University of Alberta
Computing Science Department
XAI in Games - W20

Assignment 2
20 Marks
Due date: 20/03 at 23:59

Overview

In this assignment you will implement the Metropolis-Hastings (MH) algorithm for synthesizing scripts for Can't Stop.[1] It is fine to discuss the assignment with your classmates, but you shouldn't share your code with them. We will use the forum on eClass to discuss the assignment. As a reminder, 5% of your marks are from class participation, which also includes being active in the forum, asking and answering questions.

I suggest you finish reading this document before starting to implement the solution for the assignment. The marks below sum up to 100, which will then be converted to 20 marks for you final grade in the course.

Imitating UCT's Strategy with Program Synthesis (40 marks)

One of the main goals of this course is to understand algorithms able to derive interpretable and explainable strategies for games. Computer and board games are excellent testbeds for these algorithms because they were designed to be easy to understand and to be challenging and interesting to humans. In this assignment you will implement the MH algorithm for producing a script for mimicking the strategy derived by UCT. The UCT code is provided for this assignment. The UCT implementation I am providing works with the codebase from the first assignment.[2] As in the first assignment, we will use a smaller version of Can't Stop to allow you to run your experiments in your own laptop or home computer. Since the game is small, we should expect UCT with a large number of simulations to find strong strategies for playing the game. You will then use MH to find a strategy in the space of strategies defined by the DSL and the sketch used in the first assignment that approximates the strategy derived by UCT.

The MH algorithm receives a training dataset of input-output pairs (state-action pairs in the context of this assignment) and produces a script that attempts to map each of the states to its corresponding action in the training set. Here, the states will be collected by having UCT play against it self a number of Can't Stop matches (e.g., 20 matches of self play). The corresponding action in each state s is the action chosen by UCT at s after searching with a fixed number of simulations (e.g., 100 simulations).

Instantiating and Running UCT

UCT can be instantiated as shown in the code below.

```
uct = Vanilla_UCT(c, simulations)
```

Here, `c` is the exploration constant of UCT ($c = 1$ worked well in this task) and `simulations` is the number of simulations performed by UCT. Larger values of `simulations` will tend to produce better strategies for playing the game (e.g., 100). Once instantiated, you can use `uct` as a player. For example,

```
chosen_play = uct.get_action(state)
```

[1] See lecture 13 for a detailed description of the Metropolis-Hastings algorithm.
[2] See lecture 6 for a detailed description of UCT.

invokes UCT for the game state named `state`, which returns an action `chosen_play` to be played at `state`.

Instantiating Can't Stop

An instance of the class Game can be created for the smaller version of Can't Stop as follows.

```
game = Game(n_players = 2, dice_number = 4, dice_value = 3,
column_range = [2, 6], offset = 2, initial_height = 1)
```

This is the same game used in Assignment 1. Feel free to experiment with other versions of the game.

DSL and Sketch

You will use the same the DSL provided for Assignment 1. Feel free to experiment with other DSLs. The sketch we will use is also the same one used in Assignment 1, which I reproduce here for your convenience.

```
from players.player import Player
from players.scripts.DSL import DSL

class PlayerTest(Player):

    def get_action(self, state):
        actions = state.available_moves()

        for a in actions:
            if {CODE TO BE DEFINED BY THE SYNTHESIZER}:
                return a
        return actions[0]
```

Proposal Function

The MH algorithm requires the implementation of a proposal function from which one obtains samples. The proposal function should allow to you to sample scripts "in the neighborhood" of the current sample. One way of achieving this is by representing the scripts as a tree according to the DSL, as in the example from lecture 13. The tree can then be used to perform "small" (as well as "big") changes to the current sample. Due to the structure of the tree, if you randomly choose a node of the tree to modify the current sample, then it is more likely that you will perform a "small modification" (e.g., change the value of a parameter in a DSL function), than a "big modification" (e.g., change the if-clause completely). This is because the tree will tend to have more leaf nodes than internal nodes. This is desired as it will allow you to focus on scripts that are in promising regions of the search space (i.e., regions of scripts encoding better strategies). But it will also allow you to eventually escape from unpromising regions of the space.

The tree is unlikely to be a symmetric proposal function. However, you may implement the MH formula as if the function was symmetric (i.e., $q(x_i|x_{i-1}) = q(x_{i-1}|x_i)$). This simplification will not interfere much in the results. This is because the tree as a proposal function will often be near symmetric and computing the transition probabilities $q(x_i|x_{i-1})$ and $q(x_{i-1}|x_i)$ can be cumbersome. See lecture 13 for details on the transition probability q and on the symmetry property of proposal functions.

Experiments (25 Marks)

You will run your implementation of the MH algorithm to iteratively generate one if-rule for the sketch shown above. Next, I will describe this iterative procedure.

First, you will generate a training set by having UCT play a few matches against itself. Let's say that this process produces a training set of size M_1. After running the MH algorithm to derive an approximation of a single if-rule that best represents the strategy played by UCT, which we will denote as r_1, you will remove from the training set all state-action pairs for which r_1 agrees with UCT. You are now left with a training set of size $M_2 < M_1$. You will then repeat the process of running MH to find a second rule r_2 that best matches the remaining training instances. Repeat this process until you have generated 5 rules.

Repeat this experiment a few times, trying to observe if rules r_1, r_2, r_3, r_4, r_5 are similar across multiple independent runs of the algorithm. If you observe that the rules are completely different every time you run the algorithm, then you might need to change the parameter values used in your search. For example, you might have to increase the number of UCT matches played or its number of simulations. Another parameter that influences the results is the number of iterations used by the MH algorithm. In my experiments I found that 1,000 MH iterations tend to be enough to consistently observe similar results in independent runs.

Report (35 Marks)

The report is worth 35 marks. However, if you do not deliver the report you will not earn marks for the earlier parts of the assignment. The report will be submitted through eClass by 20/03 at 23:59. The report, which must be in pdf format, should contain:

1. A link to your code in GitHub. Alternatively, you can upload a zip file with your code onto eClass.
2. A representative set of rules $r_1, \cdots, r_5$ encountered by your system.
3. A discussion relating the rules encountered in Assignment 1 with those encountered in this assignment.
   (a) Are the rules found in the two assignments similar? All 5 rules or only a subset of them?
   (b) If the rules found in the two assignments are identical and if we assume that they form an evolutionarily stable strategy in the context of the experiment of Assignment 1, can we state that the strategy found by the MH algorithm is a Nash equilibrium for the smaller version of Can't Stop? If not, the strategy found by the MH algorithm is a Nash equilibrium for which game?
   (c) What are the advantages and the disadvantages of the evolutionary approach studied in the first assignment and the imitation learning approach studied in this assignment? Your discussion should mention the running time of the algorithms, their requirements, and possible guarantees.
4. A discussion about the rules $r_2, \cdots, r_5$. Are they providing important information about the strategy played by UCT? How would you measure the usefulness of the rules derived by the MH algorithm?