Facilitating Testing and Debugging of Markov Decision Processes with Interactive Visualization

Sean McGregor,[1] Hailey Buckingham,[2] Thomas G. Dietterich,[1]
Rachel Houtman,[2] Claire Montgomery,[2] and Ronald Metoyer[1]
School of Electrical Engineering and Computer Science[1]
Department of Forest Engineering[2]
Oregon State University
Corvallis, Oregon 97331–5501

***Abstract*—Researchers in AI and Operations Research employ the framework of Markov Decision Processes (MDPs) to formalize problems of sequential decision making under uncertainty. A common approach is to implement a simulator of the stochastic dynamics of the MDP and a Monte Carlo optimization algorithm that invokes this simulator to solve the MDP. The resulting software system is often realized by integrating several systems and functions that are collectively subject to failures of specification, implementation, integration, and optimization. We present these failures as queries for a computational steering visual analytic system (MDPVIS). MDPVIS addresses three visualization research gaps. First, the data acquisition gap is addressed through a general simulator-visualization interface. Second, the data analysis gap is addressed through a generalized MDP information visualization. Finally, the cognition gap is addressed by exposing model components to the user. MDPVIS generalizes a visualization for wildfire management. We use that problem to illustrate MDPVIS.**

I. INTRODUCTION

Many challenging optimization problems in sustainability [13], [11], game AI [33], and autonomous control [20] require considering the long-term impacts of actions whose outcomes are stochastic. For example, forest managers must decide whether to suppress a wildfire whose results may prevent wildfires from spreading over subsequent decades [13]. A *policy* that accounts for these temporal and stochastic effects is produced by an optimization system integrating several components that are subject to failures of specification, implementation, integration, and optimization. Since the system stochastically expresses and hides these failures, Testing and debugging these failures (referred to as "bugs") requires exploration. Visual analytics combined with computational steering is well suited to this exploratory task. We introduce a generalized visualization (see Figure 1), MDPVIS, to support this task.

To address a broader class of optimization problems, we target the common optimization formulation of a 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 system state and the chosen action. An MDP is solved by learning a decision making rule (policy) that maximizes the long-term sum of rewards.

Some MDPs are small enough to solve with exact algorithms such as Policy Iteration and Value Iteration [5], but most MDPs of practical interest require Monte Carlo methods. In these cases, the standard approach is to implement a software simulation of the MDP and then apply a Monte Carlo optimizer, like policy gradient search [29], [10] to find a near-optimal policy.

Andrew Ng relays an example [19] of a soccer playing agent whose MDP optimizer learns a policy that maximizes expected reward by exploiting a bug. The soccer agent received a reward for touching the ball under the theory that possession time is associated with scoring goals. 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 (a more difficult to discover policy may actually be optimal).

The multitude of possible MDP bugs and fixes give rise to a highly iterative development process. During our design process for MDPVIS we conducted a series of semi-structured interviews [28] 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" [27] 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 development 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.

MDPVIS 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 MDP programmers, decision makers, and stakeholders need to perform. Third, it provides an interface by which the users can interact with the MDP simulator and optimizer to test and compare different parameters and explore their effects on the resulting behavior of the system.

These three contributions relate to the three challenges

identified by Sedlmair et al. [27]: 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).

We adapt computational steering from the high performance scientific visualization community [22] to achieve our three primary contributions. Whereas computational steering traditionally refers to modifying a computer process during its execution [17], we treat optimization as an open-ended process whose parameters are repeatedly changed for testing, debugging, and building system comprehension.

Software testing is a subfield of software engineering that includes more precise definitions of testing, bugs, and debugging. In this paper, we take a high-level view of testing and debugging. In particular, we define testing as the "...execution of a program with the intent to produce some problems - especially a failure." These failures are generally called "bugs," whose debugging is defined as "Relating a failure or an infection to a defect (via an infection chain) and subsequent fixing of the defect" [34].

