CS47100 Homework 2

Due date: 5 am, Monday, March 23rd

This homework will involve conceptual questions and programming exercises. Instructions at the end of the document present detail about how to turn in the assignment on Gradescope and turnin.

Constraint-Satisfaction-Problem (10 Pts)

1. (10 Pts) Cryptarithmetic problems are mathematical puzzles in which the digits are replaced by letters of the alphabets. The numbers per letters have to be different and no leading digit will be zero. For example: Consider the Cryptarithmetic problem :

$$\begin{array}{r} SEND \\ +MORE \\ \hline = MONEY \end{array}$$

   The solution is S = 9. E = 5, N = 6, D = 7, M =1 , O = 0, R =8, Y = 2, as

$$\begin{array}{r} 9567 \\ +1085 \\ \hline = 10652 \end{array}$$

   Now solve the Cryptarithmetic problem

$$\begin{array}{r} TO \\ +OUT \\ \hline = TOP \end{array}$$

   with added constraint that O is never prime.

   (a) (5 Pts) State the variables and constraints of the problem and the domain of the variables. Remember to introduce variables for addition carries

   (b) (5 Pts) Solve the problem by hand and show the steps, using the strategy of backtracking with forward checking and the Minimum Remaining Values and least-constraining-value heuristic.

Propositional Logic (30 Pts)

2. (12 pts) Decide whether each of the following sentence is valid, satisfiable or unsatisfiable. Valid means the sentence is true for all variable assignments, satisfiable means for at least one assignment it is true, unsatisfiable means there is no assignment for which the sentence is true. Here, *Sugar*, *Sweet*, *Sweetener* are Boolean variables, which take values of either true or false. Show support for your decisions using truth tables.

   (a) (2 Pts) $Sugar \Rightarrow Sweet$

(b) (2 Pts) $(Sugar \Rightarrow Sweet) \Rightarrow (\neg Sweet \Rightarrow \neg Sugar)$

(c) (2 Pts) $Sugar \vee Sweet \vee \neg Sweet$

(d) (2 Pts) $((Sugar \Rightarrow Sweet) \vee (Sweetener \Rightarrow Sweet)) \Rightarrow ((Sugar \wedge Sweetener) \Rightarrow Sweet)$

(e) (2 Pts) $(\neg\ Sugar \vee Sweet \vee (Sweet \Rightarrow Sugar))$

(f) (2 Pts) $(Sweet \wedge Sugar) \vee (\neg\ Sugar)$

3. (12 pts) You are given the following knowledge base:

$$\begin{aligned} D &\Rightarrow S \\ B \wedge \neg Q &\Rightarrow T \\ S &\Leftrightarrow \neg Q \\ B & \\ D \wedge T &\Rightarrow R \\ B &\Rightarrow D \end{aligned}$$

(a) (4 pts) Convert this knowledge base into horn form,

(b) (4 Pts) Prove R using Forward Chaining

(c) (4 Pts) Prove R using Backward Chaining

4. (6 Pts) Prove using propositional resolution that the following sentence entails D:

$$(B \vee \neg A) \wedge (A \vee S) \wedge (C \Rightarrow D) \wedge (\neg S \vee D) \wedge (\neg B \vee C)$$

First Order Logic (20 pts)

5. (12 pts) Inference via resolution.

(a) (4 pts) By listening to the workers of the Dunder-Mifflin Office, Michael learned the statements below. Convert these statements into First Order Logic using the predicates mentioned in table 1.

Predicates	**Meaning**
isSecurityOfficer(x,y)	x is the security officer of y
isHungry(x)	x is hungry
makeSoup(x)	x makes/made soup
isOutside(x,y)	x is outside the place y
isEmployee(x,y)	x is an employee of y
isInside(x,y)	x is inside the place y
getCold(x)	x has got/ gets cold
isSick(x)	x is sick

Table 1: Predicates to be used for Question 5

i. Dwight is the security officer of the Dunder-Mifflin.

ii. Dwight is not hungry.

iii. Dwight made soup.

iv. A security-officer is not supposed to go outside the office

v. No employee of Dunder-Mifflin gets cold inside the office.

vi. If Dwight makes soup, then he is hungry or sick.

vii. Only if Dwight is in cold, he gets sick.

Hint: for example, the claim "*All security officers of Dunder-Mifflin are inside the office*" can be expressed in first-order logic as

$$\forall x \ (\text{isSecurityOfficer}(x, \text{Dunder-Mifflin}) \implies \text{isInside}(x, \text{Dunder-Mifflin})).$$

(b) (4 pts) Convert the above statements into Conjunctive Normal Form.
Hint: the claim "*All security officers of Dunder-Mifflin are inside the office*" can be expressed as: $\text{isSecurityOfficer}(x, \text{Dunder-Mifflin}) \implies \text{isInside}(x, \text{Dunder-Mifflin})$.

