Interpretable Local Tree Surrogate Policies

John Mern,[1] **Sidhart Krishnan,** [2] **Anil Yildiz,** [1] **Kyle Hatch,** [2] **Mykel J. Kochenderfer** [1]

[1] Department of Aeronautics and Astronautics, Stanford University
[2] Department of Computer Science, Stanford University

Abstract

High-dimensional policies, such as those represented by neural networks, cannot be reasonably interpreted by humans. This lack of interpretability reduces the trust users have in policy behavior, limiting their use to low-impact tasks such as video games. Unfortunately, many methods rely on neural network representations for effective learning. In this work, we propose a method to build predictable policy trees as surrogates for policies such as neural networks. The policy trees are easily human interpretable and provide quantitative predictions of future behavior. We demonstrate the performance of this approach on several simulated tasks.

Introduction

Deep reinforcement learning has achieved state of the art performance in several challenging task domains (**?**). Much of that performance comes from the use of highly expressive neural networks to represent policies. Humans are generally unable to meaningfully interpret neural network parameters, which has lead to the common view of neural networks as "black-box" functions. Poor interpretability often leads to a lack of trust in neural networks and use of other more transparent, though potentially less high-performing policy models. This is often referred to as the performance-transparency trade-off.

Interpretability is important for high-consequence tasks. Domains in which neural networks have already been applied, such as image classification, often do not require interpretable decisions because the consequences of mislabeling images are typically low. In many potential applications of deep reinforcement learning, such as autonomous vehicle control, erroneous actions may be costly and dangerous. In these cases, greater trust in policy decisions is typically desired before systems are deployed.

Our work is motivated by tasks in which a human interacts directly with the policy, either by approving agent actions before they are taken or by enacting recommendations directly. This is often referred to having a human "in the loop". An example is an automated cyber security incident response system that provides recommendations to a human analyst. In these cases, knowing the extended course of actions before committing to the recommendation can enhance the trust of the human operator.

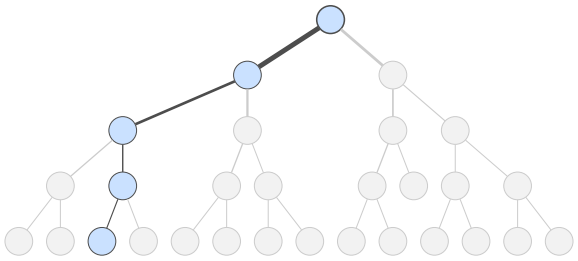


Figure 1: Surrogate Policy Tree. The figure shows a surrogate policy tree for the vaccine planning task, a finite-horizon MDP with 8 discrete actions. Each node represents an action and the states that led to it. Node borders and edge widths are proportional to the probability of encountering that node or edge during policy execution. The most likely trajectory from root to leaf is shown in blue.

It is difficult for humans to holistically interpret models with even a small number of interacting terms (**?**). Neural networks commonly have several thousand parameters and non-linear interactions, making holistic interpretation infeasible. Some existing methods constrain neural networks architectures and train them to learn human interpretable features during task learning. The training and architecture constraints, however, can degrade performance compared to an unconstrained policy. A common approach is to learn transparent *surrogates* from the original models (**?**). A major challenge in this approach is balancing the fidelity of the surrogate to the original with the interpretability of the surrogate. Surrogate models that are generated stochastically can additionally struggle to provide consistent representations for the same policy.

In this work, we propose a method to develop transparent surrogate models as local policy trees. The resulting trees encode an intuitive plan of future actions with high-fidelity to the original policy. The proposed approach allows users to specify tree size constraints and fidelity targets. The method is model-agnostic, meaning that it does not require the original policy to take any specific form. Though this work was

motivated by neural networks, the proposed approach may be used with any baseline policy form.

During execution of a tree policy, the actions taken by an agent are guaranteed to be along one of the unique paths from the root. Experiments on a simple grid world task highlight the impact of algorithm parameters on tree behavior. The experiments also show that using the trees as a receding-horizon policy maintains good performance relative to the baseline model. Additional experiments on more complex infrastructure planning and cyber-physical security domains demonstrate the potential utility of this approach in real-world tasks.

Background

Despite its importance in machine learning, there is no precise qualitative or mathematical definition of interpretability. In the literature, interpretability is commonly used to describe a model with one of two characteristics. The first is explainability, defined by **?** as the degree to which a human can understand the cause of a decision. The second characteristic is predictability, which is defined by **?** as the degree to which a human can consistently predict a models result. In this work, we will refer to models that are either explainable or predictable as interpretable.

