A Heuristic for the Tree Augmentation Problem Using Reinforcement Learning

Cody Klingler

April 27, 2022

Lane Department of Computer Science and Electrical Engineering
West Virginia University

Motivations

The Tree Augmentation Problem (TAP) is an NP-hard optimization problem with applications in networking.

There exist polynomial-time approximation algorithms which guarantee a solution within a constant factor of the optimal solution. These algorithms don’t have public implementations and most are quite large or complex.

The goal of my project is to create a heuristic using reinforcement learning which performs on par with these approximation algorithms.

2-Edge Connected

2-Edge Connected

A graph is considered to be 2-edge connected if there is no edge whose removal leaves the graph disconnected.

2-Edge Connected

A graph is considered to be 2-edge connected if there is no edge whose removal leaves the graph disconnected.

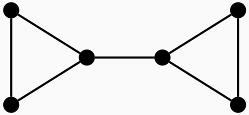


Figure 1: A graph that is not 2-edge connected

2-Edge Connected

A graph is considered to be 2-edge connected if there is no edge whose removal leaves the graph disconnected.

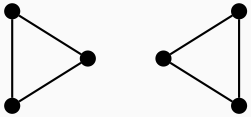


Figure 1: A graph that is not 2-edge connected

2-Edge Connected

A graph is considered to be 2-edge connected if there is no edge whose removal leaves the graph disconnected.

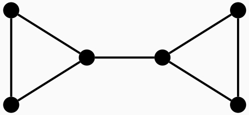


Figure 1: A graph that is not 2-edge connected

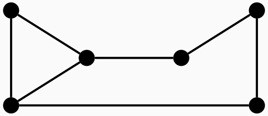


Figure 2: A graph that is 2-edge connected

2-Edge Connected

A graph is considered to be 2-edge connected if there is no edge whose removal leaves the graph disconnected.

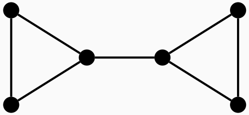


Figure 1: A graph that is not 2-edge connected

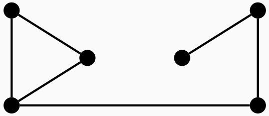


Figure 2: A graph that is 2-edge connected

2-Edge Connected

A graph is considered to be 2-edge connected if there is no edge whose removal leaves the graph disconnected.

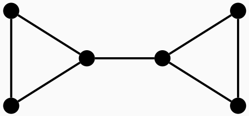


Figure 1: A graph that is not 2-edge connected

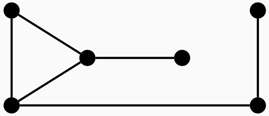


Figure 2: A graph that is 2-edge connected

2-Edge Connected

A graph is considered to be 2-edge connected if there is no edge whose removal leaves the graph disconnected.

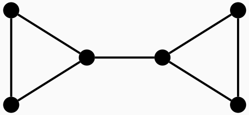


Figure 1: A graph that is not 2-edge connected

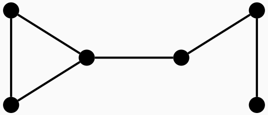


Figure 2: A graph that is 2-edge connected

2-Edge Connected

A graph is considered to be 2-edge connected if there is no edge whose removal leaves the graph disconnected.

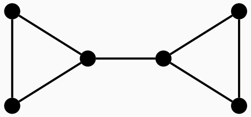


Figure 1: A graph that is not 2-edge connected

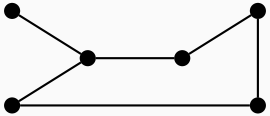


Figure 2: A graph that is 2-edge connected

Tree Augmentation Problem

The Tree Augmentation Problem (TAP) is defined as follows:

Tree Augmentation Problem

The Tree Augmentation Problem (TAP) is defined as follows:

Given a graph $G = (V, E)$ and a tree $T = (V, F)$ where $E \cap F = \emptyset$, find a minimal $E' \subseteq E$ such that $(V, E' \cup F)$ is 2-edge connected.