In other words, developers test software for bugs by comparing the results of execution to the expected results. Debugging is the exploratory process carried out in order to attribute a bug to the incorrect program code (and ultimately fix it). It is clear, from this definition, that testing and debugging requires that the developer have the ability to 1) clearly associate program output with program input in order to create test cases by executing the code under specific input conditions and 2) compare actual outputs to expected outputs. Our visualization tool supports these two tasks within the context of MDPs.

MDPVIS' target users are researchers interested in steering the optimization itself, simulator developers who are interested in ensuring the policies optimized for the problem domain are well founded, or domain policy experts primarily interested in the outcomes produced by the optimized policy. In real-world settings these roles can be filled by a single person, or each role could be performed by a large team of developers and domain experts. As a secondary contribution, this paper lists testing questions for MDPs from these target users that inform the design of MDPVIS.

Our design process for MDPVIS followed Munzner's nested model [18], which includes steps for characterizing the problem domain and designing visual encodings. In the following sections we formally introduce MDPs with their Data and Task Abstraction, review the optimization visualization literature in section III, give our visual encoding for MDPs in section IV, and present the visualization's prototype for the Wildfire Suppression domain in section V.

II. 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. We formally state the MDP as the standard infinite horizon discounted Markov Decision Process (MDP) with a designated start state distribution [4], [24] $\mathscr{M} = \langle S, A, P, R, \gamma, P_0 \rangle$. 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 discounted expected value of the MDP. For convenience we also define $P_{n|\pi}$ to be the distribution of states at time n when following policy π.

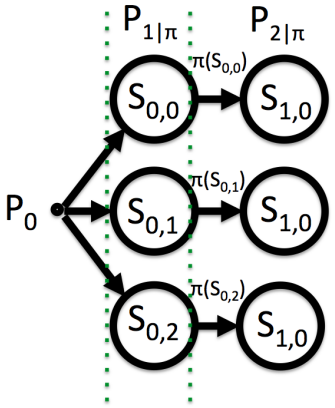

Fig. 2. A set of three rollouts generated starting at states drawn from the initial state distribution (P_0) and transitioned according to the current policy ($\pi(s)$) until a rollout depth of 2. The initial state and all subsequent states are defined on a set of quantitative and categorical variables. 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 annotated the set of rollouts above with two dotted green lines to highlight a set of states drawn from the distribution of states at a particular time horizon under policy π.

This formulation specifies a model that generates data in the form of "Monte Carlo rollouts" detailed in figure 2. These rollouts are the output of the system under test, but since the distribution of these rollouts is defined by applying a policy in many successive states, the rollouts are tightly coupled with the parameter space of the MDP's component functions.

Sedlmair et al. [27] 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 and debugging processes. Finding MDP bugs requires exploring the rollouts to test for bugs. Similarly, establishing bug causality (debugging) requires varying the model parameters and examining the resulting rollouts.

Table I gives a series of testing questions derived from experience optimizing for a wildfire suppression policy domain and from interviewing MDP algorithm researchers not involved in the wildfire policy project. The table labels these questions according to Sedlmair et al.'s [27] tasks for visual parameter space analysis: *optimization, partitioning, fitting, outliers, uncertainty, and sensitivity*. We use these tasks to highlight a few potential bugs below.

Fitting: In many applications the MDP simulator is meant

to simulate real world phenomena to optimize a policy. 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 testing questions 1 through 5 of Table I.

Outliers: In Markov Decision Processes, rewards and penalties are given by a function whose inputs are the current state and the selected action. Even carefully crafted reward functions can result in the algorithm exploiting unforeseen interactions between the MDP's constituent parts. The capacity for a policy to earn rewards from unlikely events mean outliers influence the resulting policy even when they are rare events. See testing questions 10 and 11 from Table I.