Techniques for model interpretability can vary greatly. We characterize the methods presented in this work using the taxonomy proposed by **?** as model-agnostic, local surrogate methods. Model-agnostic methods are those that can be may be applied to any model with the appropriate mapping from input to output. Surrogate modeling techniques distill the behavior of a complex *baseline* model into a more transparent surrogate. Local methods provide interpretability that is valid only in the neighborhood of a target input point. As a result, they tend to maintain higher fidelity to the original model in the acceptable region than models that attempt to provide global interpretations.

Markov Decision Processes

The control tasks in this work are assumed to satisfy the Markov assumption and may be modeled as either Markov decision processes (MDPs) or partially observable Markov decision processes (POMDPs). MDPs are defined by tuples $(\mathcal{S}, \mathcal{A}, T, r, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces, respectively, $T(s' \mid s, a)$ is the transition model, $r(s, a, s')$ is the reward function, and γ is a time discount factor. POMDPs are modeled by the same tuple with the addition of the observation space $\mathcal{O}$ and observation model $Z(o \mid s, a)$. A policy is a function that maps a state to an action in MDPs or a history of observations to an action in POMDPs. To solve a MDP is to learn the policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that maximizes the expected sum of discounted rewards

$$V(s_t) = \mathbb{E}\left[\sum_{\tau=t} \gamma^{\tau-t} r\big(s_\tau, \pi(s_\tau)\big)\right] \tag{1}$$

over all states. Learning an optimal policy is equivalent to learning the optimal action value function

$$Q(s_t, a_t) = \mathbb{E}\left[r(s_t, a_t) + \sum_{\tau=t+1} \gamma^{\tau-t} r\big(s_\tau, \pi^*(s_\tau)\big)\right] \tag{2}$$

where π^* is the optimal policy. When learning an action value function estimator, the effective policy is

$$\pi(s) \leftarrow \arg\max_{a \in \mathcal{A}} \hat{Q}(s, a) \tag{3}$$

where $\hat{Q}(s, a)$ is the learned approximator. These problems can be solved using a variety of methods such as dynamic programming, Monte Carlo planning, and reinforcement learning (**?**). Similarly, various function types can be used to model the policy or value function estimator. Neural networks are commonly used as policies for complex tasks.

Related Work

There has been a wealth of prior work in the field of explainable and interpretable artificial intelligence. The book by **?** provides an overview of model interpretability techniques for general machine learning methods. Methods for specific models and tasks have also been proposed. **?** provides a survey of recent work in explainable reinforcement learning. The discussion in this section is restricted to methods relevant to deep reinforcement learning. Methods requiring domain specific representations (**?**) are not considered.

A common technique for model-agnostic surrogate modeling is to learn a surrogate model of an inherently interpretable class. An useful example is locally interpretable model-agnostic explanations (LIME) (**?**). LIME learns sparse linear representations of a target policy at a specific input by training on a data set of points near the target point. While the resulting linear functions are interpretable, they are often not consistent and small variations in the training data can result in drastically different linear functions.

Several methods have been proposed to distill neural network policies to decision tree surrogates. Linear model U-trees (**?**) and soft decision trees (**?**) take similar approaches to tree representation and learning. Decision trees are global policy representations that map input observations to output actions by traversing a binary tree. Actions can be partially understood by inspecting the values at each internal node. Unfortunately, the understanding provided by decision trees can be limited because the mappings still pass the input observation through several layers of affine transforms and non-linear functions. Further, both methods empirically showed significant performance loss compared to the baseline policies.

Structural causal models (**?**) also try to learn an inherently interpretable surrogate as a directed acyclic graph (DAG). The learned DAG represents relations between objects in the task environment that can be easily understood by humans. The DAG structure, however, must be provided for each task, and the fidelity is not assured.

Our work proposes tree representations that provide intuitive maps over future courses of action. When used as a policy, the realized course of action is guaranteed to be along the branches of the tree. In this way, the method is similar to methods of neural network verification that seek to provide guarantees of network outputs over a given set of inputs. **?** provides an overview of verification for general neural networks. Additional works have been proposed specifically for verification of neural network control policies (**?**).

