Manchester Metropolitan University

Artificial Intelligence Principles

6G7V0011 - 1CWK100

Dr. Peng Wang

Email: p.wang@mmu.ac.uk

Department of Computing and Mathematics

Wednesday, Oct. 22nd, 2024

Outline

Outline

Manchester Metropolitan University

Review

Manchester Metropolitan University

- Artificial Intelligence and SOTA techniques
- Agents and Rational agents
- Four types of agent programs
  1. Simple reflex agent
  2. Model-based agent
  3. Goal-based agent
  4. Utility-based agent

In goal-based and utility-based agents, when the goal(s) change, the agent needs to **search** for a new solution.

Agent Programs - Goal-based Agent

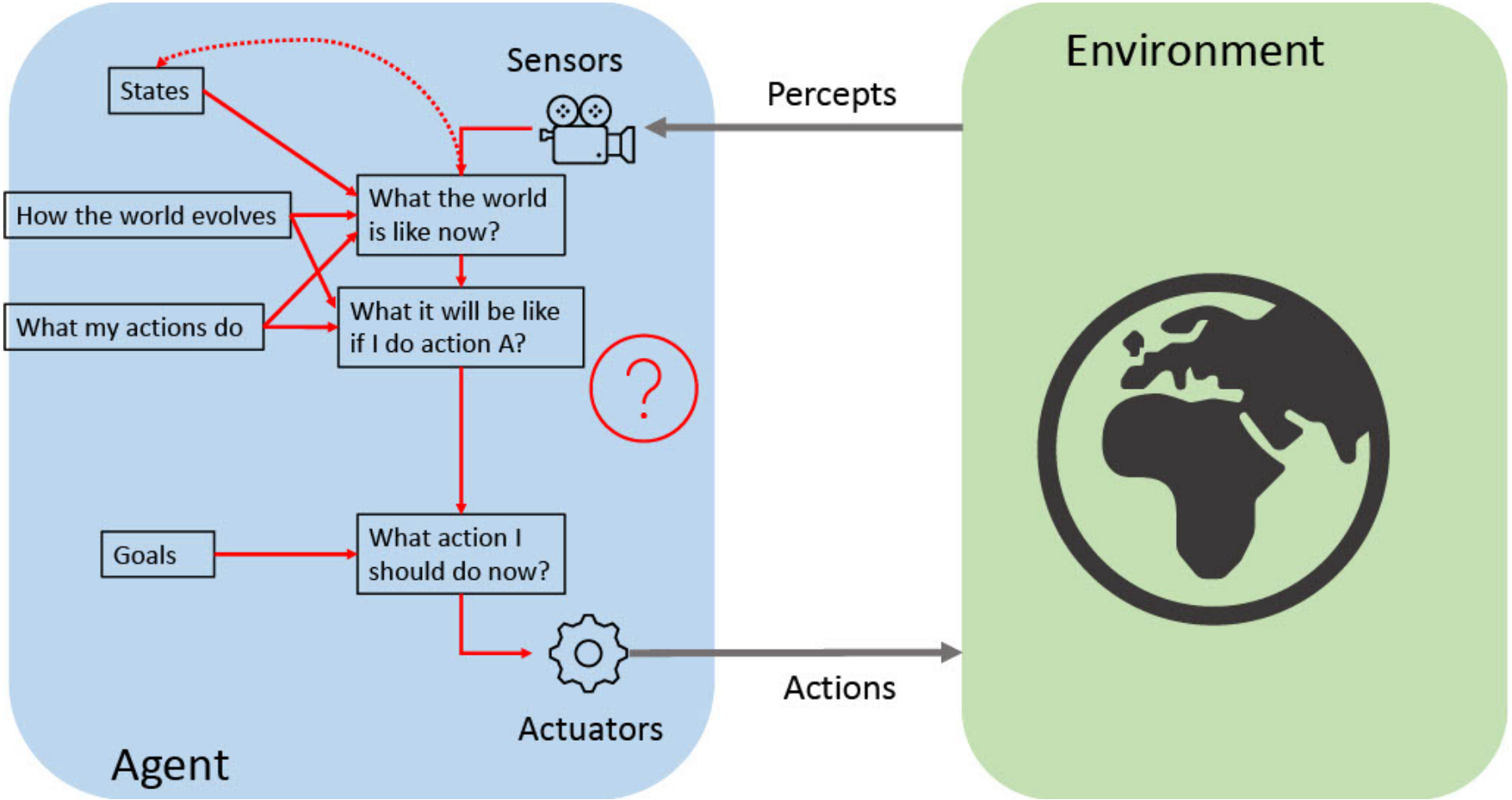


Figure 1: Goal-based agent

Agent Programs - Utility-based Agent