Partition: For the partitioning task we expand Sedlmair et al.'s definition which focuses on segmenting the input parameters into groups of similar outcomes. For testing MDPs, we are also interested in discovering which parameters produce different distributions of outcomes when they should be the same and conversely, which sets of parameters produce the same distribution of outcomes when they should be different. When these expectations are violated the MDP likely has a bug. See testing questions 12 through 16.

Optimization: Virtually all MDP optimization algorithms work by iteratively improving the policy until convergence. Convergence occurs when the optimization algorithm decides it can't improve the current policy. The converged policy may be globally optimal, but large state and action spaces often mean the optimization algorithm can only guarantee the policy is a local optimum. For local search optimization algorithms, visualizing several local optima gives perspective on the tradeoffs the optimized policies are implicitly making to maximize elements of the reward function. For example, a policy gradient optimization algorithm [29], [3], [21] for the wildfire domain may find a policy minimizing fire suppression costs if the initial policy suppresses all fires, whereas it may find a policy maximizing ecological value if the initial policy allows all wildfires to burn. In such cases, the rewards, policies, and optimization algorithm are tightly coupled. To test whether an optimized policy has found an acceptable optimum, the MDP practitioner can apply updates to the policy and re-optimize. See testing questions 6 through 9.

Uncertainty: Optimization algorithms produce policies that select actions regardless of how certain they are of the best possible action. States where the optimization algorithm is most uncertain about which action to select require testing for inadequate state representation or insufficient training examples. See testing questions 23 and 24.

Sensitivity: Changing the parameters of the reward function allows for exploring 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 testing questions 17 through 22.

III. Related Work

Many problems have been formulated as Markov Decision Processes, including domains as diverse as RC car control [1], [9], invasive species management [11], and real time strategy games [33]. While no general large MDP visualization 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. [8] 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 debugging classifiers is an active area of research, but published research does not address debugging classifiers for MDPs. Groce et al. [12] 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. [14] includes a classifier debugging system for email messages. The user provides model feedback and correction through an interactive bar chart for a naive Bayes model. Kulesza et al. selected naive Bayes for its interpretability since many classifiers are too complex for end users to understand.

Migut and Worring [16] 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 [23]. 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.

A noteworthy visualization for natural resource management, Vismon [6], gives an interface for exploring tradeoffs in a set of management choices for an Alaskan fishery. Here the fishery manager filters 121 different management choices derived from varying two management parameters. The manager cannot view any management options that are not pre-computed before selecting a policy.

Simulation steering is one branch of visualization that attempts to bring the user into a optimization process by

ID	Question	Task	Interaction	R	P	π	M
1	Is the reward function giving the expected rewards?	fitting	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 the stakeholder's preferences?	fitting	Look at the parameters of the reward function.	●			
3	Do simulated transitions result in realistic states?	fitting	Examine individual trajectories.			●	
4	Do simulated transitions result in realistic state distributions?	fitting	View the histograms of state variables at the horizon.			●	
5	Does the historical policy produce the historical results?	fitting	Add a variable for each variable with historical data that gives the variable's percentile. Display this derived variable in a fan chart.			●	
6	Can the user do better than the optimized policy by tweaking the parameters?	optimization	Have the user tweak the parameters of the policy function and generate new Monte Carlo rollouts.		●		●
7	Is the policy converged to a local optimum?	optimization	Ask it to optimize from the current position.		●		●
8	Is the policy converged to an acceptable local optimum?	optimization	Change the starting policy to a completely different policy and re-optimize.		●		●
9	Is the optimization algorithm making efficient use of computation?	optimization	Add variables to the output describing the learning process.				●
10	Are rollouts that complete different from rollouts that break?	outliers	Load the failing and completing rollouts as a comparison.	●	●	●	●
11	Does the policy inappropriately exploit modeling choices?	outliers	Detecting this unforeseen problem requires exploration.	●	●	●	●
12	What is the state of the world when the transition function breaks?	partition	Select only the rollouts that don't complete and explore them.	●	●	●	●
13	Does one policy have a higher risk of catastrophic outcome despite having a better expected value?	partition	Compare the rollouts from both.		●	●	
14	When faced with a specific situation, what does the policy select?	partition	Filter the histograms to a single state and see what action is selected.		●		
15	How does an optimized policy perform when compared to a hand-coded policy?	partition	Compare the rollouts from both.		●		
16	What does a single outcome look like?	partition	Request the full state information.			●	
17	Do small changes in the parameters produce vastly different outcomes	sensitivity	Change parameters then compare the two rollout sets.	●	●	●	●
18	Do small changes in the parameters produce different policies?	sensitivity	Change the parameters and reoptimize.	●	●	●	●
19	Do different policies earn reward through maximizing different components of the reward function?	sensitivity	Compare rollout sets.	●	●	●	
20	Does the policy respond properly to changes in the reward function?	sensitivity	Change the reward parameters and re-optimize.	●			●
21	What are the differences in outcomes produced by different policies?	sensitivity	Load the two sets of rollouts as a comparison.		●	●	
22	What are the most important drivers of policy?	sensitivity	Filter variables in the histogram and watch how the proportion of selected actions update.		●		
23	What is the distribution of states at a particular horizon?	uncertainty	View the fan charts.		●	●	
24	How certain is the policy function about a specific choice?	uncertainty	View the shifting distribution of action selection while filtering/brushing the state variables.		●		