Tree Augmentation Problem

The Tree Augmentation Problem (TAP) is defined as follows:

Given a graph $G = (V, E)$ and a tree $T = (V, F)$ where $E \cap F = \emptyset$, find a minimal $E' \subseteq E$ such that $(V, E' \cup F)$ is 2-edge connected.

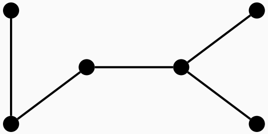


Figure 3: A TAP problem

Tree Augmentation Problem

The Tree Augmentation Problem (TAP) is defined as follows:

Given a graph $G = (V, E)$ and a tree $T = (V, F)$ where $E \cap F = \emptyset$, find a minimal $E' \subseteq E$ such that $(V, E' \cup F)$ is 2-edge connected.

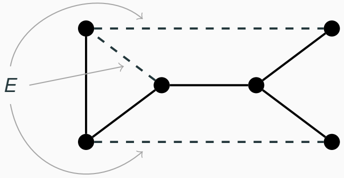


Figure 3: A TAP problem

Tree Augmentation Problem

The Tree Augmentation Problem (TAP) is defined as follows:

Given a graph $G = (V, E)$ and a tree $T = (V, F)$ where $E \cap F = \emptyset$, find a minimal $E' \subseteq E$ such that $(V, E' \cup F)$ is 2-edge connected.

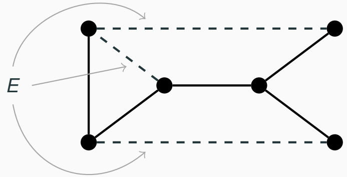


Figure 3: A TAP problem

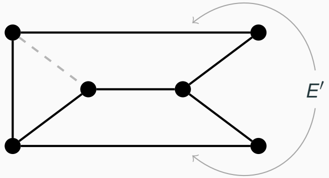


Figure 4: TAP solution

Covering an Edge

Covering an Edge

An edge $(u, v) \in E$ is said to cover $(u', v') \in F$ if the tree-path between u and v, $PATH(u, v) \subseteq F$, contains (u', v').

Covering an Edge

An edge $(u, v) \in E$ is said to cover $(u', v') \in F$ if the tree-path between u and v, $PATH(u, v) \subseteq F$, contains (u', v').

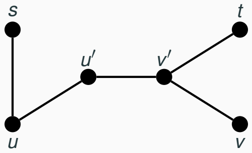


Figure 5: A tree

Covering an Edge

An edge $(u, v) \in E$ is said to cover $(u', v') \in F$ if the tree-path between u and v, $PATH(u, v) \subseteq F$, contains (u', v').

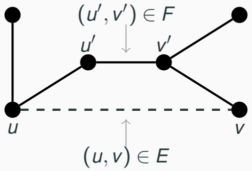


Figure 5: Add edges E

Covering an Edge

An edge $(u, v) \in E$ is said to cover $(u', v') \in F$ if the tree-path between u and v, $PATH(u, v) \subseteq F$, contains (u', v').

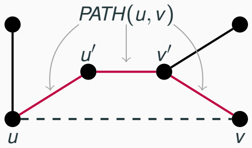


Figure 5: (u, v) covers (u', v')

Covering an Edge

An edge $(u, v) \in E$ is said to cover $(u', v') \in F$ if the tree-path between u and v, $PATH(u, v) \subseteq F$, contains (u', v').

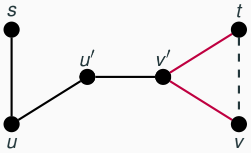


Figure 5: (t, v) does not cover (u', v')

Covering an Edge

An edge $(u, v) \in E$ is said to cover $(u', v') \in F$ if the tree-path between u and v, $PATH(u, v) \subseteq F$, contains (u', v').

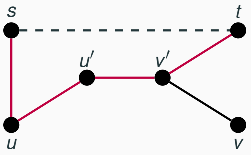


Figure 5: (s, t) covers (u', v')

Note that adding (u, v) to F creates a cycle along $PATH(u, v)$. Therefore removing an edge $e \in PATH(u, v) \cup (u, v)$ from $(V, F \cup (u, v))$ does not disconnect the graph.