The methods proposed in this work also resemble Monte Carlo tree search (MCTS) algorithms (**?**). MCTS methods search for an optimal action from a given initial state or belief by sampling trajectories using a search policy. The result of an MCTS search is a tree of trajectories reachable from the initial state. The tree trajectories, however, are not limited to those reached under a fixed policy, but rather by a non-stationary tree search policy. The resulting tree cannot be used to interpret or predict future policy behavior. Trees from planners using UCB exploration (**?**) tend to grow exponentially with search depth h as $O(|\mathcal{S}|^h|\mathcal{A}|^h)$.

Proposed Method

We present model-agnostic methods to represent policies for both MDPs and POMDPs as interpretable tree policies. Trees are inherently more interpretable than high dimensional models like neural networks. The proposed method uses the baseline policy to generate simulations of trajectories reachable from a given initial state. The trajectories are then clustered to develop a width-constrained tree that represents the original policy with good fidelity. The methods assume that baseline policies return distributions over actions or estimates of action values for a given state. Building trees also requires a generative model of the task environment.

In addition to representing the future policy, the tree also provides useful statistics on expected performance such as likelihood of following each represented trajectory. The tree may be used as a policy during task execution with guarantees on expected behavior. We present methods to generalize to states not seen during tree construction by using the tree as a constraint on the baseline policy.

Local Tree Policy

Before describing the tree construction algorithm, we first present the local tree policies that it produces. A policy tree is a rooted polytree where nodes represent actions taken during policy execution. An example tree policy for a stochastic, fully observable task is shown in fig. 1. Each node of the tree represents an action a_t taken at time t. The root action of each tree is the action recommended by the policy at the initial state. Each path from the tree root to a leaf node gives a trajectory of actions $a_t, a_{t+1}, \ldots, a_{t+h}$ that the agent may take during policy execution up to some depth h. Trajectories leading to terminal conditions before h steps result in shallower tree branches.

Policy trees are interpretable representations of the future behavior of a policy that give explicit, quantitative predictions of future trajectories. Each node provides an estimate of the probability that the action sequence up to that node will be taken $P(a_0, \ldots, a_t \mid s_0, \pi)$ and estimates of the policy value $Q(s, a)$. Each node also contains a set of example states or observations that would result in that action.

Figure 2 provides more detailed views of the policy tree in fig. 1. The trajectory probabilities and value estimates shown in fig. 2a are calculated during tree construction and may be presented for any surrogate policy. The states in the example problem $s_t \in \mathbb{R}^n$ are real vectors. Each node in fig. 2b shows the mean of that node's state values. While the mean state value is useful for understanding this problem, it may not be useful in all problems. For example, the mean value may be meaningless for problems with discrete state spaces. Methods to compactly represent a node's set of states or observations cannot be generally defined and instead should be specified for each problem.

Trees are an intuitive choice for the local surrogate model of a control policy. Local surrogate methods seek to provide simple approximate models that are faithful to the original policy in a space around a target point. For control tasks, often only predictions of *future* behavior are useful. Trees provide compact surrogate models by only representing behavior in states that are forward-reachable from the current state. This is in contrast to methods that define local neighborhoods by arbitrarily perturbing the initial point (**?**). In these cases, explanatory capacity is wasted on unnecessary states.

