IEEE Transactions on Software Engineering*, 40(3):307–323, 2014.

[27] Rachel M. Houtman, Claire A. Montgomery, Aaron R. Gagnon, David E. Calkin, Thomas G. Dietterich, Sean McGregor, and Mark Crowley. Allowing a Wildfire to Burn: Estimating the Effect on Future Fire Suppression Costs. *International Journal of Wildland Fire*, 22(7):871–882, 2013.

[28] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential Model-Based Optimization for General Algorithm Configuration (extended version). *University of British Columbia, Department of Computer Science*, pages 507–523, 2010.

[29] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient Global Optimization of Expensive Black-Box Functions. *Journal of Global Optimization*, 13:455–492, 1998.

[30] Daniel Keim, Gennady Andrienko, Jean-daniel Fekete, G Carsten, Guy Melan, Daniel Keim, Gennady Andrienko, Jean-daniel Fekete, and G Carsten. Visual Analytics: Definition, Process, and Challenges. *Information Visualization - Human-Centered Issues and Perspectives*, pages 154–175, 2008.

[31] Todd Kulesza, Margaret Burnett, Weng-keen Wong, and Simone Stumpf. Principles of Explanatory Debugging to Personalize Interactive Machine Learning. In *ACM Conference on Intelligent User Interfaces*, 2015.

[32] H. J. Kushner. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Basic Engineering*, 86:97–106, 1964.

[33] Sean McGregor. (Pull Request) fixed self.noise capitalization, 2015.

[34] Sean McGregor. Flask Server. https://github.com/MDPvis/gym/blob/feature-mdpvis-updated/flask$_server.py$, 2016.

[35] Sean McGregor. Flask Server for RLPy. https://github.com/MDPvis/rlpy/blob/mdpvisV2Flask/flask$_server.py$, 2016.

[36] Sean McGregor. IBM Watson AI XPRIZE Competitors, 2017.

[37] Sean McGregor, Hailey Buckingham, Thomas G. Dietterich, Rachel Houtman, Claire Montgomery, and Ron Metoyer. Facilitating Testing and Debugging of Markov Decision Processes with Interactive Visualization. In *IEEE Symposium on Visual Languages and Human-Centric Computing*, Atlanta, 2015.

[38] Sean McGregor, Hailey Buckingham, Thomas G. Dietterich, Rachel Houtman, Claire Montgomery, and Ronald Metoyer. Interactive visualization for testing Markov Decision Processes: MDPVIS. *Journal of Visual Languages and Computing*, 2016.

[39] Sean McGregor, Rachel Houtman, Claire Montgomery, Ronald Metoyer, and Thomas G. Dietterich. Factoring Exogenous State for Model-Free Monte Carlo. 2017.

[40] Sean McGregor, Rachel Houtman, Claire Montgomery, Ronald Metoyer, and Thomas G. Dietterich. Fast Optimization of Wildfire Suppression Policies with SMAC. 2017.

[41] Sean McGregor, Rachel Houtman, Claire Montgomery, Ronald Metoyer, and Thomas G. Dietterich. Visualizing High-Dimensional MDPs with Model-Free Monte Carlo. In *The Multi-disciplinary Conference on Reinforcement Learning and Decision Making (RLDM)*, Ann Arbor, 2017.

[42] Malgorzata Migut and Marcel Worring. Visual Exploration of Classification Models for Risk Assessment. In *Visual Analytics Science and Technology (VAST), 2010 IEEE Symposium on*, pages 11–18, 2010.

[43] J. Mockus. Application of bayesian approach to numerical methods of global and stochastic optimization. *Journal of Global Optimization*, 4:347–365, 1994.

[44] Tamara Munzner. A Nested Model for Visualization Design and Validation.

[45] Andrew Y. Ng. *Shaping and policy search in reinforcement learning.* Doctor of philosophy, University of California, Berkeley, 2003.

[46] Andrew Y. Ng and M Jordan. PEGASUS : A policy search method for large MDPs and POMDPs. *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, 2000.