Covering an Edge

An edge $(u, v) \in E$ is said to cover $(u', v') \in F$ if the tree-path between u and v, $PATH(u, v) \subseteq F$, contains (u', v').

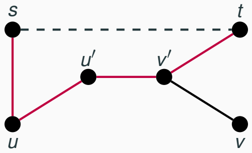


Figure 5: (s, t) covers (u', v')

Note that adding (u, v) to F creates a cycle along $PATH(u, v)$. Therefore removing an edge $e \in PATH(u, v) \cup (u, v)$ from $(V, F \cup (u, v))$ does not disconnect the graph.

The graph $(V, E' \cup F)$ is 2-edge connected if and only if each edge in F is covered by an edge in E'.

Contracting Covered Edges

Contracting Covered Edges

If it is known that an edge $(u, v) \in E$ is in the solution E', then all edges along $PATH(u, v)$ may be contracted without loss of generality.

Contracting Covered Edges

If it is known that an edge $(u, v) \in E$ is in the solution E', then all edges along $PATH(u, v)$ may be contracted without loss of generality.

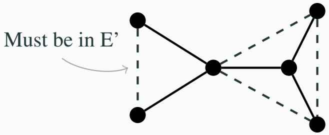


Figure 6: Reducing a TAP by contraction

Contracting Covered Edges

If it is known that an edge $(u, v) \in E$ is in the solution E', then all edges along $PATH(u, v)$ may be contracted without loss of generality.

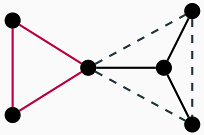


Figure 6: Reducing a TAP by contraction

Contracting Covered Edges

If it is known that an edge $(u, v) \in E$ is in the solution E', then all edges along $PATH(u, v)$ may be contracted without loss of generality.

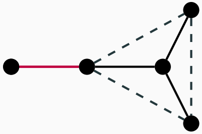


Figure 6: Reducing a TAP by contraction

Contracting Covered Edges

If it is known that an edge $(u, v) \in E$ is in the solution E', then all edges along $PATH(u, v)$ may be contracted without loss of generality.

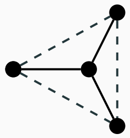
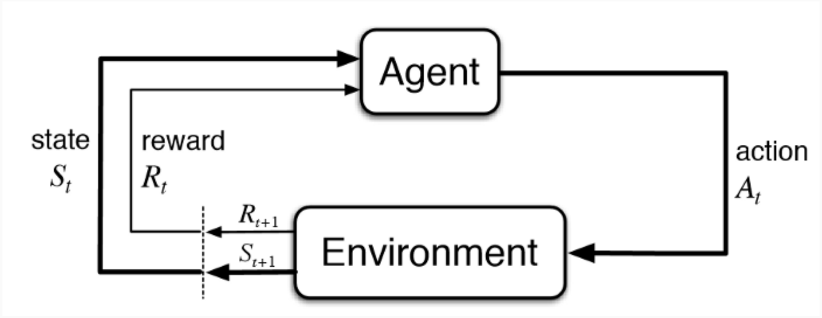


Figure 6: Reducing a TAP by contraction

Reinforcement Learning Overview

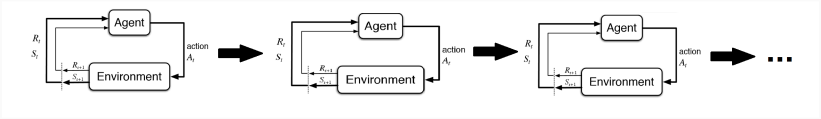


I created my environment from scratch in Python, and the models are from Stable Baselines 3 which is a fork of OpenAI's Baselines library.

The models that I'm using are Proximal Policy Optimization (PPO) and Maskable PPO.