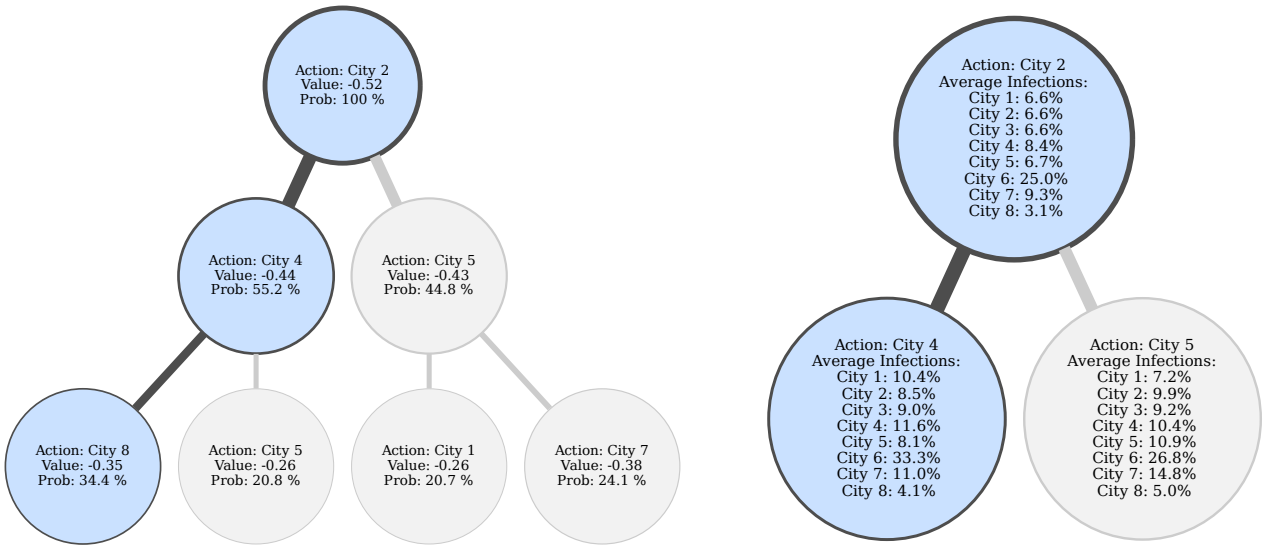
Tree Build Algorithm

Trees are built by simulating multiple executions of the baseline policy from the given initial state or belief and clustering them into action nodes. The process is illustrated in fig. 3. Each simulation is represented by a collection of particles p_t for each time t along the simulation trajectory. Particles are tuples (s_t, r_t) of the state at time t and reward received from $t-1$ to t. For POMDPs, particles also record the observation o_t and belief b_t.

Tree construction begins by first generating a set of particles for the initial state or belief. For the initial step, all particles are assigned to the root action node an_0, which takes the action given by the baseline policy $a_0 = \pi(s_0)$ for MDPs or $a_0 = \pi(b_0)$ for POMDPs. The particles are then advanced through one simulation time step to produce the particles for $t+1$, as shown in fig. 3a. The $t+1$ particles that did not enter terminal states are clustered to new action nodes as shown in fig. 3b. This process continues until a terminal state is encountered or a fixed depth limit is met. Action nodes that do not have at least $n_{\min}$ particles are not expanded further to limit over-fitting, where $n_{\min}$ is specified by the user.

The point of entry for tree construction is the BUILD function, presented in algorithm 1 for MDPs. The initial particle set is created and clustered into the root action node. Each root particle for an MDP policy is initialized with the same state s_0. Each action node is a tuple (P, a, Ch), where P is the node's set of particles, a is the action taken from that node, and Ch is the set of child nodes. The BUILD function for POMDPs follows the same procedure as for MDPs, though states and observations for the initial particle set are sampled from the initial belief b_0.

Particles are advanced through recursive calls to the ROLLOUT function (algorithm 2). Each time ROLLOUT is called from an action node an, it proceeds if the size of the node particle set an_P exceeds the minimum threshold and the depth limit $d_{\max}$ has not been exceeded. If these conditions are met, each particle is advanced by calling the simulator GEN(s, a) using that particle's state and the node action. The set of new particles are then grouped into new action nodes by the CLUSTER function. ROLLOUT is recur-



(a) (b)

Figure 2: Detailed Tree Views. (a) The figure shows the first three layers of the vaccine surrogate policy tree. Each node provides its action, estimated value, and probability of reaching it. (b) The figure shows the first two tree layers with the mean of the observations leading to each action node displayed.

Algorithm 1: Build MDP

procedure BUILDMDP(s_0, π, n, d_{max})
  $P \leftarrow \emptyset$
  $Ch \leftarrow \emptyset$
  $d \leftarrow 0$
  for $i \in 1 : n$
    $P \leftarrow P \cup \{(s_0, 0)\}$
  $p_0 \leftarrow \pi(s_0)$
  $a_0 \leftarrow \arg\max_{a \in \mathcal{A}} \pi(s_0)$
  root $\leftarrow$ Node(P, a_0, p_0, Ch)
  root $\leftarrow$ ROLLOUT(root, π, d)
  return root

Algorithm 2: Rollout

procedure ROLLOUT(an, π, d)
  if $|an_P| \geq n_{min}$ and $d < d_{max}$
    $P \leftarrow \emptyset$
    for $p \in an_P$
      $s \leftarrow p_s$
      $s', o', r',$ done $\leftarrow$ GEN(s, an_a)
      if not done
        $p' \leftarrow$ PARTICLE(s', o', r')
        $P \leftarrow P \cup \{p'\}$
    $C \leftarrow$ CLUSTER(P, π)
    for $c \in C$
      $c' \leftarrow$ ROLLOUT$(c, \pi, d+1)$
      $an_{ch} \leftarrow an_{ch} \cup \{c'\}$
  return node

