SAND2022-9852C

Sandia National Laboratories

Exceptional service in the national interest

Shortest Path Navigation using Reinforcement Learning

By Srideep Musuvathy, Tyson Bailey, Abel Osvaldo Gomez Rivera, Meghan Sahakian

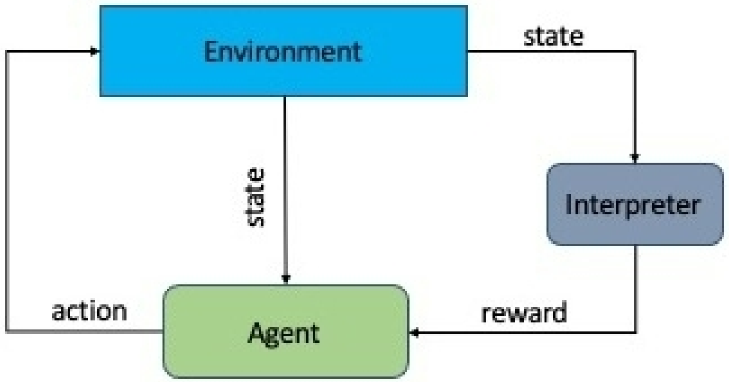
RELACSS

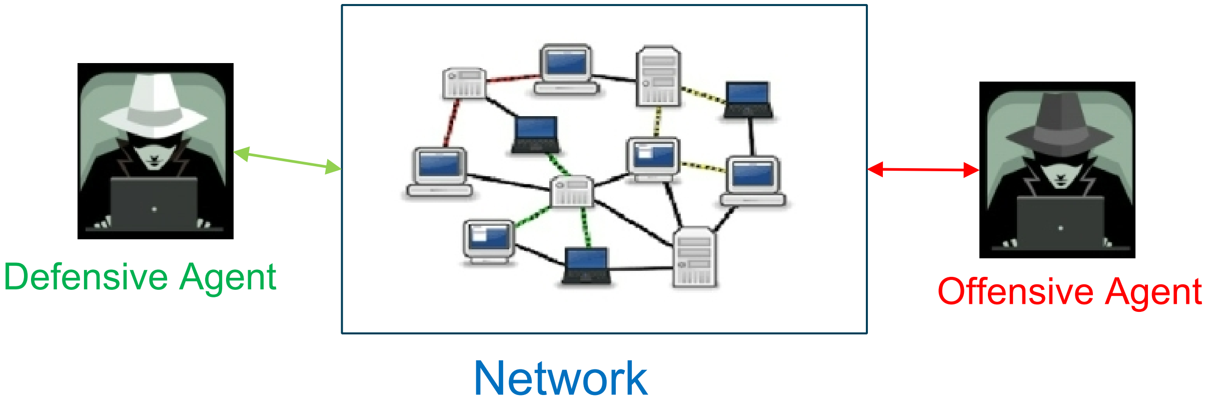
7/14/2022

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Cyber RL in 60 seconds





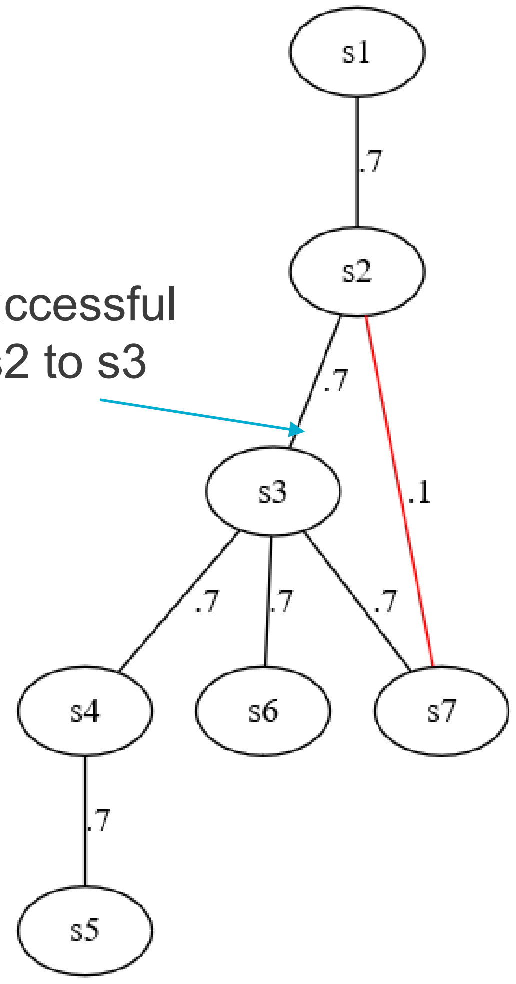
How can we test and protect our systems?

