Introduction to Intelligent, Cognitive, and Knowledge-Based Systems – Final Project

Date of submission – Project: 30.06.20 (23:55:00)

Presentation: 20.06.20 (23:55:00)

Introduction

The final project involves building and describing a complete general agent that operates in multiple domains and carry on a variety of tasks in these domains. Your agent will face the following challenges:

- Some of the domains will be deterministic, some will have probabilistic effects (though you will not know the probabilities in advance), some will have failures in action (where taking an action leaves the agent in the same state) - You have seen this in Assignment 3.
- Some tasks will have more than one goal. It is up to the agent to decide on how to prioritize them.
- Some domains will be transparent: your agent will have perfect perception of everything that is true in the environment. However, in others, features of the environment (e.g., fluent describing that food exists in a location) will only be revealed once the agent fulfils some conditions (e.g., it is standing in the same place as the food). You have seen this in Assignment 4.

You are required to submit the source code for the agent per the instructions below, and to present the agent in the two last classes of the course (June 21 and June 28). A written report on the agent is due together with the submission of the code.

Requirements

In each evaluation, as in the exercises, your agent will receive a domain description and task description in PDDL. It will then begin interacting with the simulation in order to carryout the task (achieve its goals). We will evaluate your agent by monitoring its CPU time spent in reasoning, the number of actions it carries out, and its success or failure in achieving goals. We will run your agent in two phases:

1. Learning phase – this is when your agent MAY use learning to determine the best policy for a given problem, (e.g., as in Q-Learning), or to learn a model of the domain (e.g., as in Rmax Reinforcement Learning). We will run the agent enough times to run in this phase (so it could learn).

2. Execution phase – this is the phase where your agent only uses the policy it derived in the previous phase.

You will be given a few examples to practice on and are welcome to invent more to test your agent.

The use of learning is OPTIONAL and not required to succeed in carrying out the tasks. However, naturally, it may improve performance. During the learning phase, we will use the following command line to run your agent:

python my_executive.py -L <domain_file> <problem_file>

* If you decide **not** to use learning, your agent should recognize the -L flag and exit immediately with the exit code (128). In python:

```
import os
exit(128)
```

This will tell our system that you have decided not to use learning, so your agent will not be penalized for this.

Learning Agent

If your agent utilizes learning, the above command line will be run automatically again and again, with every single run ending after the agent has reached the goal, or a timeout (which we define externally) is reached. It is up to your code to decide on how to save information from one run to the next. We recommend using a file whose name makes use of the domain name (specified in the domain file), and the problem name (specified in the problem file). Make sure that files that pertain to different domains and/or problems are distinguished, and only use files in the same folder of your agent code.

One difficulty we have found in the past is in learning the same environment, but with different goals, for example different food locations within the same maze, which itself is an instance of the maze domain. The problem is with how PDDL works: it distinguishes a domain (actions that can be carried out) from problem (initial state of agent in the environment and goal). It really should distinguish between domain (e.g., maze movement actions), environment (e.g., a specific maze), and initial/goal states of the agent within the environment (e.g., initial and goal locations). To solve this, we use the problem-name (given in the problem file). The problem name will be composed of two parts: "<env>-<task>", where <env> is a string telling your agent which environment it is, and <task> is a string telling your agent what is the task (initial state, goal states) to be carried out.

For example, your agent may load a domain and problem files with domain name "maze", problem name "maze0-prob4", which specified a specific maze layout, in which the initial and goal states are specified in a particular way. A different run may have problem name "maze0-prob3", which indicates that it is the same maze, but the initial location of the agent, and the goal locations, may be different.

This is done so that your agent, for example, can learn a map of the maze, without committing to a specific initial location or goals. Thus, use of model-based reinforcement learners can be beneficial.

Execution

Whether learning or not, the agent will be given the opportunity to execute the tasks, once. We will use the command:

python my_executive.py -E <domain_file> <problem_file>

This is essentially the same as in all previous assignment: the agent is required to try to carry out the task. If it has relevant learned knowledge, it can use it to improve execution. The agent will be evaluated on this run (we may run multiple times to average the results).

Reporte

You will need to write a report describing the agent and the choices you made in developing it. The report (we estimate 3-7 pages should be enough) needs to describe the pseudo-code for the major components of the agent, including of course the main control loop, but also any other components that are used (e.g., a learning algorithm, a CHOOSE() mechanism, etc.)

A **preliminary** presentation is required in class (5-6 slides, no more) on the two last classes of the course, June 21 and June 28.

You need to submit the **presentations** separately! (until June 20 at 23:59:59 IST).

The report is due with the agent code, June 30 at 23:59:59 IST. Please submit a folder with the code, a PDF file for the report, and a PDF of the presentation.

Grading

Grading of the agent code is based on the number of successful runs, their efficiency (in terms of number of actions), and the agent efficiency (in terms of CPU time).

Grading of the report is based on the description quality and clarity, the use of techniques used in class (bonus for those going outside of class materials!), and the presentation.

- At the beginning of each file, add your name and ID
- Your code will be run on an ubuntu16 make sure your code works on a this O.S. If your code works on your O.S that isn't ubuntu16 but won't work on our computer, you will not be able to appeal on your grade.
- Your work should be performed independently.

GOOD LUCK!