Reinforcement Learning Overview

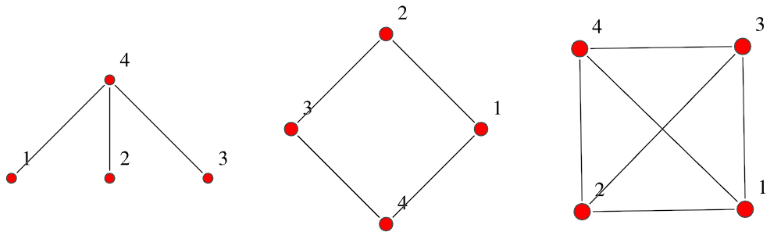
The agent will be continually trained on randomized environments until it learns to solve the problem (by maximizing reward)



Observation Space

The agent must have data from the environment so that it may make informed decisions. This input is called an *observation* and the structure of the input is called the *observation space*

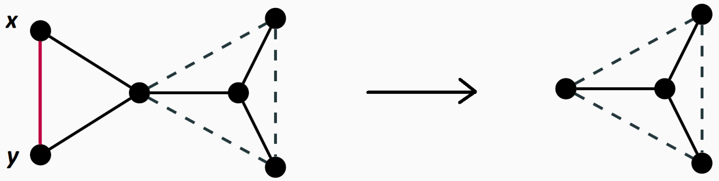
For my observations, I am currently using an adjacency matrix of both the tree and the edgeset. This is equivalent to a 2 channel binary image.



$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

Actions and State Changes

Actions in the problem can be thought of as a tuple of vertices $(v1, v2)$ which corresponds to an edge in the graph. Taking an action affects the environment for the next iteration



In this case, the action chosen by the model was (x, y).

By continually selecting one edge at a time and contracting, we can ensure that our model arrives at a valid 2-connected solution that satisfies the TAP.

Action Spaces

Actions are represented as tuples, But the way tuples are chosen may be done in more than one way.

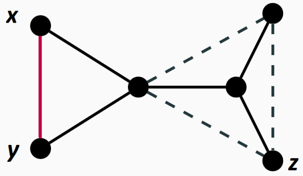
Discrete action space Select a single action $\alpha \in S$, where
$S = \{(1,1),(1,2),(1,3),(1,4),(2,1),(2,2),(2,3),(2,4),(3,1),(3,2),(3,3),...\}$

multi-discrete action space: Select an action (α,β) such that $\alpha,\beta,\in S$, where $S = \{1,2,3,4\}$

The multi-discrete action space is much more natural to the problem, and as you will see, performs better in practice.

Note that the agent may select an action that corresponds to an invalid edge.

Reward and Invalid actions



The action (y, z) is not in the valid edge-set, and should be avoided by the agent. The agent will learn to avoid invalid actions on its own if they give a negative reward, which is also known as a punishment.

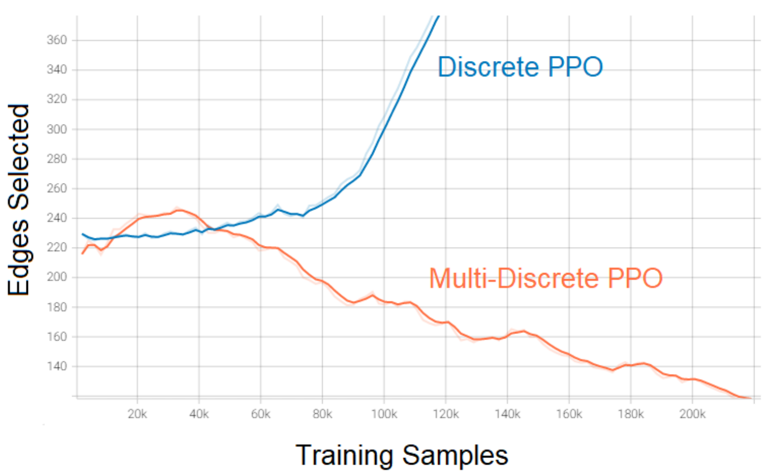
I believe the optimal reward function is a constant negative reward for selecting *any* edge.

This reward function will also make the agent avoid invalid actions given that they do not reduce the environment.

Results

The following results are from randomized edgesets and trees with 100 vertices and a density of .5.