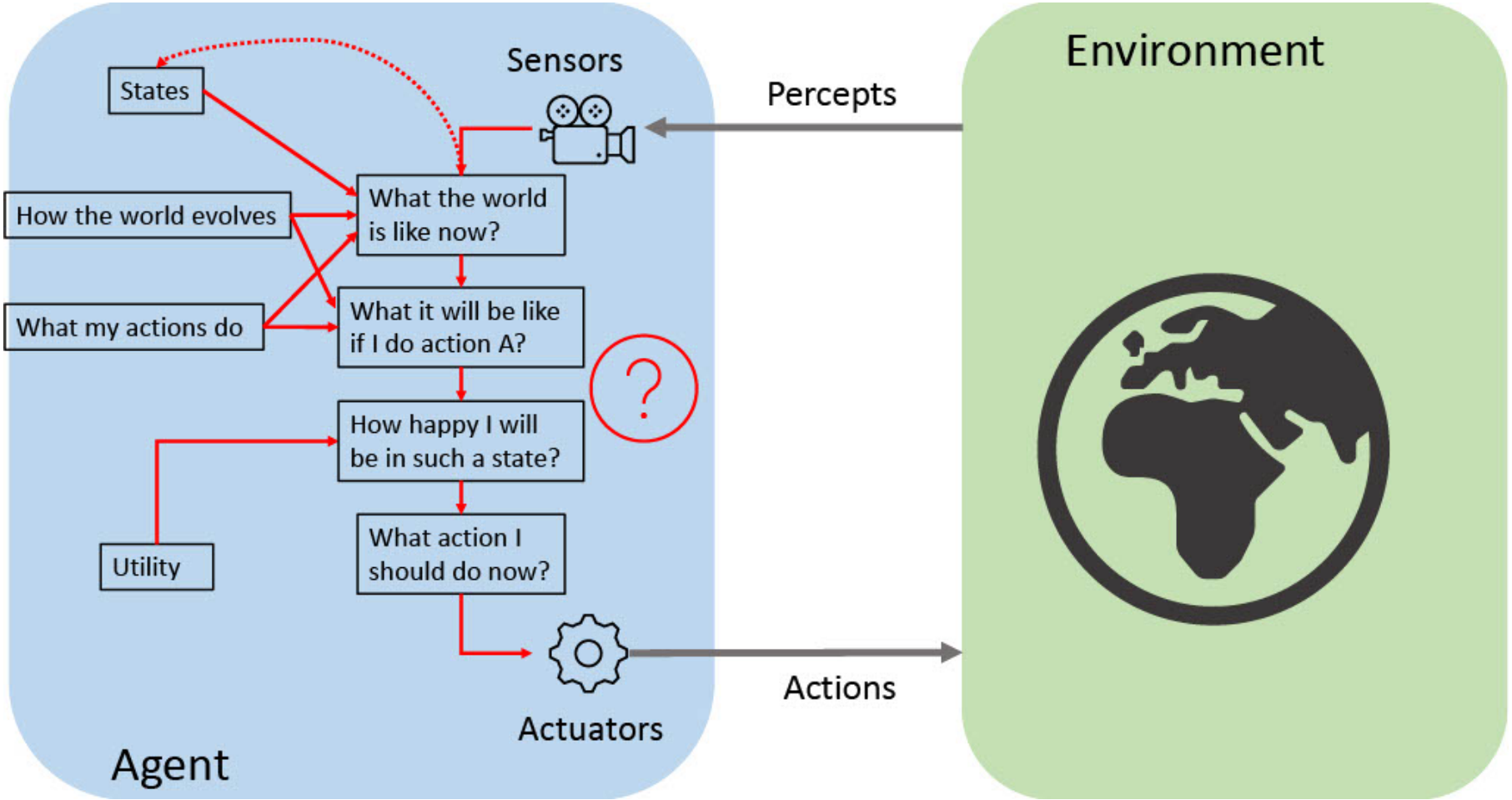


Figure 2: Utility-based agent

Search

Manchester Metropolitan University

In goal-based and utility-based agents, when the goal(s) change, the agent needs to **search** for a new solution.

Search

- **Search** is the computational process undertaken by an agent when no immediately obvious actions are available.
- *Informed search*: Agent can **estimate** the cost from current state to goal(s).
- *Uninformed search*: No such functionality.

*Why **estimate**?*

The cost is from prior knowledge/experience/intuition, etc., but not the truth/fact. You (agent) can only find out by doing (have done) it.

Search happens before action! Alternatively, consider search as a Planning process!