TABLE I. MDP TESTING QUESTIONS WITH THEIR SEDLMAIR ET AL. [27] TASKS, THEIR REALIZATION IN MDPVIS, AND THE COMPONENTS TESTED BY THE QUESTION, WHERE M IS THE OPTIMIZATION FUNCTION THAT PRODUCES POLICIES FOR THE MDP.

allowing the user to select actions at each timestep as the simulator executes. Simulation steering for epidemic response decisions in Afzal et al. [2] show 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 provides user-based optimization for a deterministic MDP.

Waser et al. [30] 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 to World Lines [32], [25], [26], [31] support stochasticity through secondary simulation controls (random levee breach locations) on the probabilistic parameters of the model, but the visualization does not center on machine learned policies.

Simulation steering is distinct from computational steering, which is concerned with changing the parameters, resolution, or representation of a computing process [22]. Computational steering was largely developed by the high performance computing world to steer computation during execution [17], but we appropriate the term for producing policies and rollouts in batches.

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 the formal properties of a class of problems.

IV. VISUAL ENCODING OF MDPS

Our visual encoding for MDPs in Figure 1 is shown in four computational steering 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 rollouts computed by the MDP. Here we explain the visual interface of MDPVIS, followed by implementation details in section V. Each of the following subsections describe parts of the example of Figure 1.

A. Steering Panels

Rewards Specification: $R(s,a)$

Policy optimization is 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 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 rollouts.

Model Definition: P

The MDP's simulator may expose model parameters to tune the system, or eliminate complexity for debugging 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 rollouts.

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

The policy controls give the current policy. If the user chooses to optimize then they can explore the sensitivity of the policy to changes in the model or rewards. Checking this linkage between parameters and policy determines whether the policy is stable for minor changes in the reward function or whether the policies earn reward via different parts of the reward function.

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. While the visualization still answers many testing questions without rendering the policy parameters, we believe the work of Broeksema et al. [8] could be a good stand-in for this area when the user does not need to modify the policy.

When the policy function is updated it is necessary to regenerate the set of Monte Carlo rollouts that are visualized in the areas below.

Exploration History

As the user repeatedly generates sets of rollouts under different parameter settings they may want to return to a prior parameter set to continue exploration from that point or to compare the sets of rollouts experienced under the prior parameter set. Here we add two buttons for every set of rollouts that have been generated. One button will reload the prior set of parameters and the rollouts that they generated into the visualization. The other button will put the visualization into "comparison mode", which displays the difference between two rollout 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 rollouts and the one being applied as a comparator.

