Report: A twist of Faith

CSE 415: Introduction to Artificial Intelligence

Winter 2019

Stephan Hagel

Student Number: 1870562

sthagel@uw.edu

Palash Roychowdhury

Student Number: 1725115

palashrc@uw.edu

1 The Project

Our project is titled "A twist of Faith". We chose option number two, reinforcement learning for the Rubik's cube. Since the state space of a Rubik's cube is very big ($N \approx 4.3 \cdot 10^{19}$) and there is only one goal state, a naive approach is not feasible. Instead we are using a method called *Curriculum learning*. The idea behind curriculum learning is to explore the state space step by step going backwards from the goal state. To be more precise, we start of with a state, which is one move away from the goal state.

The agent learns how to solve this state over multiple episodes to learn the Q-values of this state and other states close by. After a certain number of episodes, we let the agent start from a different state, which is also one move away from the goal state and repeat the procedure. After this has been done for a reasonable number of different start states, we move on to states which are two moves away from the goal. The same procedure is repeated on those states. Then we go on to states, which are three moves away from the goal and so on.

Theoretically, we have to repeat this procedure N_G times, where N_G is the *God's number* of the Rubik's cube. This number is defined as the maximum number of moves needed, in which a Rubik's cube can be solved in from any starting state. For a $3 \times 3 \times 3$ Rubik's cube the God's number is 26, for the $2 \times 2 \times 2$ version it is 14. Going all the way up to that number would still result in a combinatorial explosion, but depending on the size of the cube we can go as far as 5 or even more moves away from the goal in a reasonable amount of time. We can let the algorithm run overnight which could result in searching state spaces which are about 7 moves away from the solution.

2 The Team

This project has been done by Stephan Hagel and Palash Roychowdhury. Palash came up with the idea of using Curriculum Learning and was mainly involved in writing the problem formulation and making it usable for both a $2\times2\times2$ and a $3\times3\times3$ cube. Stephan focused on implementing the Q-learning algorithm and implementing the Curriculum Learning. We both worked together very closely though and helped each other with problems we had, so both are equally important in making both parts work together.

3 How to run the code?

The code consists of two main parts: Cubes.py and run.py. The file Cubes.py contains the problem formulation for the Rubik's cube, so that it can be used with the Q-Learning algorithm. It is written similar to the problem formulations used in homework assignment 2. The file run.py is used to actually run the Q-Learning algorithm. When it is executed, the user can choose to either use default values for several parameters or to input custom ones. This is done via the standard input (stdin). The execution of the program consists of two main parts: The first part is to learn the Q-Values via Q-Learning with Curriculum learning. This is done with minimal output to speed up the process. After the agent did the Q-Learning for the number of times given as parameters, the learned model will be tested with a given start state. As of right now the state, at which the model is tested, is hard-coded into the program but can easily be changed.

4 Sample transcript

The following two screenshots (on next page) show an excerpt of the program starting to run and when it hits the goal state in some episodes.

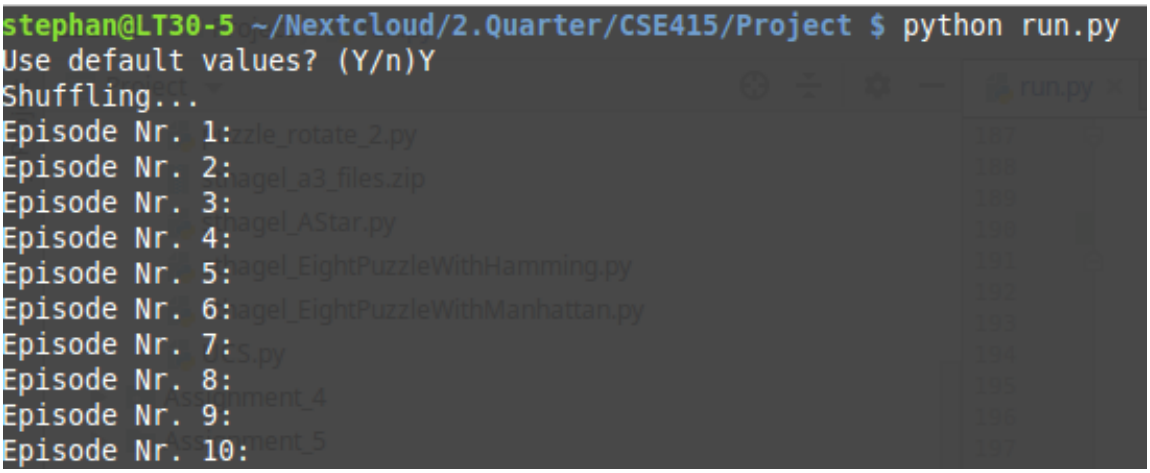

Figure 1: Running the program.

