# Planning Visualizations

Christopher H. Lin and Jonathan Bragg

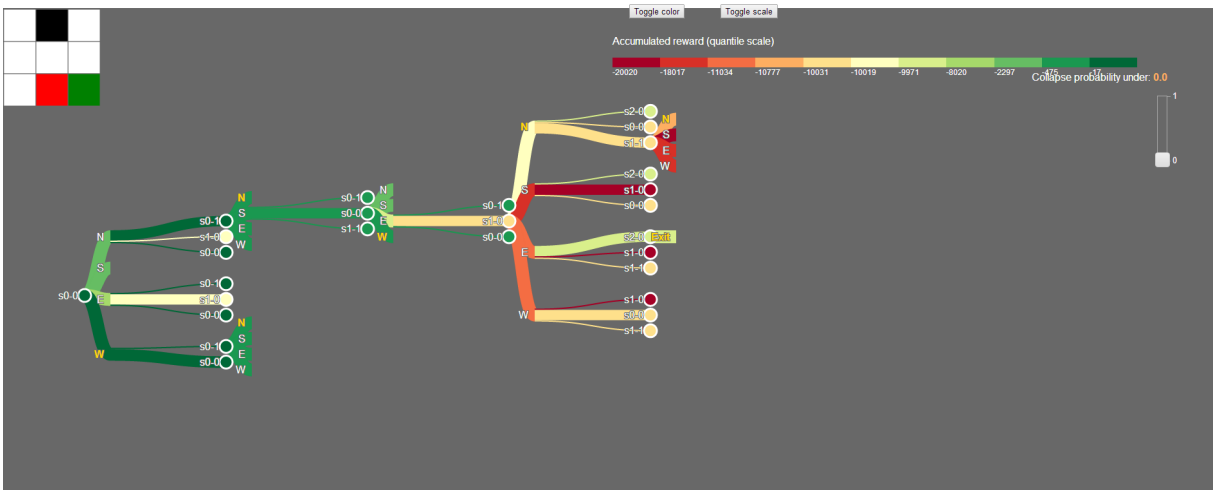


Fig. 1. Plan to Understand: Making Sense of MDPs

**Abstract**—We create a tool for visualizing Markov Decision Processes (MDPs) that assists AI and planning researchers in understanding them. MDPs are a formalism for modeling planning problems. They, as well as their solutions can be complex. Our visualization tool helps researchers to diagnose problems in their model, understand optimal policies, and devise improved algorithms for planning. Using a standard benchmark MDP from the literature, we show that researchers can answer questions like "Why did my policy choose this action instead of that action?" and "What are the possible values of the states that I might transition to after taking this action?"

## 1 Introduction

Having tools that enable quick understanding of results is essential to making progress in any research field. In particular, one needs to interpret results before deciding which next actions or steps to take. In the Planning subfield of Computer Science, researchers do not have access to such a tool. They commonly model complex planning problems using Markov Decision Processes (MDP), and then use a variety of methods to solve the MDP, in order to produce a plan, which tells an intelligent agent what optimal actions to take in order to achieve its goal, given the state of the world.

AI and planning researchers commonly find themselves scratching their heads wondering why the plans produced by their algorithms behave the way they do. We develop a general tool that not only allows users to quickly understand any arbitrary MDP, but also the optimal policies for those MDPs. Our tool enables researchers to gain insight into why certain actions are the best, so that they can possibly design modified algorithms that improve on those behaviors.

We develop a visualization technique, which we term "Multiedges," that enables the visualization of trees where nodes can be connected to several other nodes by a single link. We demonstrate the usefulness of our visualization, by using a standard MDP from the planning literature as an example. We show how interacting with the visualization for this example helps users to answer questions like "Why did my policy choose this action instead of that action?", "What are the possible values of the states that I might transition to after taking this action?", and "Given the actions the agent has taken so far, how well has it been doing?"

## 2 Related Work

Solution methods for MDPs have been well studied in the planning literature [9], but our work is the first to investigate the use of visualization techniques to better understand those solutions. The general solution to an MDP is naturally represented as a directed graph, and paths through that graph form trees with potentially repeated states. While the full graph may be infinite depending on the state space, paths that form trees are tractable and enable one to understand why the policy takes certain actions; in this work, we focus on tree representations.