Motivation Example - Path Finding

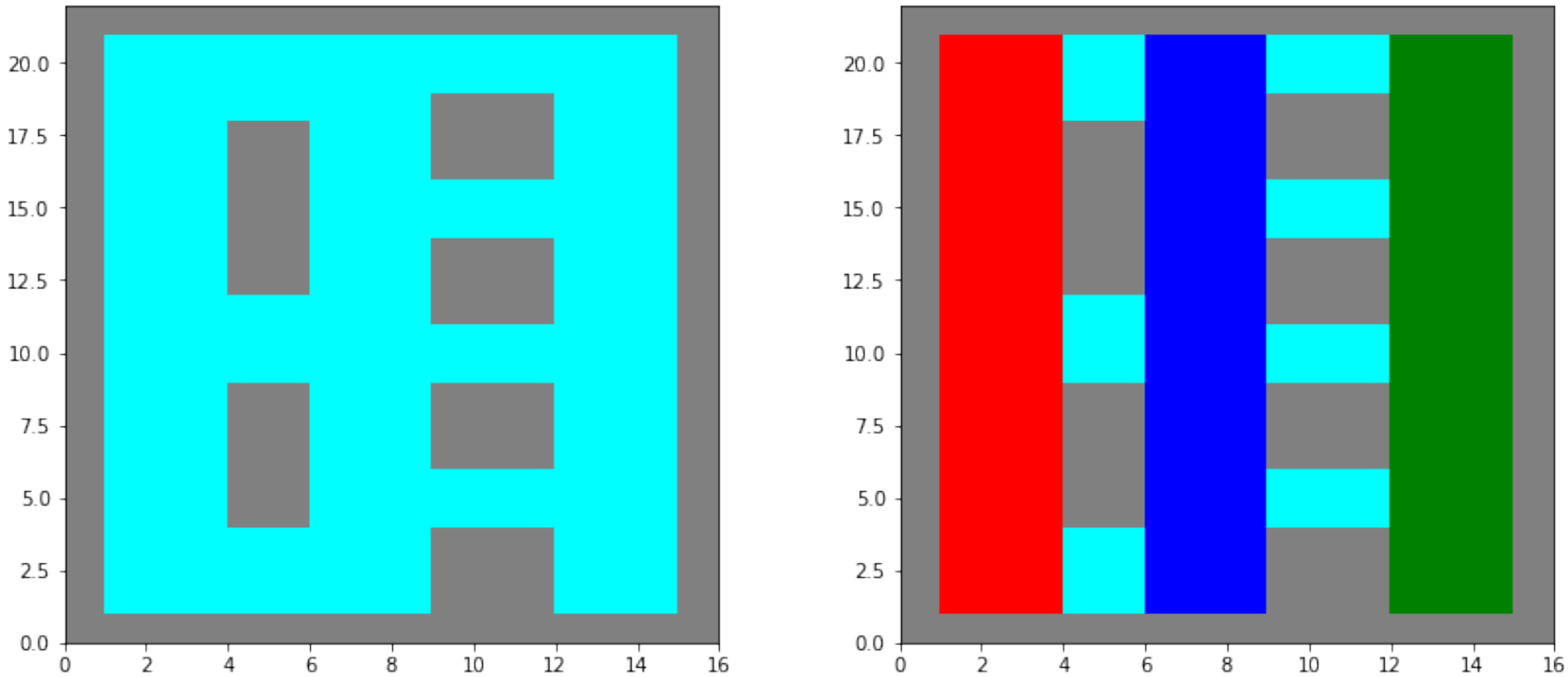


Figure 3: Environment and start (red) & end (green) states, with gray obstacles

Question How many choices from start state to goal state?

Formulating a Search Problem

Manchester Metropolitan University

1. A set of possible **states** that the **world (agent + environment)** can be in. All the possible states together are called **state space**.
2. The **initial state** (start state) that the agent starts in.
3. A set of *one* or *more* **goal states**.
4. The **actions** that are available to the agent.
5. A **transition model** describes what each action does.
6. An **action cost function**, denoted by Action-Cost(s, a, s'), that gives the numeric cost of applying action a in state s to reach state s'.
7. A sequence of actions forms a **path**.
8. A **solution** is a path from the initial state to a goal state.
9. An **optimal solution** has the lowest **path cost** among all solutions.
10. The state space can be represented as a **graph** in which the vertices are states and the directed edges between them are actions.

