COMP3702 Tutorial 1

Artificial Intelligence

Matt Choy | matthew.choy@uq.edu.au

Agent Design Problem

- To build useful, intelligent computer systems, we must first clearly define the inputs (percepts) and outputs (actions) we expect the systems to do
- In this course, we do this using the Agent Design Problem framework.
- Using this framework means that we don’t have to save the same problems over and over again.
  - If the problem definition is the same, we can re-use an existing solution
  - We can evaluate multiple algorithms’ performance, both in speed and accuracy
- Standardises the interface between the set of problems and solving algorithms.

Agent Design Problem Components

The following components are required to solve an agent design problem

1. Action Space (A) The set of all possible actions the agent can perform. This is sometimes called the *action set* in the discrete case. An action is denoted $a \in A$.
2. Percept Space (P) The set of all possible things an agent can perceive.
3. State Space (S) The set of all possible configurations of the world the agent is operating in. This is sometimes called the *set of states* in discrete state systems. A state is denoted as $s \in S$
4. World Dynamic / Transfer Function ($T: S \times A \rightarrow S'$) A function that specifies how the world changes when the agent performs actions in it. This maps a 2-tuple of states and actions to a single resulting state (this function specifies how we move from one state to the next.
5. Percept Function ($\mathbb{Z} : S \rightarrow P$) A function that maps a world state to a perception. Note that for fully observable problems, the percept function is the identity function $I(x)$
6. Utility Function ($U: S \rightarrow \mathbb{R}$) A function that maps a state (or sequence of states) to a real number. This indicates how desirable it is for an agent to be in that state (or set of states). $U(s) = cost\ or\ reward$

Search Problem

- After we define the Agent Design Problem, we just need to define the initial state and goal state to fully define the search problem.
- We use a State Graph Representation to represent a search algorithm concretely in a program – it's just another way to think of the problem.

State Graph Representation

- Typically used in problems with continuous or very large state spaces to compactly represent the state space.
- Formally, a state graph $G(V, E)$ comprises of a set of vertices (V) which represent states and edges (E) which represent world dynamics
  - Each edge $\overline{ss}' \in E$ **is labelled with the cost to move** from s to s'.
  - Each edge **may be labelled with the action used** to transition from s to s'.
- A **solution** is a path from the initial to goal vertices
- An **optimal solution** is the shortest path (lowest cost path) through the state graph, from the initial state to the goal state

Exercise 1.1 - Tic Tac Toe

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.

Use the Agent Design Problem components:

- Action Space
- Percept Space
- State Space
- World Dynamics or Transition Function
- Percept Function
- Utility Function

Exercise 1.1 – Tic Tac Toe

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**.

Assumptions

- At each time step the agent makes a move (immediately followed by a move by the opponent) – this is done to frame the problem as a single-agent decision making problem.
  - We represent the behaviour of the opponent in the world dynamics
- The goal of the game is to win in as few steps as possible.

Exercise 1.1 – Tic Tac Toe

State Space

- All possible permutations and combinations of the tic-tac-toe board.
- There are $3^9 = 19{,}683$ possible combinations, so how can we compactly represent this?

$$S = \{(t_1, t_2, t_3, \dots, t_9) | t_i \in [X, 0, \_]\}$$

Exercise 1.1 – Tic Tac Toe

Action Space

- The set of all possible actions that the player can perform
- Placing the player's symbol in any of the available squares

$$A = \{p_i \mid i \in [1, 2, \dots, 9]\}$$

where p_i is the action to place the player's symbol in tile t_i

Exercise 1.1 – Tic Tac Toe

World Dynamics

- Agent Move: When the agent performs an action p_i the tile t_i is updated with the agent's symbol
- Opponent Move: One of the available tiles is filled with the opponent's symbol (opponent's choice of tile depends on problem assumptions)
- Not completely deterministic as there are multiple possible squares in which the opponent could choose between
- If the agent is playing against a human opponent, we need to model how the opponent will play (not deterministic, or consistent).

Exercise 1.1 – Tic Tac Toe

Utility Function

- Want the utility function to reward winning in the smallest number of moves
  - In this case, # of tiles placed == number of moves.
- We could define the utility function as:

$$U(s) = \begin{cases} 0, & \text{if no 3-in-a-row cases} \\ 10 - numOccupied, & \text{if agent has 3-in-a-row} \\ numOccupied - 10, & \text{if opponent has 3-in-a-row} \end{cases}$$

Exercise 1.1 – Tic Tac Toe

Percept Space / Percept Function

- Is the environment fully observable or partially observable?