Several algorithms have been developed to draw trees in the familiar node-link format, with circles denoting vertices and lines between them denoting edges. [18] introduced an algorithm for drawing "tidy" trees, which lays trees out so that parents and children are easily discernible. [11] introduced an algorithm for drawing "tidier" trees, such that the trees follow some aesthetically principles. For instance, they designed the algorithm such that the drawn trees contain no edge crossings and space is not wasted. However, both these methods are for binary trees. [17] develoepd a method for general trees, and [3] improved this algorithm to run in linear time. SpaceTrees [10] and Degree-of-Interest Trees [4] incorporate intelligent interaction into the display of trees. They allow users to click on nodes to expand them and dynamically lay out the trees as users change their views of the tree. [6] improved Degree-of-Interest Trees to handle larger amounts of data, by more appropriately displaying multiple foci.

Other work assumes that a parent node is an aggregation of its child nodes, which is not applicable in our setting. [8] introduced Treemaps, which ably display attributes of nodes, but are less able to display hierarchical structure. [15] introduced Sunbursts, a circular variation on the Treemap, to try to improve the hierarchical display of Treemaps. [19] combined techniques from Treemaps and node-link diagrams to enable a visualization creator to take advantage of the relative benefits of each approach.

A number of approaches also use interaction techniques to help nav-

igate trees and graphs. [13] used touch gestures to address issues of edge congestion. Hierarchical edge bundling [7] and interactive edge bundling [12] can also help to address graphs with large numbers of edges. Motifs [16, 5] can summarize portions of graphs in order to better understand or compare large structures. Attribute-driven [14, 2] or hierarchical aggregation [1] can also aid interactive exploration by organizing or filtering graphs according to properties of the graph or underlying data.

## 3 BACKGROUND

### 3.1 Markov Decision Processes

AI researchers frequently use Markov Decision Processes (MDPs) to model planning problems [9].

An MDP is a four-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$, where

- $\mathcal{S}$ is a finite set of discrete states.
- $\mathcal{A}$ is a finite set of all actions.
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the transition function describing the probability that taking an action in a given state will result in another state.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ is the reward for taking an action in a state and transitioning to another state.

An agent executes its actions in discrete time steps starting from some initial state. At each time step, the system is at one distinct state $\mathbf{s} \in \mathcal{S}$. The agent can then execute any action $\mathbf{a}$ from a set of *actions* $\mathcal{A}$, transitions stochastically to a new state $\mathbf{s}'$ given by the probability $T(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ , and receives a reward upon arriving, $R(\mathbf{s}, \mathbf{a}, \mathbf{s}')$. The transition process exhibits the *Markov property*, i.e, the new state $\mathbf{s}'$ is independent of all previous states given the current state $\mathbf{s}$.The model assumes *full observability*, i.e., after executing an action and transitioning stochastically to a next state as governed by $\mathcal{T}$, the agent has full knowledge of the state.

A solution to an MDP is in the form of a *policy*, $\pi : \mathcal{S} \rightarrow \mathcal{A}$, a mapping from the state space to the action space. The polcy tells the agent the best action to take for every state in which the agent is. AI researchers usually try to find an *optimal policy* ($\pi^* : \mathcal{S} \rightarrow \mathcal{A}$), which is a policy that achieves the maximum expected utility, or sum of rewards. We evaluate any policy $\pi$ by its *value function*, the set of values that satisfy the following equation:

$$V^{\pi}(\mathbf{s}) = \sum_{\mathbf{s}' \in \mathcal{S}} T_{\pi(\mathbf{s})}(\mathbf{s}'|\mathbf{s})(R(\mathbf{s}, \pi(\mathbf{s}), \mathbf{s}') + \gamma V^{\pi}(\mathbf{s}')). \quad (1)$$

$\gamma \in (0, 1]$ is the *discount factor*, which controls the value of future rewards. Any optimal policy's value function must satisfy the following system of *Bellman equations*:

$$V^{*}(\mathbf{s}) = \max_{\mathbf{a} \in \mathcal{A}} \sum_{\mathbf{s}' \in \mathcal{S}} T_{\mathbf{a}}(\mathbf{s}'|\mathbf{s})(R(\mathbf{s}, \mathbf{a}, \mathbf{s}') + \gamma V^{*}(\mathbf{s}')). \quad (2)$$