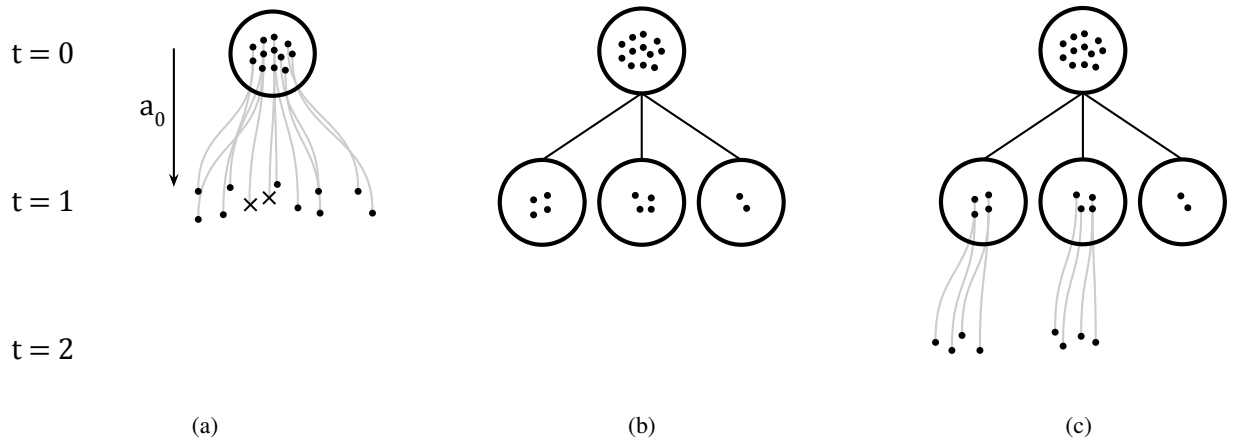
sively called on each new action node and the returned node is added to the child node set an_{Ch}.

The action nodes of the tree T encode a deterministic policy for the sampled particles, $T : \mathcal{S}_R \rightarrow \mathcal{A}$, where $\mathcal{S}_R$ is the set of states contained in the particle set. The particles are clustered into action nodes to minimize how much this policy deviates from the baseline policy π while meeting constraints on tree size. To achieve this, we developed a clustering algorithm that approximately solves for the optimal clustering through recursive greedy optimization.

The recursive clustering approach is presented in algorithm 3. The algorithm progressively clusters particles into an increasing number of nodes k until a distance measure δ between the actions assigned by the tree and the baseline policy is less than a threshold δ^* or the maximum number of nodes $c_{\max}$ is exceeded.

Clusters are assigned according to a greedy heuristic process shown in algorithm 4. In this approach, the complete set of actions assigned to each particle under the baseline policy A^U is ranked according to frequency by the UNIQUEACTIONS function. Action nodes for each of the top k actions and all particles assigned that action by the baseline policy are clustered. Any particles not clustered to a node at the end of this process are assigned to the previously formed node that minimizes the distance between that action and the action assigned by the baseline policy.

The distance metric may be any appropriate measure assigned by the user for the given policy type. For action value function policies, an effective distance measure for an action



(a) (b) (c)

Figure 3: Tree build example. (a) An initial set of particles is sampled from the given initial state or belief. These particles are advanced in the simulation using the action at the root node a_0. Particles reaching terminals states are marked with $\times$. (b) Non-terminal particles are clustered into new action nodes. (c) Action nodes with a sufficient number of particles are advanced another time step. This process will continue for each action node until all particles terminate or reach a maximum depth.

Algorithm 3: Recursive Cluster

procedure CLUSTER(P, π)
  $k \leftarrow 1$
  repeat
    $C, \delta \leftarrow$ GREEDYCLUSTER(P, k, π)
    $k \leftarrow k + 1$
  until $\delta \leq \delta^*$ or $|C| \geq c_{\max}$
  return C

a would be

$$\delta = \|\hat{Q}(s, a) - \max_{a' \in \mathcal{A}} \hat{Q}(s, a')\| \tag{4}$$

which intuitively gives the predicted sub-optimality of the action. For stochastic policies which output distributions over actions, the difference in action probability would be an appropriate distance.

