Example Notebooks

aimacode

About Me

Name : Rakesh Kumar
University Name : Indian Institute of Technology, Palakkad
University Location : Palakkad, India
University Website : https://iitpkd.ac.in
Field Of Study : Computer Science and Engineering
Degree Currently Being Pursued : Undergraduate
E-mail : thisisrakeshnotchintu@gmail.com, iamrakesh28@rocketmail.com
Github : https://github.com/iamrakesh28
CodeChef : https://www.codechef.com/users/iamrakesh28
Time zone : India Standard Time (GMT+5:30)

I am an undergraduate student pursuing Bachelor of Technology in Computer Science and Engineering at Indian Institute of Technology, Palakkad. My main areas of interest are Artificial Intelligence and Algorithms (Competitive Programming). I am good in Python and C/C++.

Courses

- Machine Learning (Stanford University) on Coursera
- Neural Networks and Deep Learning (deeplearning.ai) on Coursera
- Google Cloud Platform Fundamentals: Core Infrastructure (Google Cloud) on Coursera.
- Introduction to Algorithms
- Design and Analysis of Algorithms
- Doing Artificial Intelligence course in this semester

Abstract

Develop example worked projects using Jupyter/IPython or similar mechanisms to demonstrate the use of the algorithms from aima-pseudocode to solve interesting problems.
Examples projects can include game playing, natural language processing, machine learning, and other topics.

Project Proposal

Most probably, I will be using IPython to demonstrate the examples. In some chapters, there are several algorithms which solve the thing but their running time differs. Some works good in some problems while others in different problem. For example, In Chapter 3-4 (Problem Solving), there are algorithms which uses heuristics to solve while BFS doesn't. If the heuristic is good, then it improves the performance. But there are some problems where finding a good heuristic is not easy, so we have to use classical algorithms. I will develop 2-3 examples to demonstrate and compare the algorithms which solve similar problem. Similarly, playing mini-chess optimally can have two algorithms (simple Adversarial Search and Alpha-Beta Pruning). While solving puzzles, playing chess, etc. can have many states (n- puzzle can have at most $n^2!$ states). So, these algorithms can be used to solve small states problems (about 10^7). For storing and accessing these states efficiently, Python Dictionary can be used. I have not studied Chapters 6-15 and Chapter 22 from the book. I will try to study these chapters in the first month. Value Iteration, Policy Iteration,etc can be used in Tetris, gridworld, stochastic robot walk,etc. In Reinforcement Learning, I can train on some simple games like Dino Run (Google Chrome game), Snake game or Pacman. Previously, I used Q-Learning to train on small board Pacman (link in my project section). Deep Neural Network can be used in classification, prediction or even in Deep Q-Learning. Below are two examples and the way how I'm going to implement these two examples.

1. **Q-Learning on Dino Run (Google Chrome game)**

   First, I will build a simple game with basic 2D graphics in python. The game display will be small (20x10). My states would include 20 boolean variables indicating which positions have cactus. The cacti sizes won't differ much (about 1-2 units). These constraints are here as I don't want total possible states to go beyond than 10^7. My actions will be to jump or not. Game training will begin at the starting game states and will continue till last. Reward will be based on how the player scores. After around 1000 episodes training, the Q-values may converges to optimal values (If not then more training will be required). The game can be modified such that it will have coins in the way and collecting them will increase our score. Training using the above algorithm will give an optimal actions at each state.

2. **Playing Chess using Alpha - beta pruning**

   Minimax algorithm can be also be used but Alpha - beta pruning reduces the number of nodes in the search tree. Each chess board configuration can lead to various possible states depending upon the players move. We will start from initial board configuration, we will perform the above algorithm upto 5-6 levels depth in the search tree and then will

use some heuristic (such as, more is the number of important pieces, more is chances to win) to decide which move is better. We will play that move. If we encounter any unvisited state, we again use the above algorithm and play. We will continue doing this until the game ends.