The $Q^*$*-value* of a state-action pair $(\mathbf{s}, \mathbf{a})$ is the value of taking action $\mathbf{a}$ in state $\mathbf{s}$ followed by $\pi^*$ afterwards. More concretely,

$$Q^{*}(\mathbf{s}, \mathbf{a}) = \sum_{\mathbf{s}' \in \mathcal{S}} T_{\mathbf{a}}(\mathbf{s}'|\mathbf{s})(R(\mathbf{s}, \mathbf{a}, \mathbf{s}') + \gamma V^{*}(\mathbf{s}')). \quad (3)$$

The optimal value function can equivalently be expressed by:

$$V^{*}(\mathbf{s}) = \max_{\mathbf{a} \in A} Q^{*}(\mathbf{s}, \mathbf{a}). \quad (4)$$

## 4 METHODS

Our visualization takes as input an MDP specification, and uses D3 to create an interactive tree that represents the MDP and its optimal policy, using the Reingold-Tilford "tidy" tree layout algorithm [11]. In this section, we first discuss how we process the MDP specification efficiently, and then discuss how we encode the MDP as a variation on the standard node-link diagram.

### 4.1 MDP Processing

Our implementation assumes that the MDP is specified by a Python class file. We use value iteration to solve the MDP, but one could implement more sophisticated methods for solving more complex MDPs.

We output our solution to the MDP as a relational data file. Our visualization also accepts an optional JSON file specifying the initial tree that should be displayed. Since specifying the entire tree before interaction begins would result in a JSON file whose size grows exponentially in the depth of the tree, we instead use the relational data file specifying the solution to the MDP in order to generate portions of the tree on demand as the user explores.

### 4.2 Visual Encodings

#### 4.2.1 MultiEdges

MDPs can almost be represented by node-link diagrams. However, we soon see a problem with a naive attempt. If states are nodes and actions are edges, then an edge must lead to multiple nodes because of the transition function. We resolve this inadequate visual mapping by using "multiedges." Figure 2 shows the multiedges in our visualization. To create "multiedges," we also let the actions be nodes. State nodes are always connected to action nodes, and state nodes are never connected to state nodes, and action nodes are never connected to action nodes. We then make the action node invisible, so that an action looks like one edge that splits into different states. We call the edge leading from a state node to an action node an "action edge" and the edge leading from an action node to a state node a "transition edge."

We encode the transition probability as the width of the edge. Wider edges correspond to more likely transitions. Every action edge has thickness corresponding to probability 1. Then, that edge is split into several transition edges that all have thickness corresponding to their transition probabilities, in a linear scale, so that if you stack the transition edges on top of each other, their total thickness corresponds to probability 1.

Multiedges are fully interactive. Mousing over action and transition edges causes the edges to darken to show that both types of edges are expandable. Clicking a collapsed action edge expands the transition edges and reveals state nodes. Clicking the label for an action edge produces the same behavior. Clicking an expanded action edge retracts the transition edges.

#### 4.2.2 Color

In addition to visualizing the structure of the MDP as a tree and the transition probabilities as edge widths, it is important to visualize the value of states and actions in order to enable understanding of why a policy takes certain actions. We encode value using color. An obvious alternative to encoding value with color is to encode it with size. However, both states and actions have value (or expected value in the case of an action edge), and since we are already encoding transition probabilities using width, it is not clear how to use size to show value. Color gives a nice sense of flow, or the change in value from the root of the tree to a leaf in the tree. We choose to use a divergent color scale ranging from red to green due to the strong domain-specific associations of green with high positive value and red with high negative value.

The default view uses color to indicate the value of an action or state, as we defined earlier. We color action edges according to their $Q$-value, and transition edges leading to nodes according to the value of that node. This view corresponds to expected future rewards. We also provide an alternate view corresponding to accumulated (past) rewards. A natural third view would show the sum of accumulated rewards and expected future reward, but we leave this view to future work.

The optimal policy action is indicated using a different color for the text corresponding to that action (gold instead of white), but can also be inferred by the color of the action with the highest expected reward.