Tree Policy Control

To use the tree as a policy during task execution, it must be able to generalize to states not encountered in the particle set of the tree. To do this, we propose a simple method that uses the baseline policy, constrained by the tree at each time step. For a tree constructed for a state s_0, the policy will always take the action at the root node an_0. For all remaining steps, the policy will return

$$a \leftarrow \arg \max_{a' \in \mathcal{A}^T} \pi(s_t) \tag{5}$$

where $\mathcal{A}^T$ is the set of actions in the child set of the preceding action node an_{t-1}. Intuitively, the agent will take the best action predicted by the baseline policy that is included in the tree. Building a new tree each time a leaf state is encountered allows the tree policies to be run in-the-loop for long or infinite-horizon problems.

Algorithm 4: Greedy Cluster

procedure GREEDYCLUSTER(P, k, π)
  $C \leftarrow \emptyset$
  $\delta \leftarrow 0$
  $A^U \leftarrow$ UNIQUEACTIONS(P, π)
  for $i \in 1 : k$
    $a \leftarrow A^U[i]$
    $P^A \leftarrow \{p \in P \mid \arg\max_{a \in \mathcal{A}} \pi(p_s) = a\}$
    $C \leftarrow \{\text{NODE}(P^A, a, \emptyset)\} \cup C$
    $P \leftarrow P \setminus P^A$
  for $p \in P$
    $an \leftarrow \arg\min_{an' \in C}$ DISTANCE(p, an', π)
    $\delta \leftarrow \delta +$ DISTANCE(p, an, π)
    $an_P \leftarrow \{p\} \cup an_P$
  return C, δ

Experiments

We ran experiments to test the performance of the proposed approach. One set of experiments are conducted on a simple grid world task. These experiments were designed to quantitatively measure the effect of various algorithm parameters and environment features on tree size and fidelity. Two additional experiments were run on more complex tasks that demonstrate the utility of the approach on real-world motivating examples. The first of these is multi-city vaccine deployment planning. In large-scale infrastructure planning tasks such as this, it is often necessary to have a reasonable prediction of all steps of the plan in order to gain stakeholder trust. The second task is a cyber security agent that provides recommendations to a human analyst to secure a network against attack. Interpretable policies are important for tasks with human oversight and cooperation.

We implemented the proposed algorithm with the distance function defined in eq. (4) in Python. Neural network training was done in PyTorch (**?**). Source code, full experiment

descriptions, and results are available in the Appendix.

Grid World

In the grid world task, an agent must navigate a discrete, 2D world to reach a goal state while avoiding trap states. The agent may take one of n total actions to move between 1 and $\lfloor \frac{n}{4} \rfloor$ units in any of the four cardinal directions on the grid. The agent moves in the intended direction with probability p and takes a random action otherwise. To incentivize solving the problem quickly, the agent receives a cost of -1 at each time step. The agent gets a positive reward of $+10$ for reaching a goal state and a penalty of -5 for reaching a trap state. The episode terminates when the agent reaches a goal state or when a maximum number of steps is reached. Because we know exact transition probabilities, we used discrete value iteration to learn a baseline policy (**?**).

We constructed surrogate trees from the baseline policy using various environment and tree-build algorithm settings. For the environment, we swept over different values of the transition probability p and the number of actions n. For the tree build algorithm, we varied the total number of particles, the minimum particle count, the distance threshold δ^*, and the maximum leaf node depth. Only one parameter was varied for each test, with all others held fixed. The baseline settings for the environment were $n = 4$ actions and a successful transition probability of $p = 0.9$. The baseline tree build parameters are 1000 total particles, 250 minimum particles, distance threshold of 0.01, and maximum depth of 10.

To test each tree, we ran it as a policy from the initial state until it reached a leaf node. The baseline policy was then used to complete the episode. We tested 2,500 trees for each parameter configuration. Select results are shown table 1. The mean change in performance of the tree policy relative to the baseline is shown along with one standard error bounds. The average depth of leaf nodes is also shown, though standard error is omitted as each had $SE < 0.05$.

From the transition probability sweep, we see that relative performance generally improves as transition probability increases At $p = 0.5$, both policies perform very poorly and the difference between the two is not significant as a result. In the deterministic case, the surrogate tree perfectly represents the baseline policy. These results suggest that surrogate trees are better suited for tasks with low stochasticity.

As we increase the number of actions, the difference in performance between the baseline and the tree policy decreases. The average leaf depth of the trees also decreases with more actions. This likely explains the improved performance, as shallower trees will transition back to the baseline policy sooner in the tests. Another possible explanation is that as the number of actions grows, the cost of taking a suboptimal action decreases. For example, with 4 actions, if the optimal action is not taken, then the agent moves in a completely different direction than it should. With $n > 4$ actions, the agent may still move in the correct direction, though by more or less distance than optimal.