Note that randomly selecting *valid* edges yields an average solution size of 52.7*edges*



Invalid Action Masking

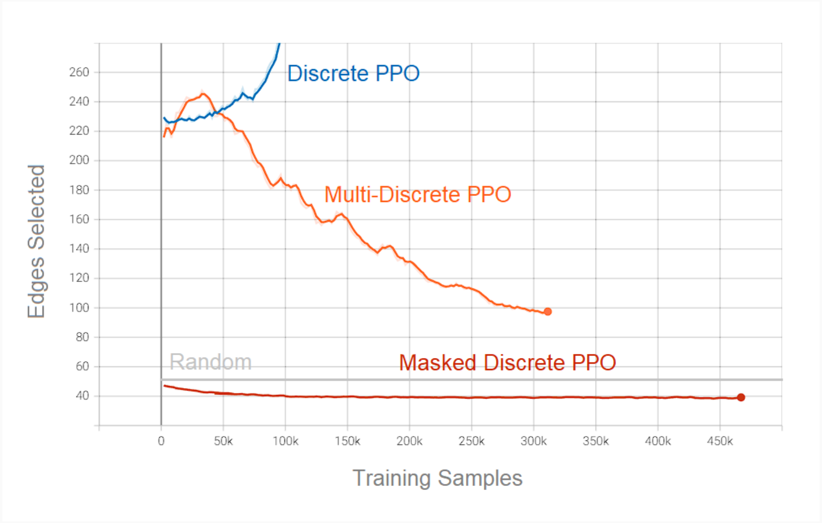
Papers have been published within the last few years which increase the performance of RL algorithms by restricting the agent from making invalid actions.

This is called *action masking*.

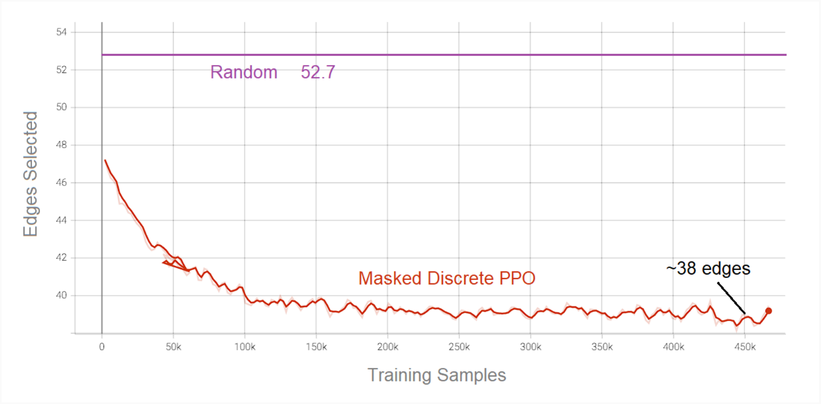
For a graph with four vertices the discrete action space, the available is as follows,
$S = \{(1,1),(1,2),(1,3),(1,4),(2,1),(2,2),(2,3),(2,4),(3,1),(3,2),(3,3),(3,4),(4,1)...\}$

But for a emphmasked discrete action space, the action space depends on current environment. For an edgeset of size 5, the available actions could look like this,
$S = \{(1,3),(1,4),(2,4),(3,4),(4,2)\}$

Action Masking Results



Action Masking Results



Ideally, we would compare the agent's performance to exact solutions, but I do not have an exact solver.

Another dataset lists similar sized graphs lists their random as 38, and exact as 31.

The problem is that their dataset is on different graph types like caterpillars and star-like graphs, which have smaller solutions.

Action Masking Results

I don’t yet have an exact solution solver, but this will be my next step.

I am confident that the agent is performing well relative to the optimal, but I’m not sure how well.

I have aggregate data of different kinds of random graphs with the same vertex count and density, but this isn’t a good comparison. The graphs used for this data are stars, caterpillars, forests,
etc. Some of these graphs will have very different solution sizes than to the graphs that I generated

This data shows the random edge count to be ≈ 38 edges and the exact edge count to be ≈ 31.

Future Work

The immediate next step is to find a value for exact solutions so that the true performance of the learning agent may be evaluated.

The environment and agent could still be tweaked in several ways. Most notably adding additional information to the observation space.

I do not think that a better model than Maskable PPO is available, as PPO is already state of the art and masking has huge performance benefits. However, there is a need for an improved Multi-Discrete action masking, as it has limitations that prevent it from fitting environment.

Questions?