One obstacle we had to overcome in using color to show value is how to show a wide range of values, while also enabling the user to distinguish between smaller ranges of values. On the one hand, some

states may have extreme values like the fire pit in our example MDP. On the other hand, a user should be able to distinguish between the values of different actions at a state, which may occupy a more narrow range of value space. We found that using a quantile scale for color, where they range of a color corresponds to the density of values in that range, provides a good compromise between these two objectives. While the quantile bins enable finer distinctions among values in dense ranges, we use tooltips to provide additional precision on demand; hovering over an action or state shows the precise value.

Showing accumulated rewards presents an additional challenge because one cannot compute the full range of possible values like one can for expected future reward. As a path in our tree grows longer, it is possible that the accumulated value will grow arbitrarily large or small. To handle any possible range of values, we compute the range dynamically using the states and actions that appear in the visualization at any point in time. Each time the user expands a node or edge, we dynamically update the legend and the colors shown in the tree to accommodate the new range of values.

#### 4.2.3 User-provided State/Action Visualization

To allow for domain-specific visualizations of MDPs, we provide functionality for the user to provide their own visualization of the state/action space of the MDP, which we plug into our visualization. Should the user provide visualizations $Z_1, Z_2, Z_3$ and functions $f : \mathcal{S} \mapsto Z_1$, $g : \mathcal{S} \times \mathcal{A} \mapsto Z_2$, and $h : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto Z_3$ to our system, we can then show domain-specific visualizations in addition to our abstract representation of the MDP. Whenever a user mouses over a state node in our visualization, we call $f$. Whenever a user mouses over an action edge, we call $g$. Whenever a user mouses over a transition edge, we call $h$.

Figure 3 shows an example. We see that $Z_1, Z_2, and Z_3$ is simply a grid with cells that can be colored. When the user mouses over the action edge "North", the grid shows that the agent is currently in state (0,1) in black, and that taking action "North" can result in two possible transitions, shown in orange.

#### 4.2.4 Probability Filter

In an effort to allow users to quickly filter nodes and links that they no longer wish to explore, we provide a slider that collapses parts of the tree. By moving the slider, the user can set a probability threshold, $p$. The visualization then begins collapsing nodes starting at the root. Any state node that has a transition edge leading to it with less than probability $p$ is collapsed, but its children are not recursively collapsed. In order to make clearer that we are not recursively collapsing nodes, we implement transitions that show each exiting element as collapsing into the parent closest to the root that is being collapsed.

## 5 Results

We now introduce an example to demonstrate the usefulness of our visualization in aiding researchers in their understanding of policies and value functions.

### 5.1 Example MDP

We use a standard MDP from the AI literature, a 3x3 GridWorld. The state space $\mathcal{S}$ is $\{(x, y) : x \in \{0, 1, 2\}, y \in \{0, 1, 2\}\}$. One can simply imagine a 3x3 grid. The actions are $\mathcal{A} = \{North, South, East, West\}$. The transition function is as expected, but we throw in some stochasticity. If an agent takes an action $\mathbf{a} in \mathcal{A}$, the expected result of that action will occur 80% of the time. 10% of the time, the agent will move in an orthogonal direction. For instance, if the agent decides to go North, it will actually move one space up 80% of the time, 10% of the time it will move to the left, and 10% of the time it will move to the right. Should the agent be unable to move in the direction specified, it will remain in its original state. In the bottom right corner is the goal state $(2, 0)$. If the agent reaches this state, it will receive a reward of 100. However, if it enters the fire pit at $(1, 0)$, it will receive a reward of -1000. Otherwise the agent pays a reward of -10 for every action it takes. Therefore, the agent wishes to reach the goal state as quickly as possible.

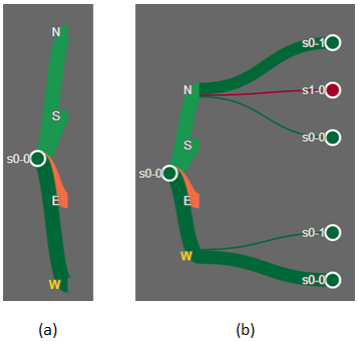


Fig. 2. (a) The visualization the user initially sees. (b) The visualization after the user expands the "North" and "West" actions.