[47] Kristin Potter, Andrew Wilson, Peer Timo Bremer, Dean Williams, Charles Doutriaux, Valerio Pascucci, and Chris R. Johnson. Ensemble-vis: A framework for the statistical visualization of ensemble data. *ICDM Workshops 2009 - IEEE International Conference on Data Mining*, pages 233–240, 2009.

[48] Martin Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* Wiley-Interscience, 1st edition, 1994.

[49] Hrvoje Ribicic, Jurgen Waser, Raphael Fuchs, Gunter Bloschl, and Eduard Groller. Visual Analysis and Steering of Flooding Simulations. *IEEE Transactions on Visualization and Computer Graphics*, 19(6):1062–1075, 2013.

[50] Benjamin Schindler, Hrvoje Ribicic, Raphael Fuchs, and Ronald Peikert. Multiverse data-flow control. In *IEEE Transactions on Visualization and Computer Graphics*, volume 19, pages 1005–1019, 2013.

[51] John Schulman, Sergey Levine, Michael Jordan, and Pieter Abbeel. Trust Region Policy Optimization. In *International Conference on Machine Learning*, 2015.

[52] M. Sedlmair, C. Heinzl, S. Bruckner, H. Piringer, and T. Möller. Visual parameter space analysis: A conceptual framework. *IEEE Transactions on Visualization and Computer Graphics*, 20(12), 2014.

[53] Niranjan Srinivas, Andreas Krause, Sham M. Kakade, and Matthias Seeger. Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design. *Proceedings of the 27th International Conference on Machine Learning (ICML 2010)*, pages 1015–1022, 2010.

[54] Neville A. Stanton, Paul M. Salmon, Laura A. Rafferty, Guy H. Walker, Chris Baber, and Daniel P. Jenkins. *Human Factors Methods: A Practical Guide for Engineering And Design.* Ashgate Publishing Company, Burlington, VT, second edition, 2013.

[55] R S Sutton, D Mcallester, S Singh, and Y Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in Neural Information Processing Systems*, 12:1057–1063, 2000.

[56] Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 216–225, San Francisco, CA, 1990. Morgan Kaufmann.

[57] Ziyu Wang, Frank Hutter, Masrour Zoghi, David Matheson, and Nando de Freitas. Bayesian optimization in high dimensions via random embeddings. *Journal of Artificial Intelligence Research*, 55:361–387, 2016.

[58] Jürgen Waser, Raphael Fuchs, Hrvoje Ribicic, Benjamin Schindler, Gunther Bloschl, and M. Eduard Groller. World Lines. *IEEE transactions on visualization and computer graphics*, 16(6):1458–1467, 2010.

[59] Jürgen Waser, A Konev, B Sadransky, Z Horváth, Hrvoje Ribicic, R. Carnecky, P. Kluding, and B. Schindler. Many Plans: Multidimensional Ensembles for Visual Decision Support in Flood Management. *Eurographic Conference on Visualization (EuroVis)*, 33(3), 2014.

[60] Jürgen Waser, Hrvoje Ribičić, Raphael Fuchs, Christian Hirsch, Benjamin Schindler, Günther Blöschl, and M Eduard Gröller. Nodes on ropes: a comprehensive data and control flow for steering ensemble simulations. *IEEE transactions on visualization and computer graphics*, 17(12):1872–81, dec 2011.

[61] Stefan Wender and Ian Watson. Applying reinforcement learning to small scale combat in the real-time strategy game StarCraft:Broodwar. *2012 IEEE Conference on Computational Intelligence and Games, CIG 2012*, pages 402–408, 2012.

[62] Western Regional Climate Center. *Remote Automated Weather Stations (RAWS)*. Western Regional Climate Center, Reno, NV, 2011.

[63] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

[64] Aaron Wilson, Alan Fern, and P Tadepalli. Using Trajectory Data to Improve Bayesian Optimization for Reinforcement Learning. *Journal of Machine Learning Research*, 15:253–282, 2014.