State Space - Abstraction of Path Finding

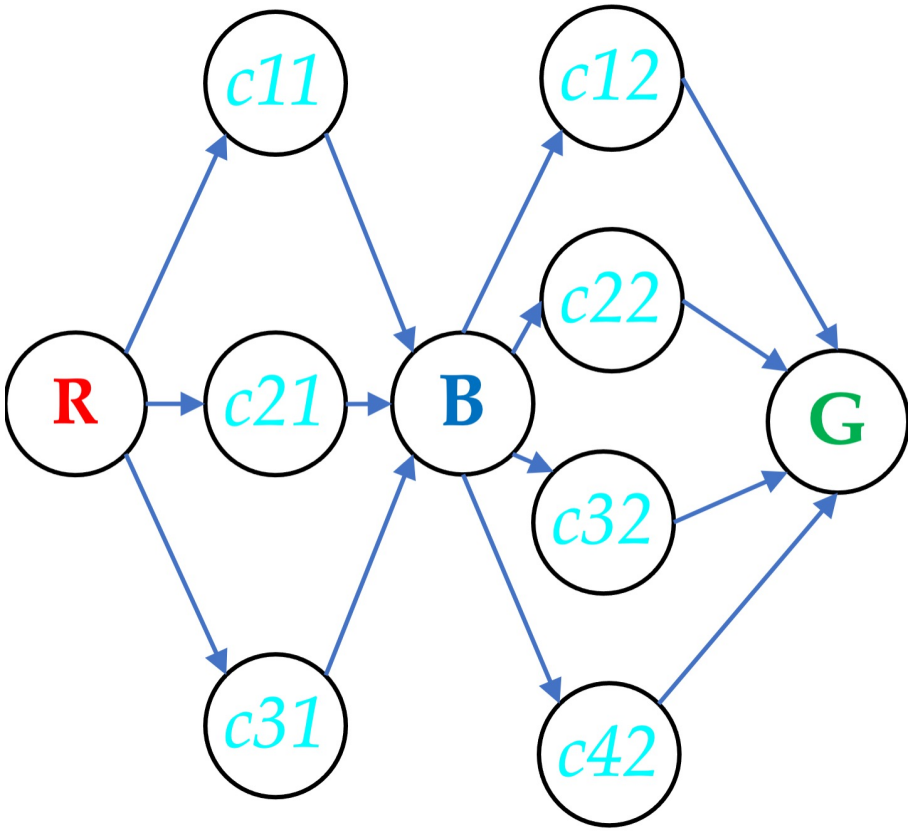


Figure 4: State Space Example

An Examplar Search Problem

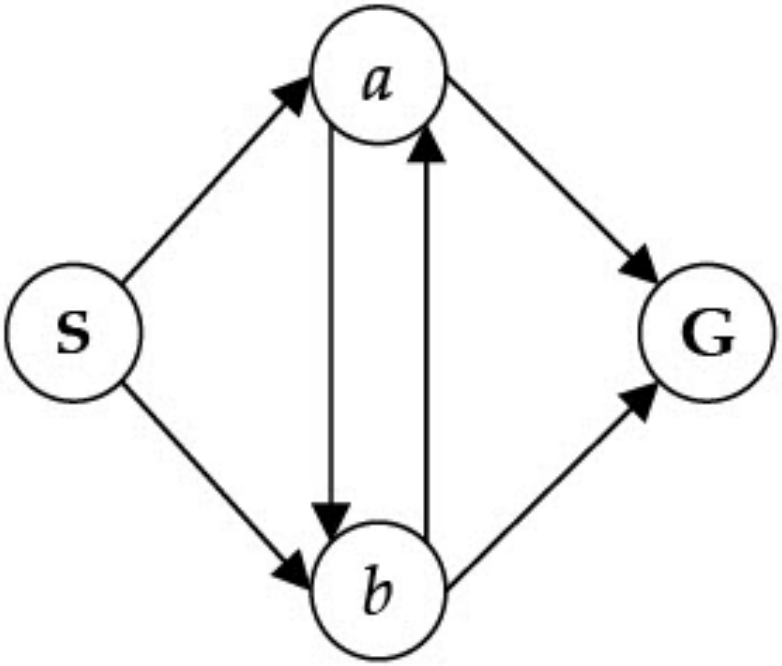


Figure 5: A state space graph

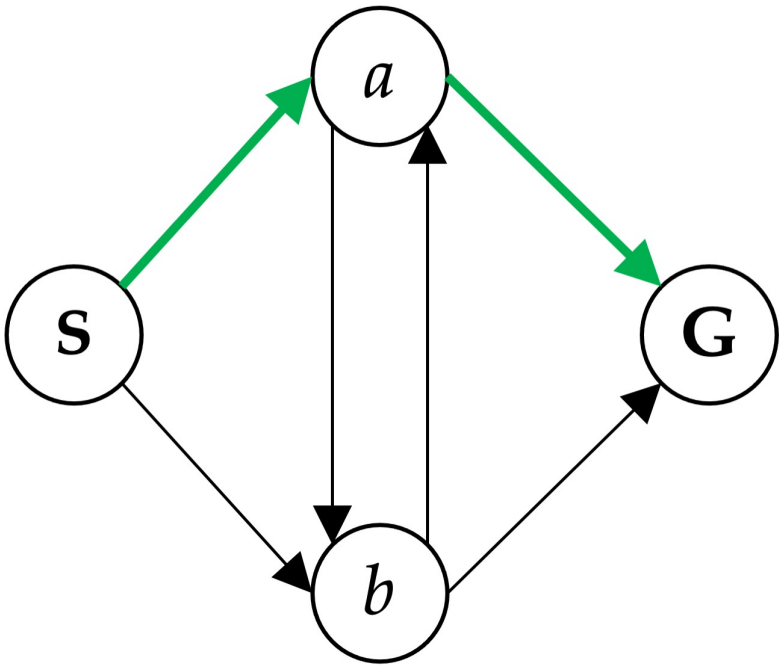


Figure 6: A path found

State Space Graphs vs. Search Trees

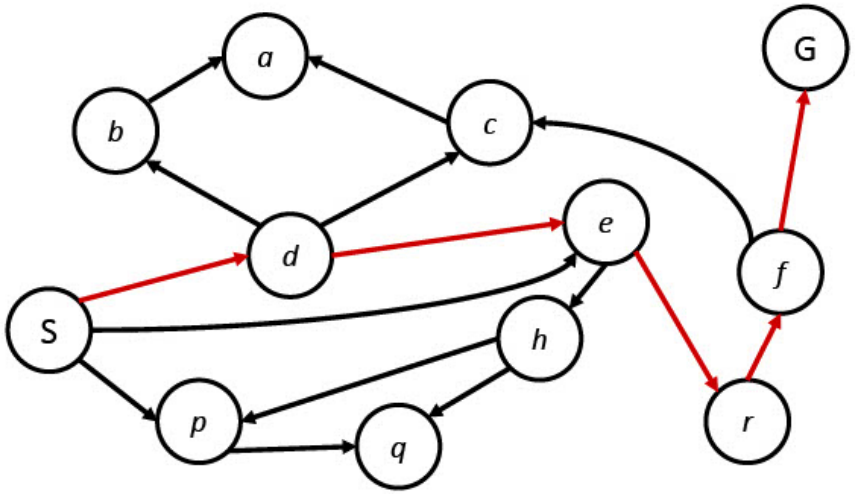


Figure 7: A state space graph

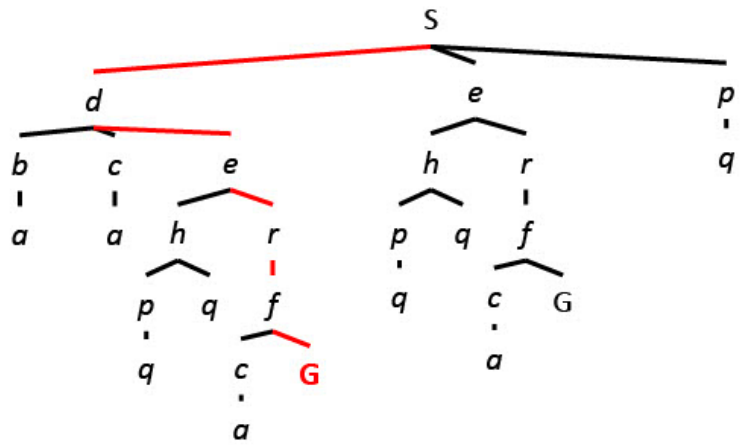


Figure 8: A search tree

- **State space graph**: A mathematical representation of a search problem
- **Search tree**: A 'what if' tree of plans and their outcomes