### 5.2 Exploring the MDP

Upon starting our visualization with this example MDP, the user is presented with just the starting state $(0, 0)$ and four possible actions, as shown in Figure 2. The user can quickly see that the optimal policy is to take action "West," since the edge representing that action is the darkest green and the text of the action is in gold. However, the user is confused. Why should the optimal action be "West?" That "East" is the least optimal action is obvious, since it leads directly to the firepit, but intuitively, "North" should be the best action since it begins the journey to the goal state without going through the high penalty fire pit.

To learn more, the user can expand the actions "North" and "West." It becomes a little clearer now why the best action is "West." Should the agent go "North," there is a small, but real chance that it will fall into the firepit, denoted by the thin, but very dark red line leading to (1,0). However, the "West" action ensures that the agent will not fall into the firepit and can still make progress by stochastically moving "North" with small probability.

Now suppose the user wishes to learn more about taking the action "West" and landing in state (0,1) so he expands that state. The user intuitively thinks that taking the actions "North" and "East" should be equivalent, because the optimal path should take the agent to the state (1,2) (The agent would not want to move "East" from the state (1,1) because it might fall into the fire pit). However, the visualization clearly shows that "North" is the better action. The user mouses over the the "North" and "East" actions to see why. The 2d visualization of the state space provided by the user clearly shows the reason – moving "North" ensures progress, whereas moving "East" might cause the agent to return to its starting position. The user could have also realzied this by expanding the two action nodes.

Now suppose that the user wishes to learn what the agent should do should it fall into the fire pit by taking suboptimal actions. The user expands the (1,0) node and sees that the optimal action is to go "North???" Intuitively, now that agent is already in the fire pit, it should just go "East" and quickly end its misery. Expanding the two actions nodes is immediately illuminating (the author admits that even this scenario was unexpected, and was taught something by the visualization). THe user sees that if the agent takes action "East," it has a low, but real chance of staying in the fire pit, because it might stochastically try to move "South" and since it cannot move further south, it stays where it is. Taking the action "North" on the other hand, ensures that the agent will reach the goal without the possibility of falling into the firepit again as long as it continues to take optimal actions.

### 5.3 Some other useful tools

#### 5.3.1 Probability Filter

Figure 5 shows how the user can use the probability filter slider to quickly clean up the display without needing to manually collapse many nodes. Suppose the user has been exploring the tree heavily, and

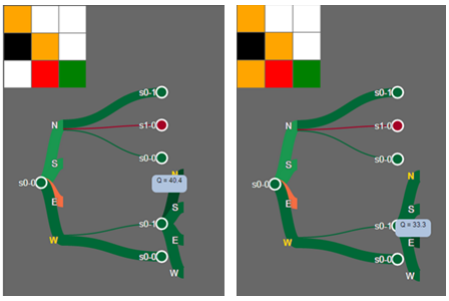

Fig. 3. The user mouses over the actions “North” and “East” from state “s0-1”.

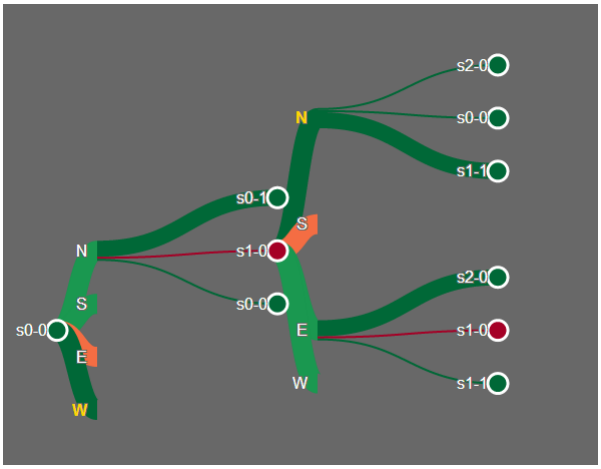


Fig. 4. The user expands the actions “North” and “East” from state “s1-0”.

