CSC520 Fall 2024 Assignment 2

Due Sep 24^{th} at 11:59pm

This assignment consists of four questions which involve written answers and code. In order to complete the assignment you must submit a written report in pdf form detailing your answers to the questions as well as your code. As discussed in class all work *must* be your own. You may not use third party libraries or example code to complete the assignment with the exception of csv file loaders. All reports must be clear and well written. All code must be clear, readable, and well-commented. Upload your sourcecode and report in a zip called `Assign2-<UNITYID>.zip`.

Question 1: Knowledge Problems (10 pts)

Identify two real-world knowledge-based problems one of which is sufficient for constraint representation and *does not* require logic the other of which *does*. Provide a brief description of the two problems and a sample constraint graph or knowledge base. Justify your choices.

Question 2: First-Order Predicate Logic (15 points)

Consider the following English sentences.

- Some dogs are pets.
- Every pet has an owner.
- Not every dog likes to play.
- Any dog that likes to play, plays with at least one toy.

1. (5 pts) Convert the sentences into first-order predicate logic. Be extremely careful about quantification; not everything in the universe may be a dog, a pet, an owner, or a toy. Use the following lexicon:

   dog(X) X is a dog.

   pet(X) X is a pet.

   owner(O, P) O is the owner of P.

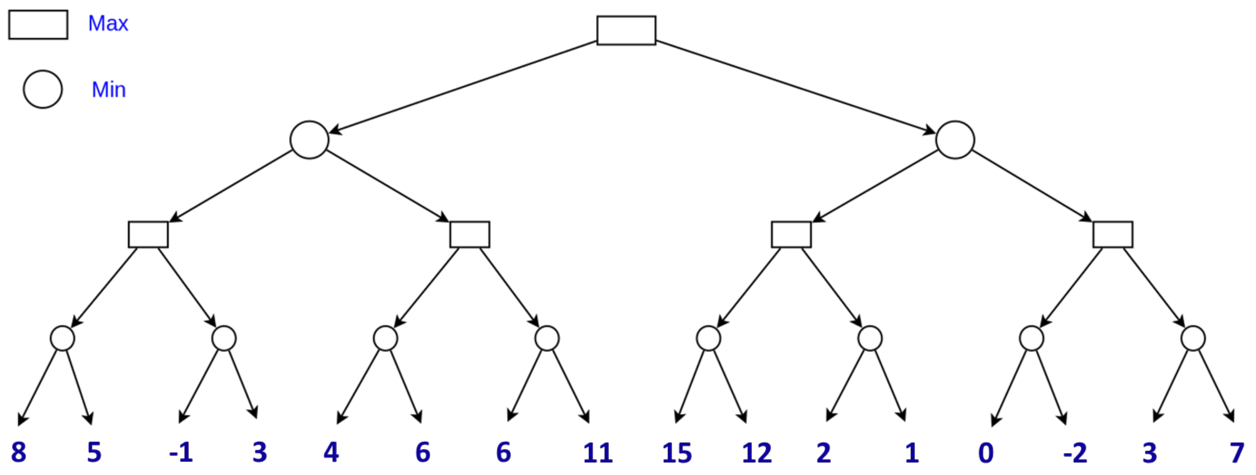
   likes_to_play(X) X likes to play.

   plays_with(X, T) X plays with toy T.

2. (5 pts) Following the steps on text book to convert the logic statements into CNF.

3. (5 pts) Using resolution prove using FOPL resolution that ”If a dog is a pet, it either likes to play or has an owner.” Number your clauses, and indicate explicitly step-by-step what resolves together, under what substitution.

Question 3: Adversarial Search (10 pts)



Consider the adversarial diagram above. Using the diagram answer the following questions:

1. Show the values that would be returned for each cell using min/max search.
2. Carry out $\alpha - \beta$ pruning, showing the thresholds, pruning steps with type, and returned values for each step. Number your steps to show the order of operations.
3. If you could shuffle the move order can you increase the number of items pruned? If so show such a change.

Question 4: Bandits (65 pts)

For this assignment, we will be exploring the concept of networked bandits to optimize content distribution across a social media network. Your task is to design and implement a networked bandit library that supports two of the three types of bandits discussed in class, and two types of reward values, individual engagement, and influence effects. Your goal is to determine which user provides the highest cumulative reward for the content exposure, i.e. who is a better client.

You have been given two datafiles, the first `content_engagements.csv` represents user engagements with the content. Each row is a presentation opportunity or a piece of content. Each column represents an individual users. The cells in each row give an engagement or preference score for each user per item. You use this file to calculate the direct score for each user.

The second file `user_network_connections.csv` describes a influence network relationship between the users. The first column provides a list of user ids while the second lists IDs who follow, i.e. consume content from, the user. When used in combination with the initial file this can be used to calculate the network, benefits of the content where, for a given user you receive a reward of 1/2 of the value of each follower's engagement score, but nothing for the user.

Your code will be called as follows:

```
java -jar Bandit.jar <alg> <reward> <range> <min> <step> <max> <engagement_file> <network_file>

python Bandit.py <alg> <reward> <range> <min> <step> <max> <engagement_file> <network_file>
```

Where:

- `alg` is one of "Static" "Rolling" or "Recency".
- `reward` is one if "Individual" or "Influence"
- `range` is the number of rows used for parameter fitting.

- `min` is the minimum parameter value for your grid search (e.g. 0)
- `step` is the step size for your grid search (e.g. 0.2)
- `max` is the maximum parameter value for your grid search (e.g. 0.9)
- `engagement_file`: A file containing data for the users and their engagement scores for each content (row).
- `network_file`: A file representing the network topology (connections between users).

On execution your code will load the datafiles and generate a set of bandits of the specified type. You will then iterate over *range* rows of the engagement file using grid search to find the best parameter values between *min* and *max*, where the parameter is set by cumulative total reward. Your code will then use those specified parameter values to iterate over the *entire* file to calculate a final reward score. On each new execution you will display the initial parameters. On each step you will print out the choice made, the current reward, and the cumulative reward. On close it will print out the relative number of pulls for each choice and the best performing user. You are advised to test your code on a subset of the data before using the full files.

In addition to the provided files you should define two separate networks (manually defined) which cover the same users. Along with your code and your new datafiles you should submit a brief report describing:

1. The best performing agent type for each reward with it's parameters and cumulative scores for the provided file.
2. The best performing agent types, parameters, and cumulative scores for your user-defined files with the network
3. A discussion of how the graph structure affected the model choice and parameters.

As always your code should be clear, readable, and well documented, and it should include a README file that explains how to build and execute it on the servers.

Note: for the purposes of this assignment you may make use of the Apache Commons CSV library, or the built in python csv library in your code, as well as standard graph libraries such as networkx.