Problem

- The goal is for an attacker to navigate a graph with edges that can disappear based on some probability function
- The probability of disappearing is simply a stand in for a defensive agent killing edges on the network to constrain the attacker
- Can our agent learn to take a longer path if it’s more reliable?

Network

Probability of successful transition from s2 to s3



State Representations

- First attempt was to use the following state representation:

Current	S1	S2	S3	S4	S5	S6	S7

Which node we are on
values 1 through 7

Whether we have visited a particular node
Values 0 and 1

Action Space

- Similar to the state space we have the option of selecting *any* node

S1	S2	S3	S4	S5	S6	S7

Rewards

- -1 for each step
- 100 for finding the terminating node.
- Given the probabilities proposed in the graph, statically the agent should *prefer* to take S2->S3->S7

Results

- Able to train (easily) using PPO and solve the problem.

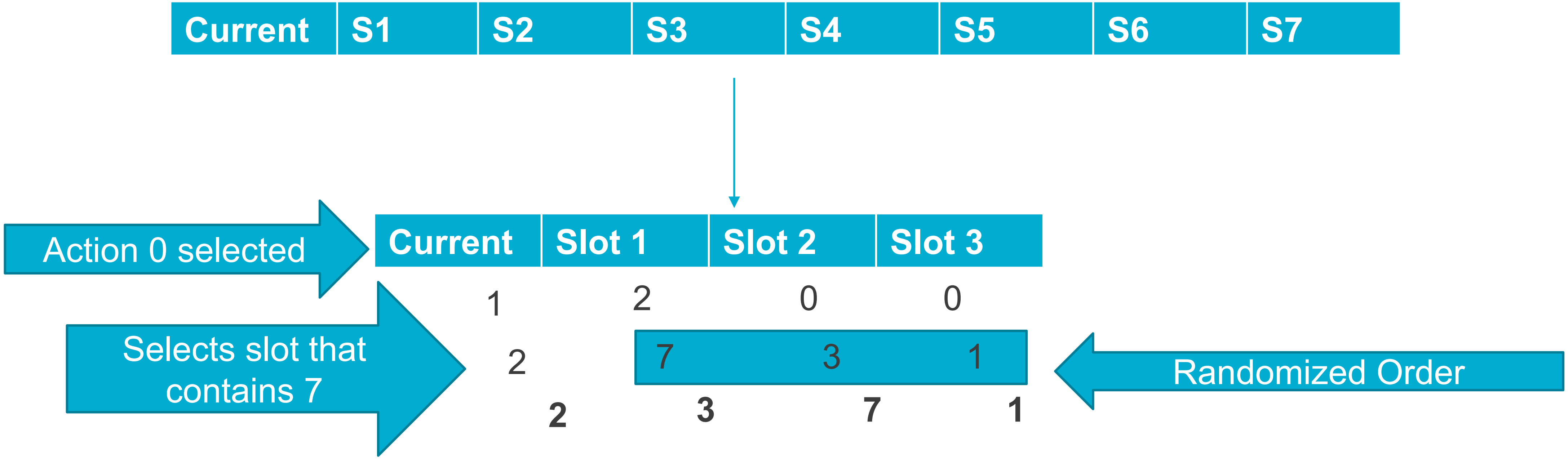
Issues Arise

- However, when training, we teach the agent to follow the longer path, but what happens if the shorter path exists how can we make the agent take it?
- Since RL tends to find a optimal path, or policy, training on a system that behaves a certain way will result in a policy that is static, meaning that if we train it to prefer s2->s3->s7 then we will always attempt that path, independent of whether s2->s7 is possible.
- The current state structure and action space, require complete knowledge of the graph, meaning that you have all nodes in your state, as well as all nodes in your actions

Solution (State Space)

- So we changed the problem slightly, rather than keeping all nodes in memory as to which ones we've visited, we propose simply keeping track of what node we are on, and a subset of nodes we can see.
- This means that if we have a very large network, we don't need such a large input space.
- We default it to 3 slots currently
- The available nodes are filtered by the probability, then all remaining nodes are shuffled and the first 3 selected.
- If less than 3 are available the slots will contain a 0 starting in the rightmost column

Solution (State Space)



Solution (Action Space)

- Similar to the state space, we chose to reduce our action space from enumerating all possible nodes we can visit, we constrain it to only being able to select a slot from the visible/available nodes.