More information about the comparison mode is included in the following visualization areas.

B. Visualization Panels

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

Having specified all the computational steering parameters, we can show the distribution of states at a particular timestep as a histogram. The user can select a range of values in the histogram via brushing, which is a visualization technique for selecting a subset of data via an input device. Brushing causes all histograms to update to reflect only those rollouts that satisfy the selected range. This supports a global-to-local [27] testing process where exploration starts with an overview over all rollouts before drilling down into specific rollouts. When the user drills down to specific rollouts the context of their brushing is shown with an unfilled 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 rollouts.

State Variable Evolution: P

An important question when debugging an MDP is "Do the state distributions reflect the real world?" (questions 4 and 5 from Table I). In MDP problems, there can be many variables that evolve over time to produce a distribution of outcomes at timesteps. In this area we represent distributions as fan charts giving the deciles of variables across rollout 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 rollout 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 rollouts is small enough to avoid the visual clutter of many intersecting lines, we render the rollouts as a line chart.

The user explores the conditional state distributions, $P(S|S_n \in F)$ by manipulating filters (F) on either the fan charts or the histograms. When filtering the fan chart the filters in the histogram area also updated, and vice versa. Changing the timestep of the fan chart's filter updates the histograms to the newly selected timestep. By filtering out values the fan charts show a conditional distribution of state variables.

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. Figure 4 shows two fan charts rendered in comparison mode.

When the user filters the rollouts and clicks a particular rollout, MDPVIS requests the state detail.