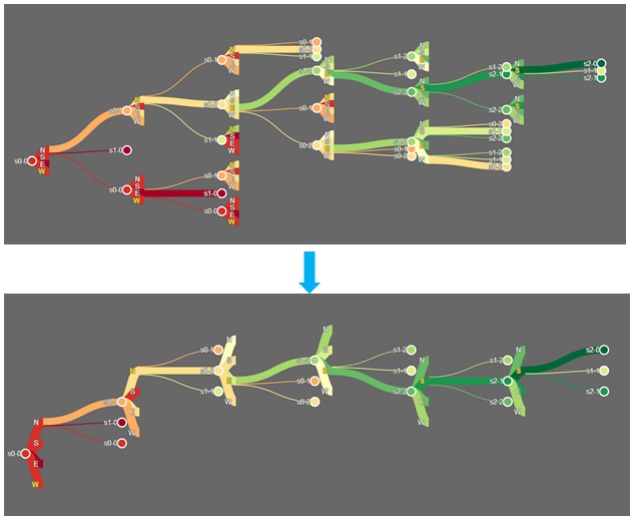

Fig. 5. The user collapses all expanded nodes that resulted from transitions of probability less than or equal to 0.1

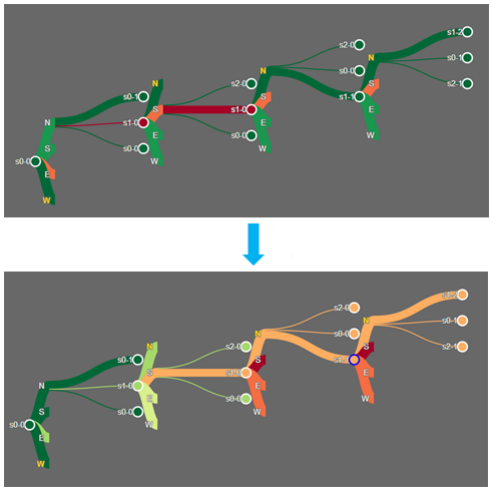

Fig. 6. The user changes the colors to show accumulated reward instead of static state value.

investigating what happens should low-probability transitions occur to the agent. The user has been trying to understand how low-probability events compare against high-probability events, and has a large part of the tree expanded. But the user now wants to start over with only the high-probability events showing. By simply using the probability filter slider, the user is able to quickly clean up the display to show only what is desired, and begin exploring again.

### 5.3.2 Cumulative Reward

Figure 6 shows how the user can change the coloring of our visualization to show accumulated reward instead of static state values and q-values. Suppose the user has expanded the trajectory shown. From the coloring, it is not immediately obvious that at the leaves of the tree, given the actions the agent has taken (the actions the user has expanded), the agent has performed very poorly. However, as soon as the user changes the color to show accumulated reward, it becomes very clear that the agent has been taking suboptimal actions and racking up very large penalties. While it was possible for the agent to gain decent rewards (as shown by the green at the root of the tree), the user now sees that the agent has managed to do all the wrong things.

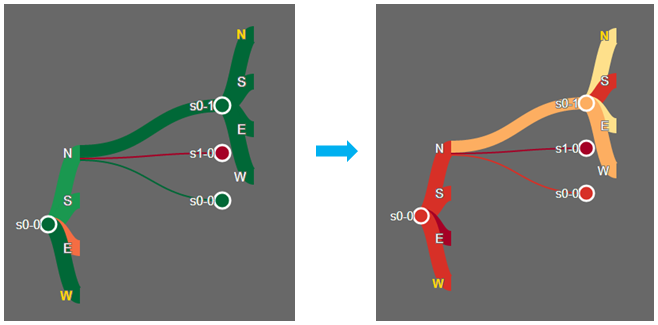


Fig. 7. The user cannot distinguish between the relative values of the actions from state (0,1), so they toggle the color scale change.

### 5.3.3 Color Scale Change

Figure 7 shows how our "Toggle Scale" button allows the user to understand actions and states that have the same color. When the user expands state (0,1), although she can clearly see that "North" is the optimal action to take, she is unable to distinguish the relative differences in value of the actions. Certainly, the user can mouse over the actions to see the q-values by tooltip, but the faster method is to change the scale of the colors. By doing so, we see that the user is now quickly able to see that "South" is a relatively horrible action, "West" is slightly better, and perhaps "East" requires more investigation.

## 6 Discussion and Future Work

There are several future directions for our work:

- While we have considered MDPs, also popular among planning and AI researchers are Partially-Observable Markov Decision Processes (POMDPS). POMDPs generalize MDPs by assuming that the state of the agent is no longer observable. Instead, as the agent interacts with the world, it receives observations that give it hints about what state it might be in. POMDPs are useful for modeling many real-world scenarios, like robots that must only rely on noisy sensors. A POMDP can be reduced to an MDP by letting each MDP state be a distribution over the possible POMDP states. We imagine a visualization where each node could be a pie graph, and hovering over nodes could indicate in another visualization a distribution over states.