Exercise 1.1 – Tic Tac Toe

Percept Space / Percept Function

- Is the environment fully observable or partially observable?
  - The game is fully observable as the agent is able to observe the exact state at every time step.
- What do we do in this case?

Exercise 1.1 – Tic Tac Toe

Percept Space / Percept Function

- Is the environment fully observable or partially observable?
  - The game is fully observable as the agent is able to observe the exact state at every time step.
- What do we do in this case?
  - The percept function is just the identity function.

Exercise 1.2 – Navigation App

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

1. Its goal is to find the shortest path to a goal location
2. The map used by the agent is 100% up to date
3. The location provided by the GPS is up to date

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 1.2 – Navigation App

About this Question

- How do we define a state?
- How precise does the navigation need to be?
- Does the user always follow the directions correctly?
- Do we need to account for traffic? Pedestrians? Speed limits?

Exercise 1.2 – Navigation App

About this Question

- How do we define a state?
- How precise does the navigation need to be?
- Does the user always follow the directions correctly?
- Do we need to account for traffic? Pedestrians? Speed limits?

Exercise 1.2 – Navigation App

Agent Design Components

- State Space: The set of all valid street addresses / landmarks?

Exercise 1.2 – Navigation App

Agent Design Components

- State Space: The set of all valid street addresses / landmarks?
- Action Space: Set of all actions that can be performed by the agent
  - Legal and driving manoeuvres
  - Specified by movement and heading?

Exercise 1.2 – Navigation App

Agent Design Components

- State Space: The set of all valid street addresses / landmarks?
- Action Space: Set of all actions that can be performed by the agent
  - Legal and driving manoeuvres
  - Specified by movement and heading?
- World Dynamics: Position changes to the next node in the direction the agent selected (assuming that the agent always follows the navigation system)
  - How is this affected if the user doesn’t necessarily follow the instructions of this system?

Exercise 1.2 – Navigation App

Agent Design Components

- State Space: The set of all valid street addresses / landmarks?
- Action Space: Set of all actions that can be performed by the agent
  - Legal and driving manoeuvres
  - Specified by movement and heading?
- World Dynamics: Position changes to the next node in the direction the agent selected (assuming that the agent always follows the navigation system)
  - How is this affected if the user doesn't necessarily follow the instructions of this system?
- Utility Function: Reach the goal state with the minimum distance travelled

$$U(s) = distanceTravelled \times -1$$

Exercise 1.3 – Web Crawler

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 idea world where no broken links exist and the internet connection always works
b. The real world, where both assumptions above are not valid

Exercise 1.3 – Web Crawler

State Space: Set of all valid web addresses / URLs

This may be hard to represent mathematically, need to build up as we go. Possible next state depends on the number of links on the current page. Impossible to enumerate over each possible sequence of characters that could for a URL.

Action Space: Set of all links which can be followed (changes depending on the current state)

World Dynamics: State changes to the URL of the selected link; in the non-ideal case, the link may be broken or the internet connection may be unavailable, causing the state to return to the previous webpage

Exercise 1.3 – Web Crawler

Utility function: Number of unique webpages visited (derived from sequence of states, and counting the distinct vertices / sites visited)

Percept Space / Percept Function: Fully observable State Space / Identity Function

Exercise 1.4 – Poker Bot

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 platers. Think about how to design a poker bot for the version of poker called *Texas hold 'em*, with the following 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) a single card (iii) another single card
- 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 bets)

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

Exercise 1.4 – Poker Bot

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 platers. Think about how to design a poker bot for the version of poker called *Texas hold 'em*, with the following 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) a single card (iii) another single card
- 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 bets)

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

Exercise 1.4 – Poker Bot

- State Space: All combinations of
  - Player’s current hands / opponent’s current hand(s), current dealer’s hand(s)
- Action Space
  - Assumes players always raise by the same amount, otherwise 1 action for each raising amount
- World Dynamics
  - Number of chips in pot changes based on player bets
  - Chips are awarded to the winner of the round
  - Cards are dealt again at the end of the round/start of the next round
  - Other player's actions (needs to be sophisticated enough to describe the actions that the other players might take)

Exercise 1.4 – Poker Bot

- Utility Function
  - Number of poker chips in hand at the end of all of the rounds
- Percept Space and Percept Function
  - Cards in the player's own hand are revealed + some of the common cards depending on the current state
  - Cards not dealt, cards in opponent's hand are not part of the percept.
  - Partially observable.
  - Same percept, but underlying state is different

Exercise 1.4 – Poker Bot

Complications

- Complications when there are multiple opponents – multi-agent problem
- Size of state space grows exponentially as number of place increases
- Modelling opponent behaviour becomes very complicated.