For the tree building algorithm parameters, we see that as δ^* increases, the depth of the leaf nodes also decreases. This is likely because higher values of δ^* leads to more aggressive node clustering and fewer branches. Similar to the trend observed by varying the number of actions, as the average leaf depth decreases, relative performance increases. Similarly, as the max depth increases, the tree is allowed to grow deeper and the relative performance drops.

Parameter	Value	Rel Change (%)	Leaf Depth
Trans. Prob.	0.5	4.5 ± 4.7	4.4
	0.7	-8.0 ± 4.0	4.4
	0.9	$\mathbf{-4.4 \pm 1.0}$	**4.2**
	1.0	0.0 ± 0.0	4.0
No. Actions	**4**	$\mathbf{-4.4 \pm 1.0}$	**4.2**
	8	-2.6 ± 0.4	2.6
	16	-1.0 ± 0.3	1.5
δ^*	0.005	-5.7 ± 1.0	4.2
	0.01	$\mathbf{-4.4 \pm 1.0}$	**4.2**
	0.1	-3.1 ± 1.0	3.1
Max Depth	3	-1.3 ± 0.9	2.0
	5	-2.9 ± 1.0	3.5
	10	$\mathbf{-4.4 \pm 1.0}$	**4.2**

Table 1: Parameter Search Results. The change in task performance relative to the baseline policy performance and the average depth of leaf nodes in the trees are given. Values generated with the baseline settings are shown in bold.

Vaccine Planning

In the vaccine deployment task, an agent decides the order in which to start vaccine distributions in cities in the midst of a pandemic outbreak. Each step, the agent picks a city in which to start a vaccine program. The spread of the disease is modeled as a stochastic SIRD model (**?**), with portions of each city population being susceptible to infection (S), infected (I), recovered (R), or dead (D). The n cities are modeled as a fully connected, weighted graph, where weight w_{ij} encodes the amount of traffic between city i and j. The state of city i changes from time t to $t+1$ as

$$dS_t^{(i)} = -\beta \tilde{I}_t^{(i)} S_t^{(i)} - \alpha_t^{(i)} S_t^{(i)} + \epsilon_t^{S,(i)} \tag{6}$$

$$dI_t^{(i)} = \beta \tilde{I}_t^{(i)} S_t^{(i)} + \epsilon_t^{S,(i)} - \gamma I_t^{(i)} - \mu I_t^{(i)} \tag{7}$$

$$dR_t^{(i)} = \gamma I_t^{(i)} + \alpha_t^{(i)} S_t^{(i)} \tag{8}$$

$$dD_t^{(i)} = \mu I_t^{(i)} \tag{9}$$

where β, γ, and μ are the mean infection, recovery, and death rates, respectively. The vaccination rate $\alpha_t^{(i)}$ of city i at time t and is equal to zero until an action is taken to deploy a vaccination program to that city. The noise ϵ is sampled from a zero mean Gaussian. The effective infection exposure at city i is defined as $\tilde{I}^{(i)} = \sum_j w_{ij} I_j$, where the sum is taken over all cities, and $w_{ii} = 1$. Cities with closer index values will have higher weights, for example $w_{12} > w_{13}$.

Each episode is initialized with up to 10% of each city infected and the remainder susceptible. One city is initialized with 25% infected. The episode concludes after five cities have had vaccine programs started. The simulation is then

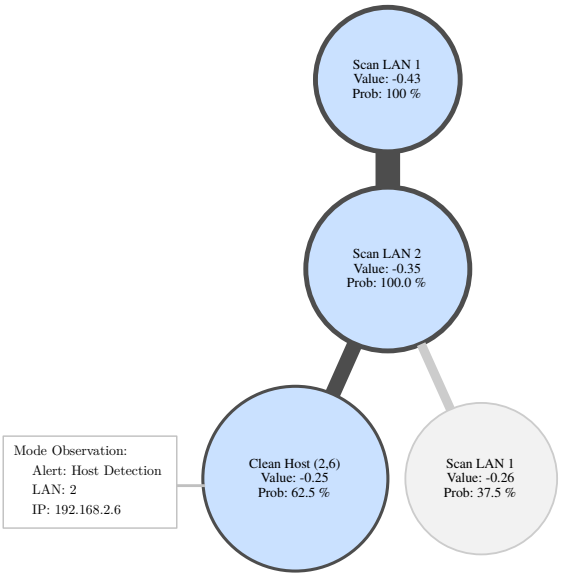


Figure 4: Cyber Security Policy Tree. The figure shows the first three layers of a surrogate tree for the cyber security task. The most frequent observation from the "Clean Host" action node set is shown.