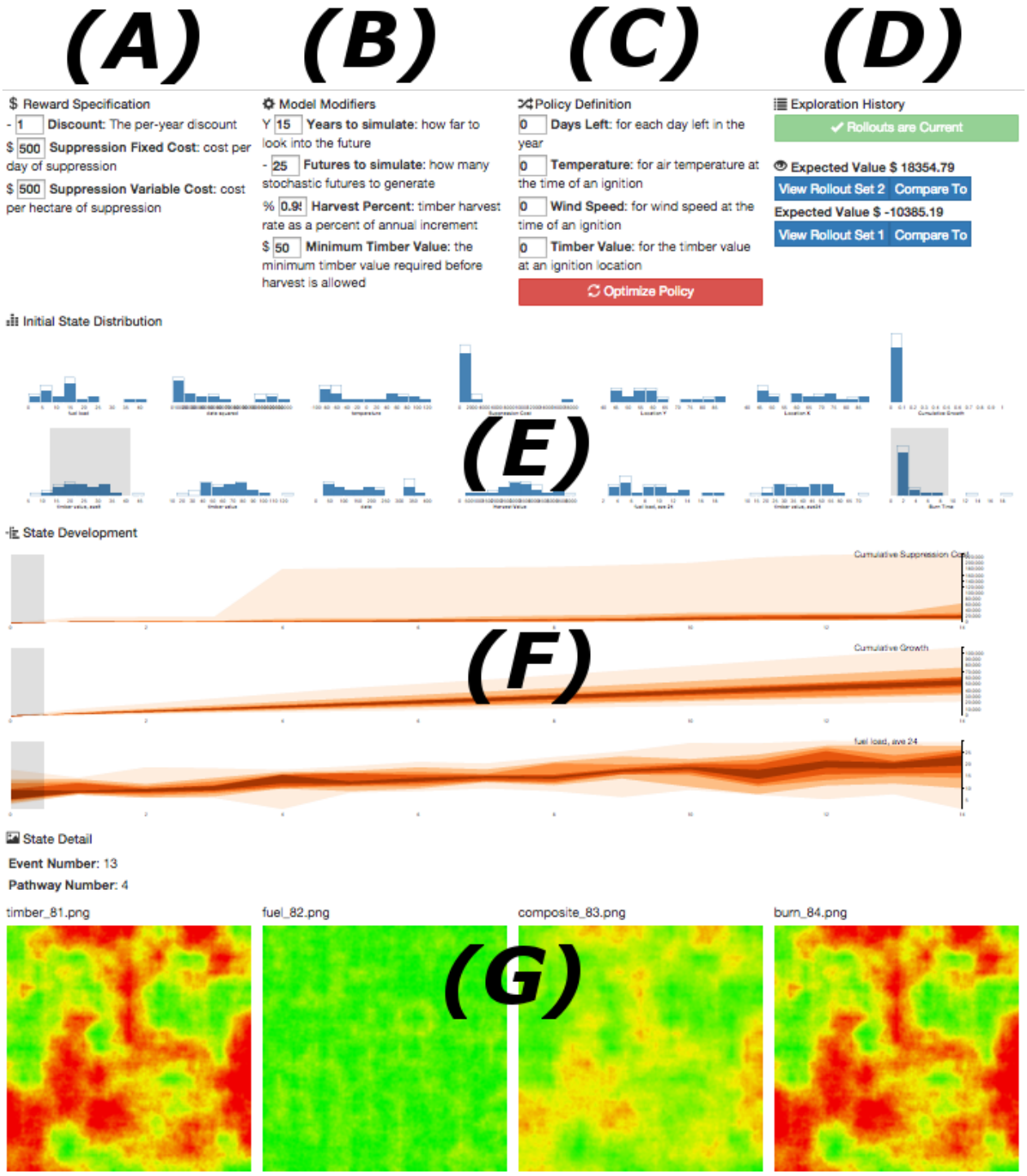

Fig. 1. A high level overview of the Markov Decision Process visualization prototype: MDPVIS. The top row has the three computational steering controls for (A) the reward specification, (B) the model modifiers, and (C) the policy definition. A fourth panel gives the history of Monte Carlo rollout sets 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 rollouts button found under the Exploration History section. The visualization has two buttons in the History panel for each set of Monte Carlo rollouts, one for visualizing the associated Monte Carlo rollouts and another for comparing two sets of rollouts. 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*.

State Detail: $S_{i,j}$

We considered but ultimately rejected rendering the details of individual rollouts with a scatterplot matrix, parallel coordinates, Multi-Dimensional Scaling (MDS), or Multiple Correspondence Analysis (MCA). However, 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 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 1, 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.

V. 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 [7] 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, rollouts, 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 steering 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 rollouts for the initial parameters.
2) ***/rollouts?QUERY*** – Get rollouts 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.
4) ***/state?IDENTIFIER*** – Get the full state details and images. This is required for high dimensional problems in which the entire state cannot be returned to the client for every state in a rollout

This is a minimal set of queries for integrating a visualization with an MDP domain. All relevant languages have web server libraries that can be integrated with the MDP's code base for serializing Monte Carlo rollouts. 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.

VI. USE-CASE STUDY: WILDFIRE SUPPRESSION POLICIES

We arrived at our generalized visualization by following Munzner's nested model [18] for the problem domain of Wildfire Suppression then generalized the results to all MDPs. Since the Wildfire Domain has high-dimensional states, it is representative of a particularly challenging group of MDPs.

We expressed the suppression policy as a smoothly 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 use policy gradient methods [29], [3], [21] 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.

Our wildfire suppression domain combines models for fire spread, vegetative growth, weather, suppression effectiveness, suppression cost, and harvest. One of the most difficult questions any landscape modeler faces is how much realism is necessary to address the hypothesis at hand? More realism means greater development, computer processing, and computer memory requirements, while less realism can lead to results that are nonsensical at best and misleading at worst. The fact that the state space is so large makes it difficult to comprehend how each assumption affects the final outcome. MDPVIS allows us to explore ways in which the simulator is failing, either through hard-to-find technical bugs or "garbage in, garbage out" assumptions that are affecting the way we value different outcomes.

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, who are co-authors on this paper, 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 a PhD student from computer science who implemented MDPVIS but did not contribute to the development of the MDP.