- If the agent selects an empty slot that is a “do nothing” action.

Results

- After taking 10000 timesteps training (not that long) the below is an example run
- For the following run, our agent started on s1, and could see s2. It chose to move to s2.
- Then it was in s2 and was able to see s3 or s1, so it chose to go to s3.
- Finally while it was in s3, the chances of seeing s7 are quite high and it took it.

```
Agent Prior state [1 2 0 0] New State [2 3 1 0] Action Taken 0 Reward for Action -1
Agent Prior state [2 3 1 0] New State [3 7 6 4] Action Taken 0 Reward for Action -1
Agent Prior state [3 7 6 4] New State [7 3 2 0] Action Taken 0 Reward for Action 100
```

Results

- Another example from the same training run
- For the following run, our agent started on s1, but couldn't see anything so it wasn't able to move.
- The next round it could see s2. It chose to move to s2.
- Then it was in s2 and was able to see s7, so it chose to go to s7.

```
Agent Prior state [1 0 0 0] New State [1 2 0 0] Action Taken 0 Reward for Action -1
Agent Prior state [1 2 0 0] New State [2 7 0 0] Action Taken 0 Reward for Action -1
Agent Prior state [2 7 0 0] New State [7 3 0 0] Action Taken 0 Reward for Action 100
```

Results

- Another example from the same training run
- For the following run, our agent started on s1, and it could see all 3 edges.
- Here it can decide, does it go to s7 or s3, it chooses s7

```
Agent Prior state [1 2 0 0] New State [2 7 1 3] Action Taken 0 Reward for Action -1
Agent Prior state [2 7 1 3] New State [7 3 0 0] Action Taken 0 Reward for Action 100
```

Results

- Another example from the same training run
- This does the same thing as last run, but if you note, all the prior runs, picked the 0^{th} slot every time, it just happened to be that the best action was in slot 0 each time.
- However, this example shows the agent learned that 7 was the important number and not slot 0.

```
Agent Prior state [1 2 0 0] New State [2 1 3 7] Action Taken 0 Reward for Action -1
Agent Prior state [2 1 3 7] New State [7 3 0 0] Action Taken 2 Reward for Action 100
```

Results

- Finally just to show, we do have the possible nodes in a shuffled order, this is the code that defines the path, you can see 3,7,1 in the code, but the state below shows 1,3,7

```
self.lookup = {
        1 : [2],
        2 : [3,7,1],
        3 : [4,6,7,2],
        4 : [5,3],
        5 : [4],
        6 : [3],
        7 : [2,3]
    }
```