1. Nodes represent states and edges represent actions
2. Search tree is dynamically built up, until you find the goal
3. More than one nodes correspond to one state in the graph

State Space Graphs vs. Search Trees

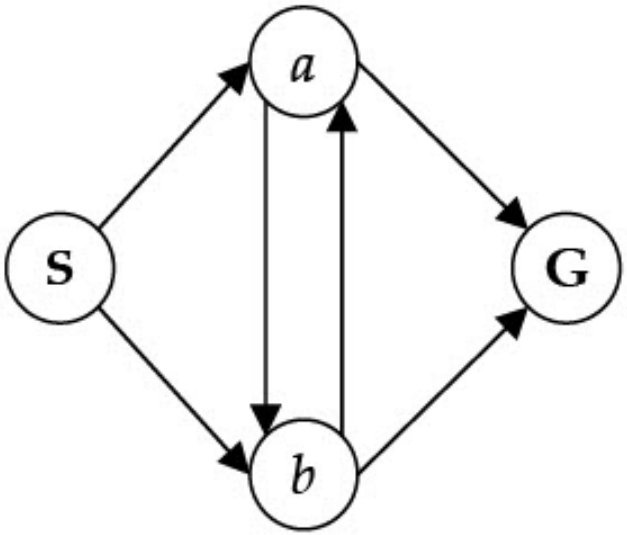


Figure 9: A four-state state space graph

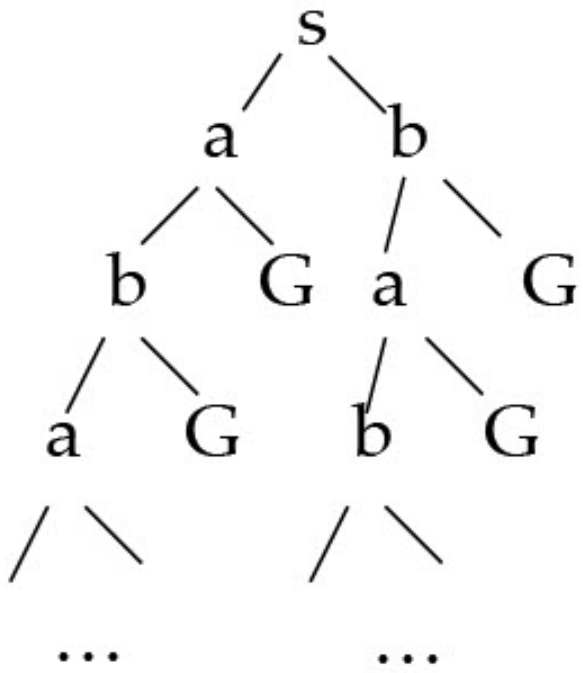


Figure 10: A infinite depth search tree

Note: 4-state graph, an infinite depth tree with cyclic state space, **algorithms are designed to detect cycles**. This is due to the fact that no cycles/loops are allowed in Trees.

