CS4618: Artificial Intelligence I

Deliberative Agents

Derek Bridge
School of Computer Science and Information Technology
University College Cork

Initialization

In [1]:

```
%reload_ext autoreload
%autoreload 2
%matplotlib inline
```

In [2]:

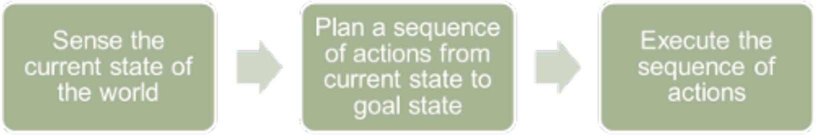
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Thinking ahead

- In the sense-plan-act cycle, the plan phase should generally be more deliberative
- Thinking ahead is a form of simulation
  - Trying out actions on a mental representation prior to executing the actions in the actual world
- Class exercise: Give precise reasons why thinking ahead is advantageous: what can go wrong if you don’t think ahead?
- Class exercise: Are there times when thinking ahead is disadvantageous: what can go wrong if you do think ahead?

Planning sequences of actions

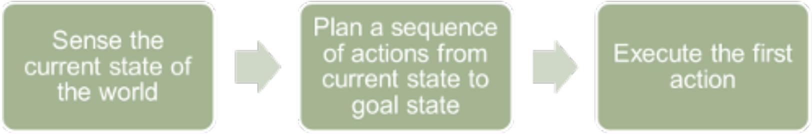
- Often, the agent will plan whole sequences of actions
- But, there are at least two ways of integrating this with execution:



Sense the current state of the world

Plan a sequence of actions from current state to goal state

Execute the sequence of actions



Sense the current state of the world

Plan a sequence of actions from current state to goal state

Execute the first action

- Class exercise: The second approach appears to be wasteful. But the first approach is suitable only for certain environments. What kinds of environments?

State space

- We implement deliberation (thinking ahead) as a form of **search** through a directed graph
- **State space**:
  - all states reachable by sequences of actions from some start state
- Represented by a directed graph in which
  - nodes represent states of the world
  - edges represent actions (state transformations)
- The task is to find a **path** from the node labelled by the **start state** to one of the nodes labelled by **goal states**

State space

- In AI, the graph may be too large to specify and store explicitly
- Instead, specify it implicitly:
  - The start state
  - The set of operators for transforming states to other states
  - The goal condition that can detect whether a state is a goal state
- There can also be a path cost function, g

The 8-puzzle

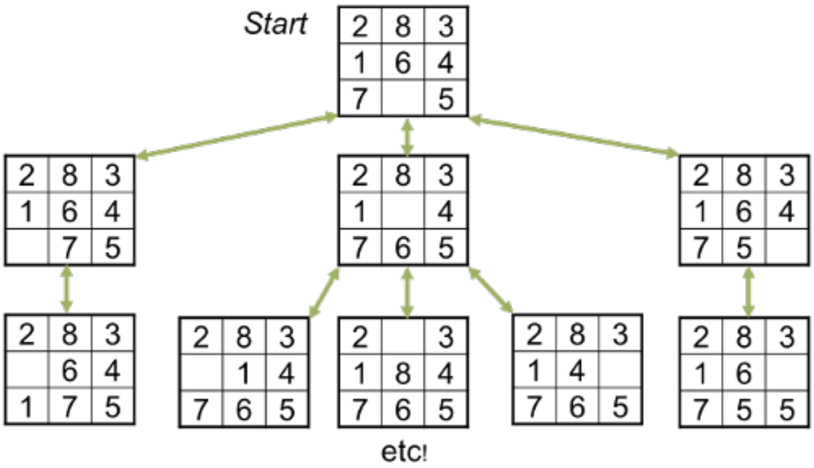
- Sliding 8 numbered tiles around a 3×3 grid
- How to represent the states:
    - 3×3 array of integers
- Start state, e.g.:

2	8	3
1	6	4
7		5

- Operators:
    - If blank is not leftmost, move it left by 1
    - If blank is not uppermost, move it up by 1
    - Etc.
- Goal state, e.g.:

1	2	3
8		4
7	6	5

8-puzzle state space



- This state space has $9! = 362,800$ states

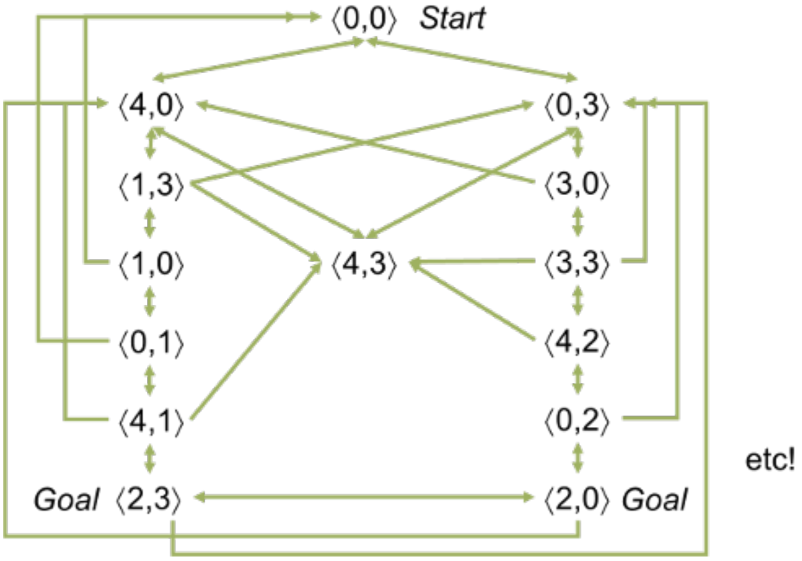
The water jugs problem

- A 4-gallon jug and 3-gallon jug with no measuring markers, and a tap
- Must get exactly 2 gallons into the 4-gallon jug
- Representation of states
  - Pair of integers, $\langle x, y\rangle$
  - x is the amount of water in the 4-gallon jug, $x \in \{0, 1, 2, 3, 4\}$
  - y is the amount of water in the 3-gallon jug, $y \in \{0, 1, 2, 3\}$
- Start state: $\langle 0, 0\rangle$
- Goal state: $\langle 2, n\rangle$

The water jugs operators

1. If $x < 4$ then $\langle 4, y\rangle$
2. If $y < 3$ then $\langle x, 3\rangle$
3. If $x > 0$ then $\langle 0, y\rangle$
4. If $y > 0$ then $\langle x, 0\rangle$
5. If $x + y \geq 4$ then $\langle 4, y - (4 - x)\rangle$
6. If $x + y \geq 3$ then $\langle x - (3 - y), 3\rangle$
7. If $x + y \leq 4 \wedge y > 0$ then $\langle x + y, 0\rangle$
8. If $x + y \leq 3 \wedge x > 0$ then $\langle 0, x + y\rangle$

The water jugs state space



Applications of state space search

- Route planning
- Pathfinding in games
- Cargo loading
- Automatic assembly
- …

```
In [ ]:
```