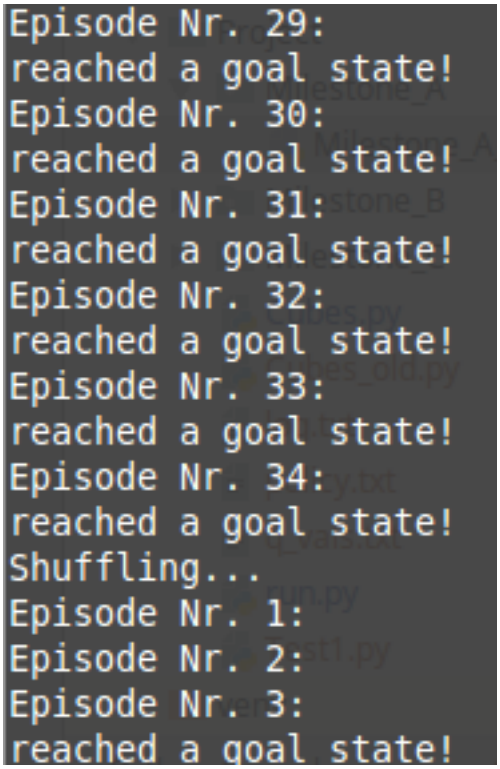

Figure 2: Hitting the goal

5 Code excerpts

The first interesting part of our code is how we actually implemented the curriculum learning. We did this by generating a solved cube for each iteration and applying n random actions to it, where n is the number of the current iteration. In code this looks like this:

```
for current_max in range(MAX_MOVES):
   num_steps = 12 * (current_max + 1)
   for k in range(num_steps):
      # Initializing a solved cube
      cube = Cubes.State(Cubes.GOAL_STATE_THREE)
      # Shuffling the cube
      for j in range(current_max + 1):
         rm = random_move()
         cube = Cubes.move(cube, rm[0], rm[1])
      if cube not in KNOWN_STATES:
         KNOWN_STATES.append(cube)
      eps = N_EPISODES * (current_max + 1)
      # Do the actual Q-Learning
      qlearn(cube, current_max + 1, DISCOUNT, EPSILON, ALPHA, MAX_ITER +
current_max, eps)
```

The `random_move()` function generates a random move, which can be applied to the cube. Inside the `qlearn()` function we run the Q-Learning algorithm a certain number of episodes, which increases with the number of scrambles done to the cube.

We also count the number of episodes, which reached the goal state. If this number is big enough, we consider this episode as done. This is realized by the loop:

```
while goal_counter < 10 * currmax and episode < eps:
   ...
```

To choose which move to apply we used the ε-greedy algorithm:

```
chance = random.uniform(0.0, 1.0)
if chance > epsilon:
    Choose an action according to the policy
   if s in POLICY:
      act = apply_policy(s)
   else:
      act, actA = random.choice(list(ACTIONS.items()))
      del actA
   else:
      act, actA = random.choice(list(ACTIONS.items()))
      del actA
new_state = ACTIONS[act].apply(s)
KNOWN_STATES.append(new_state)
```

Most of implementation for the policy and q values could be taken from the value iteration part of Assignment 5, but we found out it is much faster to only update the Q-Values of states that have actually been visited. This is done by the `update.policy` function. The Q-Values are updated via

```
rew = Cubes.reward(new_state, SIZE, LIVING_COST)
sample = rew + discount * maxQ
# Update the Q value for the current state
Q_VALUES[(s, act)] = (1.0 - alpha) * Q_VALUES[(s, act)] + alpha * sample
update_policy(s, KNOWN_STATES, ACTIONS)
```

If we hit the goal state, we also decrease the learning rate to achieve a faster convergence.

```
if fabs(rew - 1.0) < 1e-7:
   goal_counter += 1
if not goal_counter == 0:
   alpha /= goal_counter
```

6 What did we learn during the project?

6.1 Stephan

I used this project mainly to catch up on things I missed by not doing all of the extra credit homework in older assignments, such as the Rubik's cube task and the Q-Learning task. I mainly learned how to implement Q-Learning and at what points it can be improved and optimized. This project also reminded me on how drastic the combinatorial explosion can be, especially for the 3^3 cube. It was also a nice exercise on how to tackle programming assignments as a team compared to doing it on your own. While helping Palash doing the bugfixing for the 2^3 cube I also learned, how important it is to write portable code, which can easily be generalized. A good counterexample for this though is the fact, that I was able to reuse a considerable

amount of code from homework assignment 5.