(c) (4 pts) Using this information, Michael claims that Dwight broke the rules. Using resolution for first-order logic, prove that Michael is correct. You might need to add zero or more statements of common-sense knowledge in order to complete this proof.

6. (8 pts) Inference via backward chaining.

(a) (4 Pts) Write logical representations for the following sentences, suitable for use with Modus Ponens using the predicates enlisted in table 2. Modus Ponens rules state that if P implies Q and P is true, then Q is true too. In mathematical notations, $P \implies Q, P \models Q$.

Predicates	**Meaning**
isEmployee(x,y)	x is an employee of y
loves(x,y)	x loves y
hates(x,y)	x hates y

Table 2: Predicates for Question 6

i. Dwight, Jim, Pam and Angela are employees of Dunder-Mifflin.

ii. Dwight loves Angela.

iii. Pam loves Jim.

iv. Angela hates Pam.

v. Dwight hates any employee from Dunder-Mifflin whom is hated by the persons Dwight loves among the employees.

vi. If Angela hates an employee from Dunder-Mifflin, and that employee loves some employee from Dunder-Mifflin, Angela hates the later employee too.

(b) (4 Pts) For the query $\exists e$ *isEmployee(e, Dunder-Mifflin)* $\wedge$ *hates* (Dwight,e). How many solutions for e actually follow from your sentences? In other words, who are the employees from Dunder-Mifflin whom Dwight hates? Solve using the logical representations from 6(a), setting different values for the variables and checking them (which is called unification).

Programming Assignment (40 pts)

Note: For the programming assignments we will use the Pacman project designed for the course CS188 at UC Berkeley. `http://ai.berkeley.edu/multiagent.html`

Please remember that solutions to any assignment should be your own. Using other people solutions, within or outside Purdue goes against the course academic honesty policy. The TAs will be using code similarity measures to detect plagiarism cases when grading the assignment.

In this assignment, you will design agents for the classic multi-agent version of Pacman, including ghosts. Along the way, you will implement both alpha-beta Pruning and minimax search and try your hand at evaluation function design.

The code base has not changed much from the previous assignment, but please start with a fresh installation, rather than intermingling files from HW1. **Please work on the provided code- do not download code from the Berkeley site.**

As in HW1, this assignment includes an autograder for you to grade your answers on your machine.

- **Files you will need to edit**: You will fill in portions of `multiAgents.py` during the assignment. You should submit this file with your code and comments. Please do not change the other files in this distribution.
- **Files you might want to look at**:
  - `pacman.py`: The main file that runs Pacman games. It describes a Pacman `GameState` type, which you will use extensively in this project.
  - `game.py`: The logic behind how the Pacman world works. This file describes several supporting types like `AgentState`, `Agent`, `Direction`, and `Grid`.
  - `util.py`: Useful data structures for implementing search algorithms.
- The rest are supporting supporting files you can ignore

After downloading and uncompressing the code, you should be able to play a game of classic Pacman :

`python pacman.py`

Now, run the provided `ReflexAgent` in `multiAgents.py`:

`python pacman.py -p ReflexAgent`

Note that it plays quite poorly even on simple layouts:

`python pacman.py -p ReflexAgent -l testClassic`

Inspect its code (in multiAgents.py) and make sure you understand what it's doing.

TODO:

- Complete Project 2: Questions 1 (Reflex Agent), 2 (Minimax), 3 (Alpha Beta Pruning) and 5 (Evaluation function) described on the Berkeley site. Submit your modified versions of `multiAgents.py` for grading. Each question is worth 10 marks.

Submission Instructions:

Upload your answers to the **conceptual** questions (i.e. Question 1) as a pdf format file in gradescope: `https://www.gradescope.com/courses/81160`

- For your pdf file, use the naming convention `username_hw#.pdf`. For example, your TA with username *mmostafi* would name his pdf file for HW2 as `mmostafi_hw2.pdf`.
- To make grading easier, please start a new page in your pdf file for each subquestion. Hint: use a `\newpage` command in LaTeX after every question ends. For example, use a `\newpage` command after each of part (a)-(b) of Question 1.
- After uploading to gradescope, mark each page to identify which question is answered on the page. (Gradescope will facilitate this.)
- Follow the above convention and instruction for future homeworks as well.

After logging into data.cs.purdue.edu, please follow these steps to submit your **coding** assignment:

1. Make a directory named `username` and copy all of your files there.
2. While in the upper level directory (if the files are in `/homes/yexiang/hw2/yexiang`, go to `/homes/yexiang/hw2`), execute the following command:

   ```
   turnin -c cs471 -p HW2 username
   ```

   For example, your professor would use: `turnin -c cs471 -p HW2 yexiang` to submit his work. Keep in mind that old submissions are overwritten with new ones whenever you execute this command.

   You can verify the contents of your submission by executing the following command:

   ```
   turnin -v -c cs471 -p HW2
   ```

   Do not forget the -v flag here, as otherwise your submission would be replaced with an empty one.

Your submission should include the following files:

1. Your modified python file `multiAgents.py`.