- Many researchers would like to be able to understand their particular planning algorithms. While our visualization shows the result of a planning algorithm, it does not show how the algorithm executed. For instance, the algorithm Value Iteration works by repeatedly recaculating state values until it converges. While our algorithm displays the final value, it does not show the intermediate ones. We imagine a visualization with a slider that can show the changing values of the states as the solution algorithm progresses.

- We use a tree in our visualization, because when AI researchers are exploring their MDPs and policies, they generally do not attempt to understand the entire MDP and tree at once. Instead, they focus in small areas of the MDP and follow trajectories through the state space. Our visualization enables this type of interaction. However, alternatively, one can imagine displaying a graph. A graph might be useful for overall high-level understanding of the MDP. If we still want to incorporate time into a graph representation, we imagine a functionality that changes the graph to show where the agent is as it moves through states.

- We imagine that sometimes a user may wish to examine some area of the tree which is not close to the starting state of the MDP. Currently our visualization requires such a user to slowly click through the tree until they find the state they are looking for. Thus, some functionality for querying states would be useful. We imagine the user to be able to replace the root of the tree with a state of their choosing.

## References

[1] J. Abello, F. Van Ham, and N. Krishnan. Ask-graphview: A large scale graph visualization system. *Visualization and Computer Graphics, IEEE Transactions on*, 12(5):669–676, 2006.

[2] A. Bezerianos, F. Chevalier, P. Dragicevic, N. Elmqvist, and J.-D. Fekete. Graphdice: A system for exploring multivariate social networks. In *Eurovis*, 2010.

[3] C. Buchheim, M. Junger, and S. Leipert. Improving walker's algorithm to run in linear time. In *Technical Report*, 2002.

[4] S. K. Card and D. Nation. Degree-of-interest trees: A component of an attention-reactive user interface. In *Advanced Visual Interface*, 2002.

[5] C. Dunne and B. Shneiderman. Motif simplification: improving network visualization readability with fan, connector, and clique glyphs. In *CHI*, 2013.

[6] J. Heer and S. K. Card. Doitrees revisited: Scalable, space-constrained visualization of hierarchical data. In *Advanced Visual Interface*, 2004.

[7] D. Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. In *Infovis*, 2006.

[8] B. Johnson and B. Schneiderman. Tree-maps: A space-filling approach to the visualization of hierarchical information structures. In *IEEE Visualization*, 1991.

[9] Mausam and A. Kolobov. *Planning with Markov Decision Processes: An AI Perspective*. Morgan and Claypool Publishers, 2012.

[10] C. Plaisant, J. Grosjean, and B. B. Bederson. Spacetree: Supporting exploration in large node link tree, design evolution and empirical evaluation. In *Infovis*, 2002.

[11] E. M. Reingold and J. S. Tilford. Tidier drawings of trees. In *IEEE Transactions on Software Engineering*, 1981.

[12] N. H. Riche, T. Dwyer, B. Lee, and S. Carpendale. Exploring the design space of interactive link curvature in network diagrams. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, 2012.

[13] S. Schmidt, M. A. Nacenta, R. Dachselt, and S. Carpendale. A set of multi-touch graph interaction techniques. In *ITS*.

[14] B. Shneiderman and A. Aris. Network visualization by semantic substrates. In *Infovis*, 2006.

[15] J. Stasko and E. Zhang. Focus+context display and navigation techniques for enhancing radial, space-filling hierarchy visualizations. In *Infovis*, 2000.

[16] T. von Landesberger, M. Gorner, and T. Schreck. Visual analysis of graphs with multiple connected components. In *Visual Analytics Science and Technology*, 2009.

[17] J. Q. Walker. A node positioning algorithm for general trees. 1990.

[18] C. Wetherell and A. Shannon. Tidy drawings of trees. In *IEEE Transactions on Software Engineering*, 1979.

[19] S. Zhao, M. J. McGuffin, and M. H. Chignell. Elastic hierarchies: Combining treemaps and node-link diagrams. In *Infovis*, 2005.