Proposal Timeline

April 28 – May 27 (Before the official coding time):

- In the first week, I will study the Chapter 4 (Search in Complex Environments) and Chapter 6 (Constraint Satisfaction Problems) from the book. I will then implement all algorithms from these chapters given in the pseudocode.
- In the second week, I will cover Chapter 7 (Logical Agents) and Chapter 8-9 (First Order Logic).
- In the third week, I will study Chapter 11 (Automated Planning) and Chapter 13 (Probabilistic Reasoning) and implement the algorithms.
- In the last week, I will cover Deep Neural Networks and Natural Language Processing chapters from the book.
- During this period I will remain in constant touch with my mentor and the Mailman community. I will remain active on IRC and Mailing lists to discuss and finalize my plan. If there will be any need to change my order of studying chapters or any modification on the implementation, then I will change it.
- Thus with the help of my mentor I will become absolutely clear about my future goals.

May 27 – June 24 (Official coding period starts):

- Solve games like Move the block, n - Puzzle and 2x2x2 Rubik's cube using BFS(one or two way), A* search, and other search algorithms from 3rd and 4th chapters. Examples for Chapter 2 (Intelligent Agents) and above games will be completed in the first 2 weeks.
- Solve search problems using different algorithms. I will either use linux shell or Jupyter Notebook to visualize the steps of the solution .
- Build Tic-Tac-Toe and Mini-Chess where the bot will play against human. These can be implemented using alpha-beta and monte-carlo-tree-search algorithms.
- Above two will be completed in last two weeks.

June 24 – June 28 (Phase 1 Evaluation):

- 4 days are for any kind of unpredictable delay

June 28 – July 22:

- In the first week, I will use examples to demonstrate the use of algorithms in Chapter 7 (Logical Agent).
- Chapter 9 (Inference in First-Order Logic) and Chapter 11 (Automatic Planning) will be covered in 2nd week.
- In the 3rd week, suitable examples will be used to cover Chapter 12 (Quantifying Uncertainty) and Chapter 13 (Probabilistic Reasoning).
- In last few days, I will cover Chapter 14 (Probabilistic Reasoning Over Time)

July 22 – July 26 (Phase 2 Evaluation):

- 4 days are for any kind of unpredictable delay

July 26 – August 19:

- First week will cover Chapter 15 (Making Simple Decisions) and Chapter 16 (Making Complex Decisions). Complex Decision problems (both stochastic and deterministic), will be solved using Value iteration, dynamic programming, etc and other algorithms from the chapters.
- 2nd Week will cover Chapter 17 (Making Decisions in Multiagent Environments) and Chapter 18 (Learning from Examples).
- In the remaining days, Deep Neural Networks, Reinforcement Learning, NLP and Robotics will be covered. Q-Learning will be used in small states games examples such as 6x6 Pacman, 5x5 Snake Game. I will either use Deep Neural Network in Q-Learning for larger state games or in some kind prediction or classification problem. NLP can used for parsing.

August 19 – August 26:

- Unfinished works will be completed within this week.
- Any further modification in the code will be done and any bug will be solved.

Concurrent Commitments in Summer

I have summer break of 3 months (28 April to 24 July). I will not be doing any other internship this summer. So, I can give 6-7 hours daily for GSoC. After the summer break, I will have classes in the morning (till 1 pm) but I can give 5-6 hours daily after the classes.

Past Projects (link)

- **Move the Brick :** It is a puzzle game, where the player has to move the red block to the exit. Other blocks are obstacles which may have to be moved to make room for the red

block. I have build two versions of it. One has only 1x1 square blocks (link) and other has 2x1 and 1x2 blocks (link). The game uses BFS algorithm to solve the puzzle in minimum steps. Each state is a configuration of the board.

- **Pacman(**link**) :** I coded classic Pacman Game in Python. The game uses search algorithm to find shortest path from ghosts to pacman. Details are there at the link.

- **Q-Learning on Pacman(**link**) :** I used q-learning algorithm to train the bot to play above pacman game. The board of the game is reduced to 7x9, such that the total possible states is around 10^6. Details are there at the link.