Tree Search I

Manchester Metropolitan University

Algorithm 1: General Tree Search

Input: Initial state **s**, goal state **g**, a search *strategy*
Output: A *node* helps to retrieve a solution (path) $\mathcal{P}$, or failure

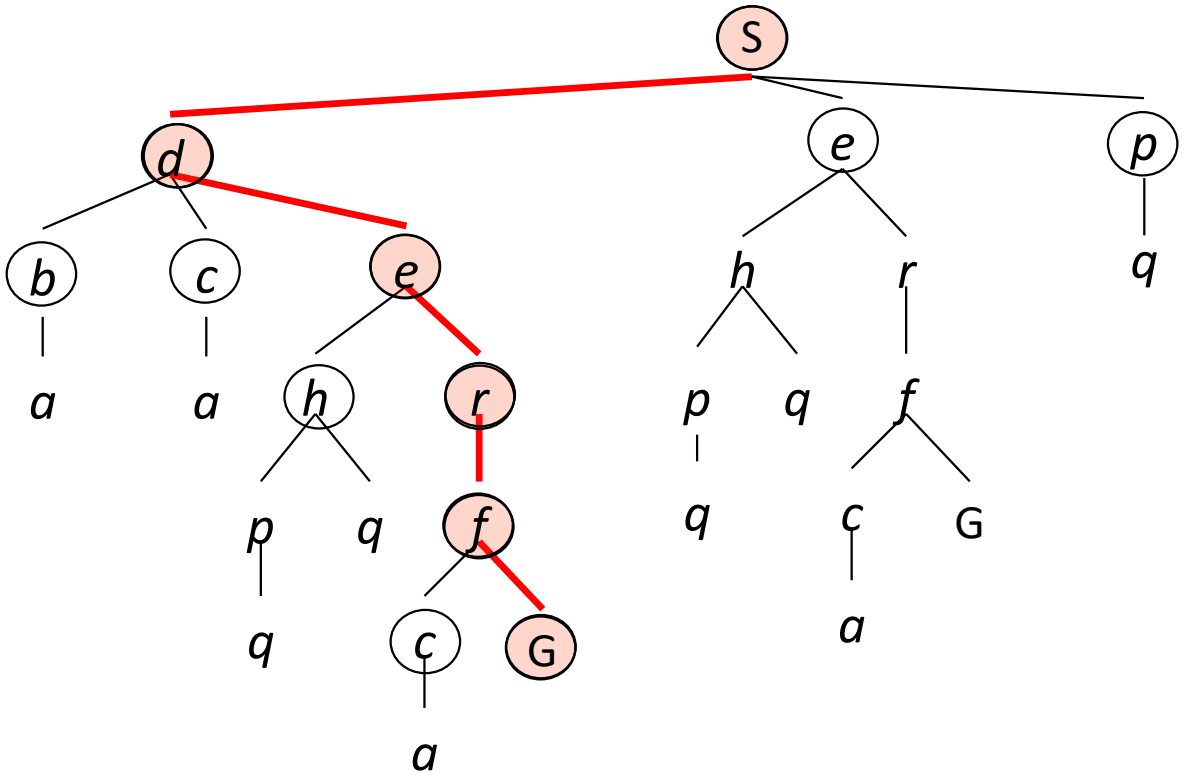
1: *node* $\leftarrow$ with **s** as state
2: **if s** $==$ **g then**
3: **return** *node*
4: **end if**
5: $\mathcal{O} \leftarrow$ *node*, $\mathcal{O}$ is a data structure like LIFO, FIFO, etc. (it can pop out, push in and delete elements, etc.)
6: **while** $\mathcal{O}$!= $\emptyset$ **do**
7: *parent* $\leftarrow$ the first/last/best node from $\mathcal{O}$
8: **if** *parent*.state $==$ **g then**
9: **return** *parent*
10: **end if**
11: **for** *child* **in** successor (of *parent*) **do**

Tree Search II

Manchester Metropolitan University

```
12:        add child to 𝒪
13:    end for
14: end while % Varies from algorithm to algorithm
15: return failure
```

Tree Search - example



s
s → d
s → e
s → p
s → d → b
s → d → c
s → d → e
s → d → e → h
s → d → e → r
s → d → e → r → f
s → d → e → r → f → c
s → d → e → r → f → G