6.2 Palash

While formulating the problem, it was a challenge to form a digital cube which could be transformed whichever way our operators wanted to transform it. Making functions which would transform the cube when it is rotated from any direction in any way for both a 3x3x3 and a 2x2x2 was a challenge. A lot of mistakes were made initially while writing a code that would work for both a 2x2x2 and a 3x3x3. Operators were made in such away that the size of the cube wouldn't matter. Various different styles of problem formulation were tried. At first we tried to use a model in which we were using tiles which would be represented on cubies of the Rubik's Cube. This was more of a 3D model which would then be modified using the operators. However, we found it difficult to modify such a model and do computations efficiently and quickly on it. Which is why we later changed our problem formulation and changed the model of the Rubik's Cube from a 3D model to a 2D matrix which can be modified much more easily and in a faster way.

Our initial thought process of implementing the Curriculum Learning for this project was to formulate the problem and then let the algorithm train starting from 1 move away from the solution all the way to the God's number of 26 moves away from the solution. This would have made sure that the whole state space was searched for the 3x3x3 Rubik's Cube. However, since the complete state space of the Rubik's cube is more than 14 Quintilian we thought that we might be able to reach the God's number for a 2x2x2 Rubik's cube which is more than 3.6 Million. Reaching all these states once was possible (taking days of computation) but doing it over and over again to iterate the Q-values resulted in multiplying the computation many folds

rendering it impossible for our computers to handle in decent time. Therefore, we decided the reduce the state space which was a major decision and learning point for us.

7 Further ideas for the project

We are working on saving the q values to an external file which can be loaded in later so that we can simply load the Q-values to a new session and can continue training the algorithm with more number of moves. This way we won't have to train the algorithm every time we want to find solution to a shuffled Rubik's Cube.

Right now we need to run the whole program for every new session session. This would be avoided if we train the program once and save all the Q-values in an external file which can be loaded directly when a new session is started without training the algorithm again. This would save most of the time that it takes to train the algorithm. This can be done by hashing all the Q-values and then storing them.

Other AI techniques like feature based learning can be implemented as well, using which the AI can learn to recognize some patterns when trying to solve a Rubik's Cube. When a cube is getting close to solution, some patterns emerge like forming "blocks". A block is formed when 2 or more pieces form a single continuous color block with all the pieces in the correct position relative to each other. A more complex block can constitute more than one color with as much as half the number of total cubies of a Rubik's Cube. These blocks can be recognized by the algorithm as features and it can try to make such blocks which will ultimately help in generating a better solution.

We also plan to parallelize the algorithm so that we can use a GPU to do these computation. A GPU has thousands of cores which can perform such tasks with

much faster time. Using such a parallelized program we might be able to train the algorithm to a state space which are 10 moves away from the solution. It would require quite a bit of effort to parallelize the code as Q-learning does require some computation that needs to be done in serial. We need to look into this more

8 References

Our main reference for this project was the report of "Deep Q-Learning for Rubik's Cube", by Etienne Simon and Eloi Zablocki (URL: https://github.com/EloiZ/DeepCube). In this report the authors explain how to use curriculum learning for a Rubik's cube and give a nice demonstration, how it speeds up the learning process. The implementation they use uses a deep learning approach with the Theano package though. Therefore their implementation could not be used for our purposes.

9 Partners' Reflections

9.1 Stephan Hagel

The roles of the project can be found in the "Team" section. The main challenge was to sync up your work with your Partner's, so that the whole code works together. Also finding a reasonable way to share the work can be quite difficult. Since Palash and I are roommates, it was quite easy though to communicate with him and arrange meetings. This made work way easier. It was also a nice exercise on how to tackle programming assignments as a team compared to doing it on your own.

9.2 Palash Roychowdhury

Main role was to formulate the problem and to come up with the Q-Learning with Curriculum Learning idea. Changed the problem formulation multiple times to fit the needs to the programming that we were doing and the computation.