[65] Tom Zahavy, Nir Ben Zrihem, and Shie Mannor. Graying the black box: Understanding DQNs. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48, 2016.

[66] Andreas Zeller. *Why programs fail: a guide to systematic debugging*. Elsevier, 2009.

[67] Chao Zhang, Arunesh Sinha, and Milind Tambe. Keeping Pace with Criminals: Designing Patrol Allocation Against Adaptive Opportunistic Criminals. *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015)*, (1):1351–1359, 2015.

[68] A. Zilinskas. A review of statistical models for global optimization. *Journal of Global Optimization*, 2:145–153, 1992.

APPENDIX

A Appendix

A.1 Parameter Space Analysis Testing Questions

Testing questions for each of the Parameter Space Analysis questions of [52], along with the interaction in MDPvis pursuing the question.

ID	Question	Interaction
1	Is the reward function giving the expected rewards?	View the discounted or undiscounted rewards as a fan chart and filter down the rollouts or regenerate rollouts with fixed policies.
2	Do the rewards reflect stakeholder preferences?	Filter to individual rollouts and examine its rewards.
3	Do simulated transitions result in realistic states?	Examine individual rollouts.
4	Do simulated transitions result in realistic state distributions?	View the histograms of state variables at the horizon.
5	Does the historical policy produce the historical results?	Add a variable for each variable with historical data that gives the variable's percentile. Display this derived variable in a fan chart.

Table A.1: **Fitting MDP Testing Questions.**

ID	Question	Interaction
6	Are rollouts that complete different from rollouts that break?	Load the failing and completing rollouts as a comparison.
7	Does the policy inappropriately exploit modeling choices?	Detecting this unforeseen problem requires exploration.
8	Are the most extreme state transitions realistic?	Filter to the most extreme transitions.

Table A.2: **Outlier MDP Testing Questions.**

ID	Question	Interaction
9	What is the state of the world when the transition function breaks?	Select only the rollouts that don't complete and explore them.
10	Does one policy have a higher risk of catastrophic outcome despite having a better expected value?	Compare the rollouts from both.
11	Are action selections meaningful?	Filter the histograms to a single state and see what action is selected.
12	Does an optimized policy realistically outperform a hand-coded policy?	Compare the rollouts from both.
13	Are state transitions realistic?	View state detail.
14	Do policies differ in their resultant state distributions as expected?	Generate two sets of rollouts under different policies and compare.

Table A.3: **Partition MDP Testing Questions.**

ID	Question	Interaction
15	Can the user do better than the optimized policy by changing the parameters?	Change the parameters of the policy function and generate new Monte Carlo rollouts.
16	Is the policy converged to a local optimum?	Ask it to optimize from the current position.
17	Is the policy converged to an acceptable local optimum?	Change the starting policy to a completely different policy and re-optimize.
18	Is the optimization algorithm making efficient use of computation?	Add variables to the output describing the learning process.

Table A.4: **Optimization MDP Testing Questions.**

ID	Question	Interaction
19	What is the distribution of states at a particular horizon?	View the fan charts.
20	How certain is the policy function about a specific choice?	View the shifting distribution of action selection while filtering the state variables.

Table A.5: **Uncertainty MDP Testing Questions.**

ID	Question	Interaction
21	Do small changes in the parameters produce vastly different outcomes	Change parameters then compare the two rollout sets.
22	Do small changes in the parameters produce different policies?	Change the parameters and reoptimize.
23	Do different policies earn reward through maximizing different components of the reward function?	Compare rollout sets.
24	Does the policy respond properly to changes in the reward function?	Change the reward parameters and reoptimize.
25	What are the differences in outcomes produced by different policies?	Load the two sets of rollouts as a comparison.
26	What are the most important drivers of policy?	Filter variables in the histogram and watch how the proportion of selected actions update.

Table A.6: **Sensitivity MDP Testing Questions.**