Figure 11: An example of search with search trees

Tree Search - example continued

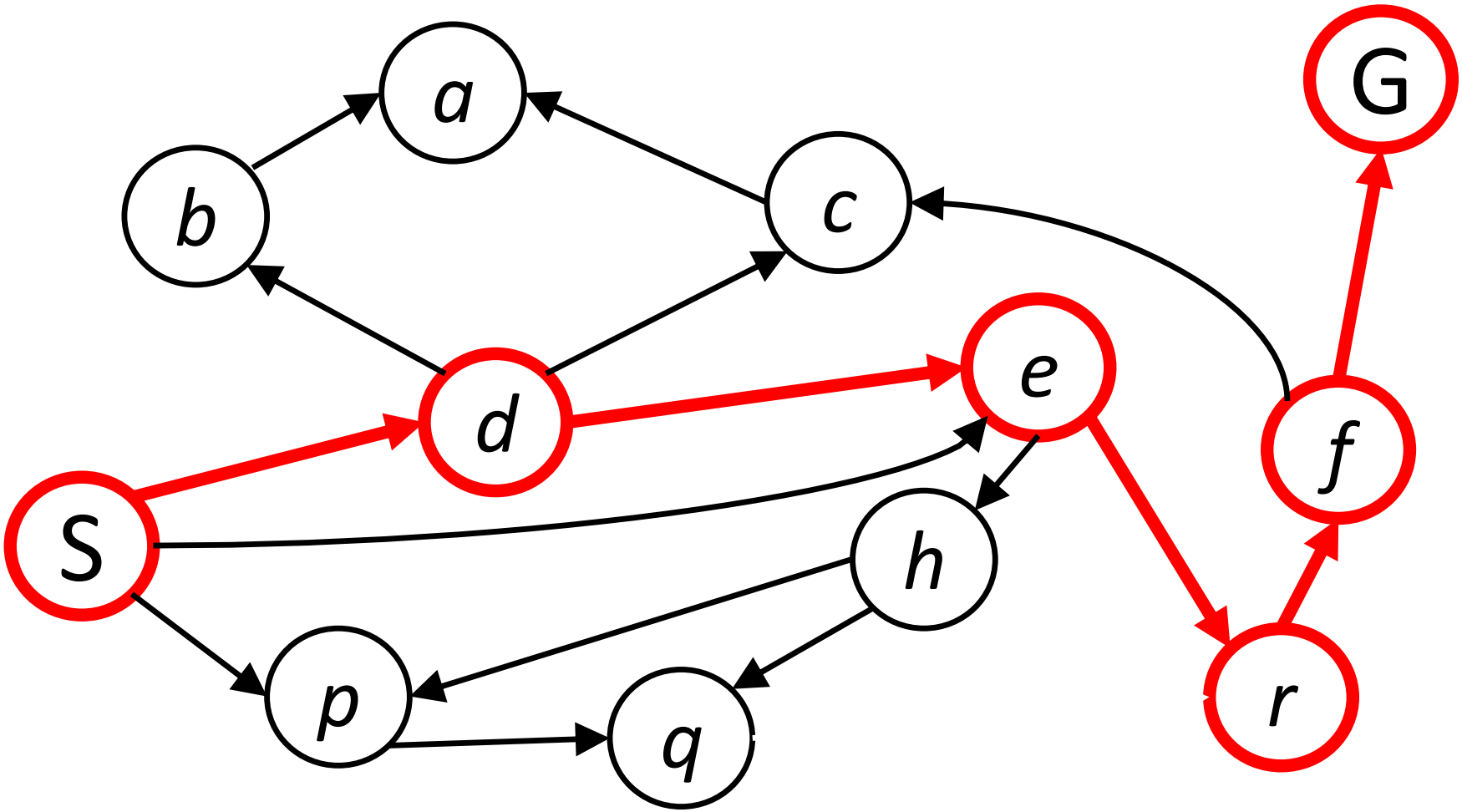


Figure 12: Path obtained

Issues of Tree Search

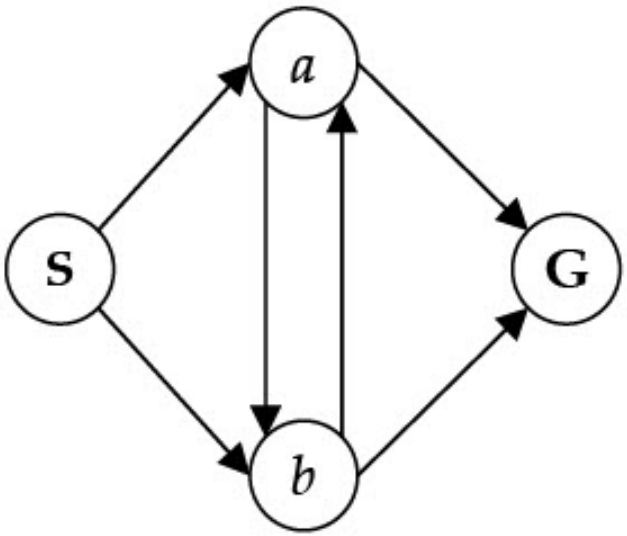


Figure 13: A four-state state space graph

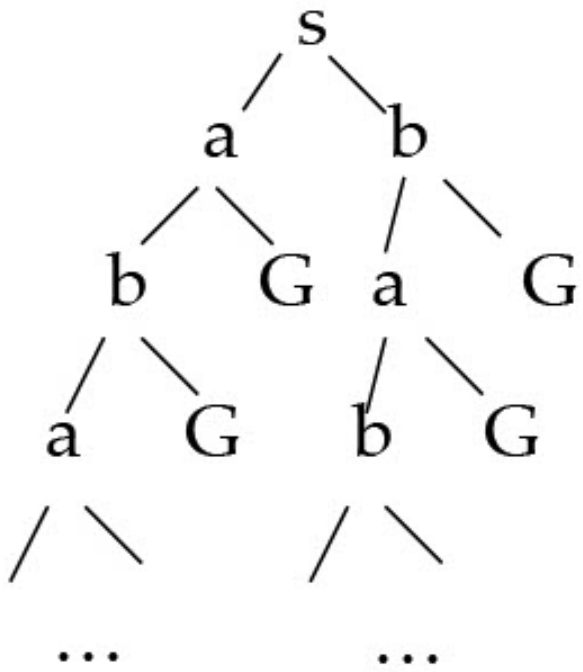


Figure 14: A infinite depth search tree

Note: Does not check for redundant paths. If search in depth first, there will be problems.

Graph Search I

Manchester Metropolitan University

Checks for redundant paths

Algorithm 2: General Graph Search

Input: Initial state **s**, goal state **g**, a search *strategy*
Output: A *node* helps to retrieve a solution (path) $\mathcal{P}$, or failure

1: *node* $\leftarrow$ with **s** as state
2: $\mathcal{O} \leftarrow$ *node*, $\mathcal{O}$ is an data structure like LIFO, FIFO, etc.
3: $\mathcal{C} = \emptyset \leftarrow$ Explored nodes % Redudant path check
4: **while** $\mathcal{O}$!= $\emptyset$ **do**