We detected bugs for most of the questions of Table I 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 rollouts 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 3). We could only see this on the larger fires because the rectangular timber harvests were masking the unusual shape of the fire spread.

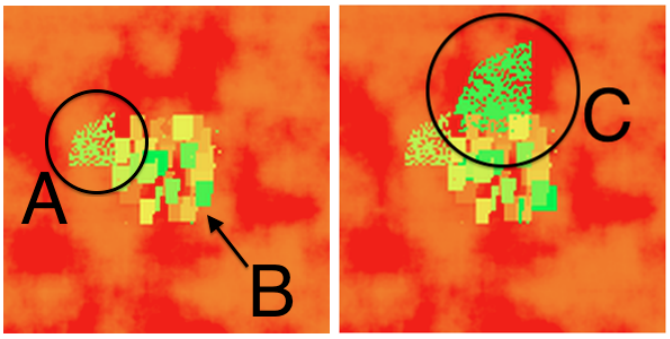

Fig. 3. Two sequential spatial snapshots of timber values for a state transition that includes one of the largest fire loss events from 200 rollouts of 60 years. The management area is visible in the center due to the rectangular timber harvests. These rectangular harvests obscure the irregular boundary 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.

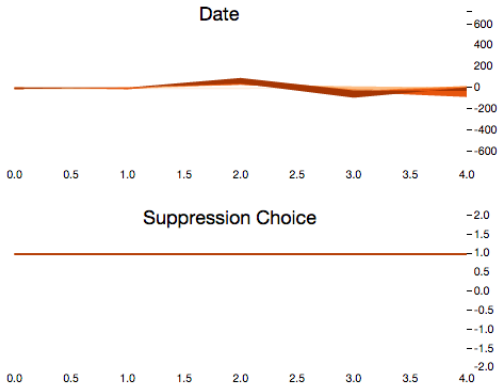

Fig. 4. Fan charts for the suppression choice and the ignition date shown in comparison mode for two rollout sets. We generated one rollout set under a suppress-all policy and a second rollout set under a let-burn-all policy. We confirmed that the proper action is being selected for each state by observing the differences in suppression choices is always 1. However, there is an unexpected difference in the dates, which should be consistent between the two rollout sets.

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

Partition: When comparing two different policies (see Figure 4) 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, 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.

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 tweaking the policy parameters. This shows that the optimizer is failing to find a local optimum.

We were able to identify the most likely source of the problem: when we optimized policies for increasing rollout 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.

Uncertainty: Perhaps the greatest problem with fitting the real world is the lack of uncertainty in the policies we produced. Despite the attention paid to faithfully simulate the real world, policies that always selected letting a wildfire burn outperformed the policies with a more nuanced decision criterion. A critical evaluation of the dynamics that result in a complete reversal of real-world policies is necessary. We plan to continue to use MDPVIS for this evaluation process after addressing the other bugs we uncovered with MDPVIS.

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. This remains an open issue for future research since this could potentially be caused by issues with modeling or our motivating hypothesis that suppression policies can meaningfully affect timber harvests in the long run.

VII. Conclusion and Availability

This paper presented MDPVIS, a domain-independent tool for 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 supports viewing rollout distributions over time and temporal comparisons between policies (either policies produced by the optimizer or policies designed by the user). 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*.

Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant No. 1331932.

References

[1] P. Abbeel, M. Quigley, and A. Y. Ng. Using inaccurate models in reinforcement learning. *International Conference on Machine Learning*, pages 1–8, 2006.

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

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

[4] R. Bellman. *Dynamic Programming*. Princeton University Press, New Jersey, 1957.

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

[6] M. Booshehrian, T. Möller, R. M. Peterman, and T. Munzner. Vismon: Facilitating Analysis of Trade-Offs, Uncertainty, and Sensitivity In Fisheries Management Decision Making. In *Proceedings of Eurographics Conference on Visualization 2012 (EuroVis 2012)*, pages 1235–1244. Computer Graphics Forum, 2012.