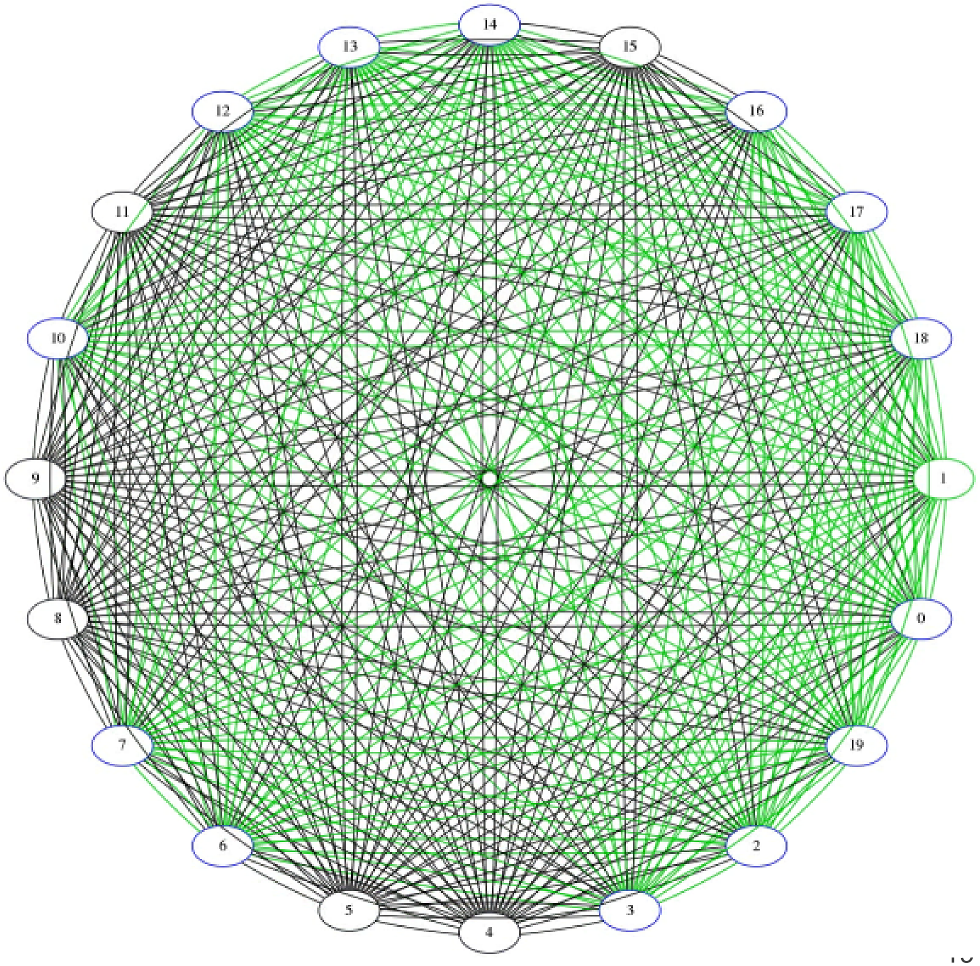
```
Agent Prior state [1 2 0 0] New State [2 1 3 7] Action Taken 0 Reward for Action -1
Agent Prior state [2 1 3 7] New State [7 3 0 0] Action Taken 2 Reward for Action 100
```

Making it more interesting

- Next we asked the question, can we learn on a more general graph and then perform the same experiment on the same smaller graph and achieve our desired goal?

