COMP3702/COMP7702 Artificial Intelligence
Semester 2, 2020
Tutorial 1

Before you begin, please note:

- Tutorial exercises are provided to help you understand the materials discussed in class, and to improve your skills in solving AI problems.
- Tutorial exercises will not be graded. However, you are highly encouraged to do them for your own learning. Moreover, we hope you get the satisfaction from solving these problems.
- The skills you acquire in completing the tutorial exercises will help you complete the assignments.
- You'll get the best learning outcome when you try to solve these exercises on your own first (before your tutorial session), and use your tutorial session to ask about the difficulties you face when trying to solve this set of exercises.

This tutorial covers material from Module 0 and the start Module 1 of COMP3702/COMP7702. The information provided below will help you answer the exercises, and will be covered in detail in Week 3's lecture.

To solve an **agent design problem**, the following components are required:

- **Action Space** (A): The set of all possible actions the agent can perform (sometimes called the *action set* in the discrete case). An action is denoted $a \in A$.
- **Percept Space** (P): The set of all possible things the agent can perceive.
- **State Space** (S): The set of all possible configurations of the world the agent is operating in (sometimes called the *set* of states in discrete state systems). A state is denoted $s \in S$.
- **World Dynamics/Transition Function** ($T : S \times A \to S'$): A function that specifies how the world changes when the agent performs actions in it; a system model. We sometimes write $T(s, a) = s'$.
- **Perception Function** ($Z : S \to P$): A function that maps a world state to a perception.
- **Utility Function** ($U : S \to \mathbb{R}$): A function that maps a state (or a sequence of states) to a real number, indicating how desirable it is for the agent to occupy that state/sequence of states. We sometimes write $U(s) =$ some cost or reward.

In addition, P&M provide more granular discussion of the dimensions of complexity in the agent design space (see P&M Ch 1.5), which we covered in the first lecture. Draw on both of these agent design frameworks when completing the exercises below.

A **state graph representation** is a way to represent a search problem concretely in a program. It is also a way of thinking about the problem, and sometimes we may or may not explicitly represent the state graph. Moreover, in problems with continuous or very large state space, a state graph is often used as a compact representation of the state space (more on this later in the course). Formally, a state graph $G = (V, E)$ comprises:

- **Vertices** (V) representing states, and
- **Edges** (E) representing world dynamics

Each edge $\overline{ss'} \in E$ is labelled with the cost to move from s to s'. It may also be labelled by the action to move from state s to s'. The initial and goal states are mapped to the initial and goal vertices of the graph. A solution is a path from initial to goal vertices in the state graph. A solution's *cost* is the sum of the cost associated with each edge in the path. Given this, the optimal solution is the shortest path through the state graph. In Week 3's lecture, we will explore several different ways to **search** over a problem's state graph representation to find a good solution.

Exercises

Exercise 1.1. Design a **tic-tac-toe** or *noughts-and-crosses* playing agent, using the design components listed above. Assume that a single time step includes a single move by the agent and the immediate move by the opponent. The goal is to win with as few steps as possible.

O	X	O
X	O	
X		

Example tic-tac-toe board

Exercise 1.2. Consider a **navigation app**, like an app on your smart phone or car that you use to find your way around UQ or other places. This program is essentially a rational agent. Assume that:

- Its goal is to find the shortest path to a goal location,
- The map used by the agent is 100% up to date,
- The location provided by the GPS is correct.

Suppose that you want to develop this navigation agent,

(a) How will you design it? Use the design components listed earlier.

(b) Select the type of environment this agent operates in (i.e., discrete/continuous, deterministic/non-deterministic, fully/partially observable, static/dynamic)? Explain your selections, and think of the effect of each assumption above to this type.

(c) Define the search problem and its corresponding state graph representation for this query.

Exercise 1.3. A **web crawler** is a program that systematically browses and downloads web pages from the internet. This is one of the programs that enables us to search the internet. A web crawler can be viewed as a rational agent. Please design a web crawler agent when the agent lives in:

(a) An ideal world where no broken links exist and the internet connection always works.

(b) The real world, where both assumptions above are not valid.

Exercise 1.4. A **poker bot** is a program that automatically plays poker on the internet. Poker bots are software agents that typically use AI techniques to attempt to beat human poker players. Think about how to design a poker bot for the version of poker called *Texas hold 'em*, with rules:

- Every player is dealt two cards, for their eyes only.
- The dealer spreads five cards face up for all to see in three *stages*: (i) three at once, (ii) then another, (iii) then the last. All five face-up cards can be used by all players to make their best possible five-card hand.
- Before and after the card/s in each *stage* are revealed, players take turns to bet.
- The best poker hand wins the pot (all the the bets).

What complications arise when a poker bot tries to play against more than one other poker player?