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

[8] B. Broeksema, T. Baudel, A. Telea, and P. Crisafulli. Decision Exploration Lab: A Visual Analytics Solution for Decision Management. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):1972–1981, 2013.

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

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

[11] T. Dietterich, M. Taleghan, and M. 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.

[12] A. Groce, T. Kulesza, C. Zhang, S. Shamasunder, M. Burnett, W.-K. Wong, S. Stumpf, S. Das, A. Shinsel, F. Bice, and K. 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.

[13] R. M. Houtman, C. A. Montgomery, A. R. Gagnon, D. E. Calkin, T. G. Dietterich, S. McGregor, and M. Crowley. Allowing a Wildfire to Burn: Estimating the Effect on Future Fire Suppression Costs. *International Journal of Wildland Fire*, 22(7):871–882, 2013.

[14] T. Kulesza, M. Burnett, W. Wong, and S. Stumpf. Principles of Explanatory Debugging to Personalize Interactive Machine Learning. In *ACM Conference on Intelligent User Interfaces*, 2015.

[15] K. Matković, D. Gracanin, M. Jelović, and H. Hauser. Interactive visual steering–rapid visual prototyping of a common rail injection system. *IEEE transactions on visualization and computer graphics*, 14(6):1699–1706, 2008.

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

[17] J. D. Mulder, J. J. van Wijk, and R. van Liere. A survey of computational steering environments. *Future Generation Computer Systems*, 15(1):119–129, 1999.

[18] T. Munzner. A nested model for visualization design and validation. *Visualization and Computer Graphics, IEEE Transactions on*, 15(6):921–928, 2009.

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

[20] A. Y. Ng, A. Coates, M. Diel, and V. Ganapathi. Autonomous inverted helicopter flight via reinforcement learning. *Experimental Robotics IX*, pages 1–10, 2006.

[21] A. Y. Ng and M. Jordan. Pegasus: A policy search method for large mdps and pomdps. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pages 406–415. Morgan Kaufmann Publishers Inc., 2000.

[22] S. G. Parker, D. Beazley, C. R. Johnson, S. L. City, and I.-b. C. Mechanisms. Computational Steering Software Systems and Strategies. 1996.

[23] K. Potter, A. Wilson, P. T. Bremer, D. Williams, C. Doutriaux, V. Pascucci, and C. 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.

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

[25] H. Ribičić, J. Waser, R. Fuchs, G. Bloschl, and E. Groller. Visual Analysis and Steering of Flooding Simulations. *IEEE transactions on visualization and computer graphics*, 19(6):1062–1075, 2013.

[26] B. Schindler, H. Ribičić, R. Fuchs, and R. Peikert. Multiverse data-flow control. In *IEEE Transactions on Visualization and Computer Graphics*, volume 19, pages 1005–1019, 2013.

[27] M. Sedlmair, C. Heinzl, S. Bruckner, H. Piringer, and T. Möller. Visual parameter space analysis: A conceptual framework. *To Appear in IEEE TVCG (Proc. InfoVis)*, 20(12), 2014.

[28] N. A. Stanton, P. M. Salmon, L. A. Rafferty, G. H. Walker, C. Baber, and D. P. Jenkins. *Human Factors Methods: A Practical Guide for Engineering And Design*. Ashgate Publishing Company, Burlington, VT, second edition, 2013.

[29] 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.

[30] J. Waser, R. Fuchs, H. Ribičić, B. Schindler, G. Bloschl, and M. E. Groller. World Lines. *IEEE transactions on visualization and computer graphics*, 16(6):1458–1467, 2010.

[31] J. Waser, A. Konev, B. Sadransky, Z. Horváth, H. Ribičić, 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.

[32] J. Waser, H. Ribičić, R. Fuchs, C. Hirsch, B. Schindler, G. Blöschl, and M. E. 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.

[33] S. Wender and I. 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.

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