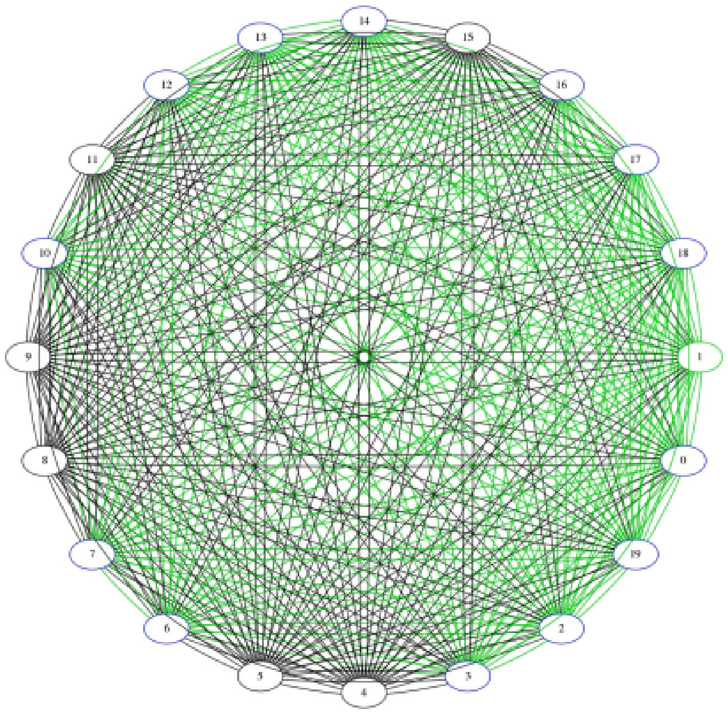
Large Graph

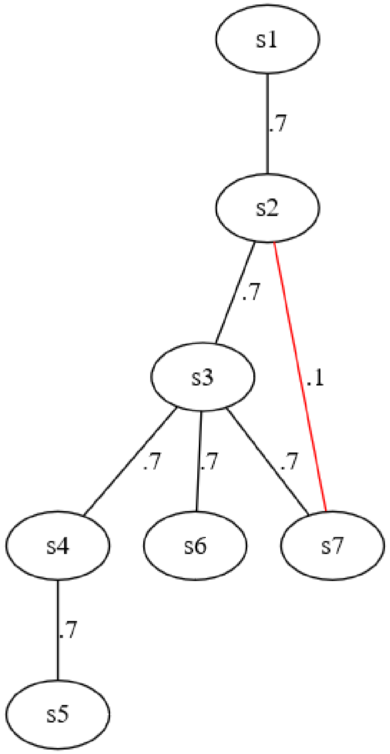
- 20 node fully connected graph.
- Green lines indicate active edges for a given timestep



Results

- The agent was successful on the smaller graph starting from s1 similar to the earlier experiments.
- However, repeated running based on probabilities it occasionally jumped into a segment of the network it got stuck in (namely s4/s5), more training might be required
- Would this work in a higher fidelity environment?





Higher Fidelity Environment

- Built the above network in Septre, using virtual machines.
- Built a translation layer to enable moving around between machines.
- Built a tool to search for a target file on the machine
- Check for any missing edges, and force the agent to return to the last node it has full access to.
- Built a python script that turns the network edge between s2 and s7 on and off for a minute at a time.

It worked!

Summary

- Pros:
  - This approach seems to work well with some appearance of generalization
  - It avoids hardcoding the underlying graph structure.
  - It prevents you from requiring total knowledge of all nodes present ahead of time
  - It avoids having a static policy that fails when an edge is removed, because it prefers the policy and the edge isn’t in the state.
- Cons:
  - Got stuck in the smaller graph, more training may fix this
  - Larger graphs (200 nodes) may require a LOT more time to train

Core Challenges

- How do we define what a target machine is? In this case we consider S7 to be our goal node, we could replace the “hard coded” S7 with a node that has a flag of interest.
- How would this affect the problem? This becomes a shortest path from all nodes to all other nodes (Dijkstras algorithm)
- Requires a translation layer to work in the target environment

Future work

- Defensive Agent experimentation
- Improve fidelity of target environment
- Continue Attacking Agent training
  - Possibly compare other RL algorithms
  - Add more useful capabilities, such as scanning for and copying files
- Better definition of scoring