run until all cities have zero susceptible or infected population. The reward at each time step is $r_t = a \sum_i dI_t^{(i)} + b \sum_i dD_t^{(i)}$, where a and b are parameters. We trained a neural network policy using double DQN (**?**) with n-step returns. The trained policy achieved an average score with one standard error bounds of -149.8 ± 0.6 over 100 trial episodes. We built trees for 100 random initial states with 2000 particles and a minimum particle threshold of 250 and tested their performance as forward policies. The trees achieved an average score of -152.2 ± 0.8 over the 100 trials, for an average performance drop of 1.6%. To compare to a baseline surrogate modeling approach, we also trained and tested a LIME model (**?**) with 2000 samples. The LIME average score was -184.2 ± 4.5, for an average performance loss of 23.0%.

The surrogate tree in fig. 2 provides an intuitive understanding of the learned policy. In fig. 2b we can see that the policy does not deploy vaccines to the most heavily infected cities first. It instead prioritizes cities with larger susceptible populations to give the vaccine time to take effect on a larger amount of the population.

Cyber Security

The cyber security task requires an agent to prevent unauthorized access to secure data server on a computer network. The computer network is comprised of four local area networks (LANs), each of which has a local application server and ten workstations, and a single secure data server. The compromise state of the network is not known may be observed through noisy alerts generated from malware scans. Workstations are networked to all others on their LAN and to the LAN application server. Servers are randomly connected in a complete graph. An attacker begins with a single workstation compromised and takes actions to compromise additional workstations and servers to reach the data server. The defender can scan all nodes on a LAN to locate compromised nodes with probability p_{detect}. Compromised nodes will also generate alerts without being scanned with low probability. The defender can also scan and clean individual nodes to detect and remove compromise. The reward is zero unless the data server is compromised, in which case a large penalty is incurred. The defender was trained using Rainbow DQN (**?**).

Automated systems such as this are often implemented with a human in the loop. Policies that can be more easily interpreted are more likely to be trusted by a human operator. A surrogate tree for the neural network is shown in fig. 4. Unlike the baseline neural network, the policy encoded by this tree can be easily interpreted. The agent will continually scan LAN 1 in most cases, and will only clean a workstation after malware has been detected.

Conclusions

In this work, we presented methods to construct local surrogate policy trees from arbitrary control policies. The trees are more interpretable than high-dimensional policies such as neural networks and provide quantitative estimates of future behavior. Our experiments show that, despite truncating the set of actions that may be taken at each future time step, the trees retain fidelity with their baseline policies. Experiments demonstrate the effect of various environment and algorithm parameters on tree size and fidelity in a simple grid world. Demonstrations show how surrogate trees may be used in more complex, real-world scenarios. The action node clustering presented in this work used a heuristic search method that provided good results, but without any optimality guarantees. Future work will look at improved approaches to clustering, for example by using a mixed integer program optimization. We will also explore using the scenarios simulated to construct the tree to backup more accurate value estimates, and refine the resulting policy. Including empirical backups such as these may also allow calculation of confidence intervals or bounds on policy performance (**?**).

Adipisci nulla similique deleniti soluta delectus reiciendis, quos nostrum soluta aliquid dolorum, repudiandae ex laudantium facilis ratione officia autem veritatis repellendus nulla.Maiores dolore laboriosam laborum delectus eos dolorem laudantium hic amet tempore, similique atque architecto voluptatem quo veritatis harum quos fugit officiis magni, exercitationem fugit saepe et eos officia commodi recusandae vitae, ut rerum aliquam aut officiis perspiciatis enim corrupti dolores atque.Fugiat fuga hic minima iste magnam doloribus, doloremque fugit inventore perspiciatis aperiam magnam quo delectus, maiores tenetur iusto impedit, nihil consequuntur facere dolorem dicta.Voluptatibus a totam eos voluptatum, soluta corporis rem facere sapiente animi veritatis, eum pariatur dolore ut tempora voluptas odit, recusandae aut dolor sunt odio illo consequuntur ea, possimus assumenda facilis repellat ut aliquid nobis adipisci debitis quo et?