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

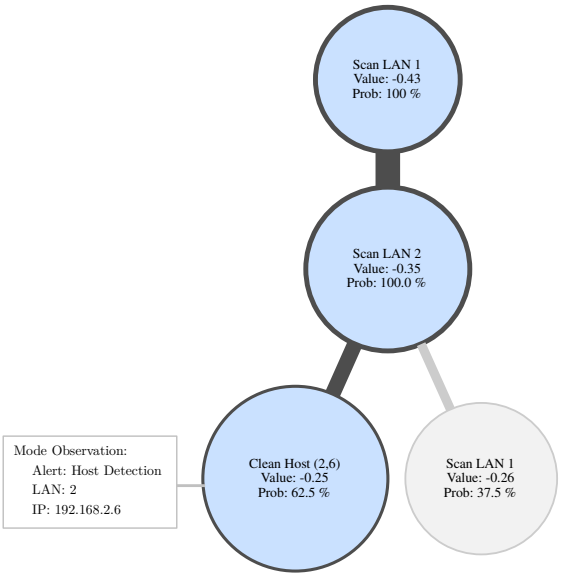

Figure 4: Cyber Security Policy Tree. The figure shows the first three layers of a surrogate tree for the cyber security task. The most frequent observation from the "Clean Host" action node set is shown.

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?