5: *parent* $\leftarrow$ the first/last/best node from $\mathcal{O}$
6: **if** *parent*.state == **g then**
7: **return** *parent*
8: **end if**
9: $\mathcal{C} \leftarrow$ *parent*.state
10: **for** *child* **in** successor (of *parent*) **do**

Graph Search II

Manchester Metropolitan University

```
11:        v ← child.state
12:        if v not in C and child not in O then
13:            add child to O
14:        end if
15:    end for
16: end while
17: return failure
```

Check This Again with Graph Search

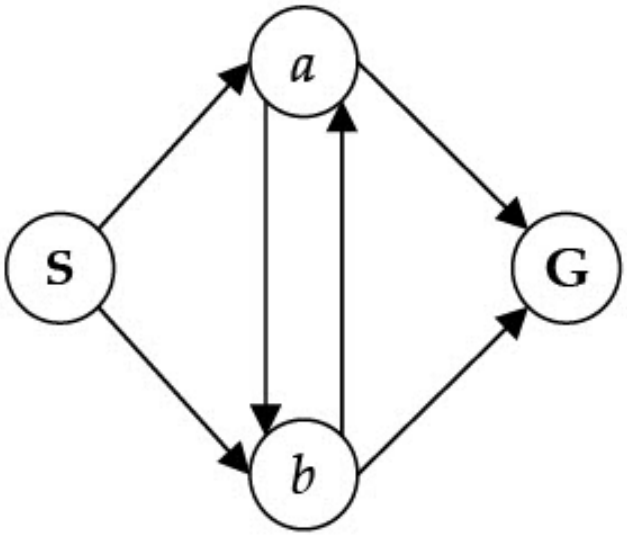


Figure 15: A four-state state space graph

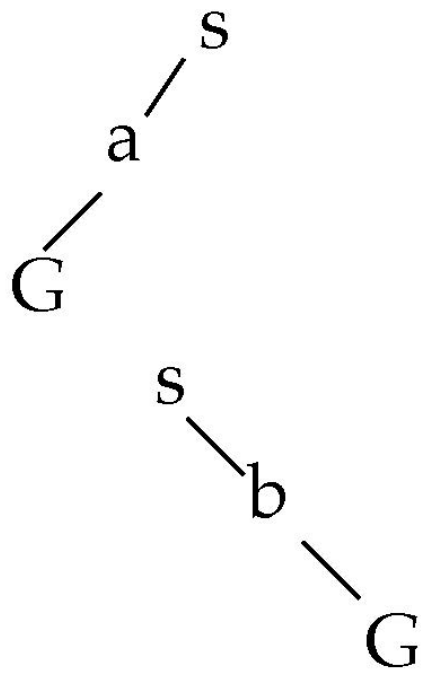


Figure 16: Paths found

Search Concepts

Manchester Metropolitan University

- Frontier, Fringe, Openset
- Reached, closedset
- Expansion
- Exploration strategy - Depth first? Breadth first? etc.

Openset: is the set of nodes we choose currently from - that is, it contains all the nodes we might be interested in looking at next.

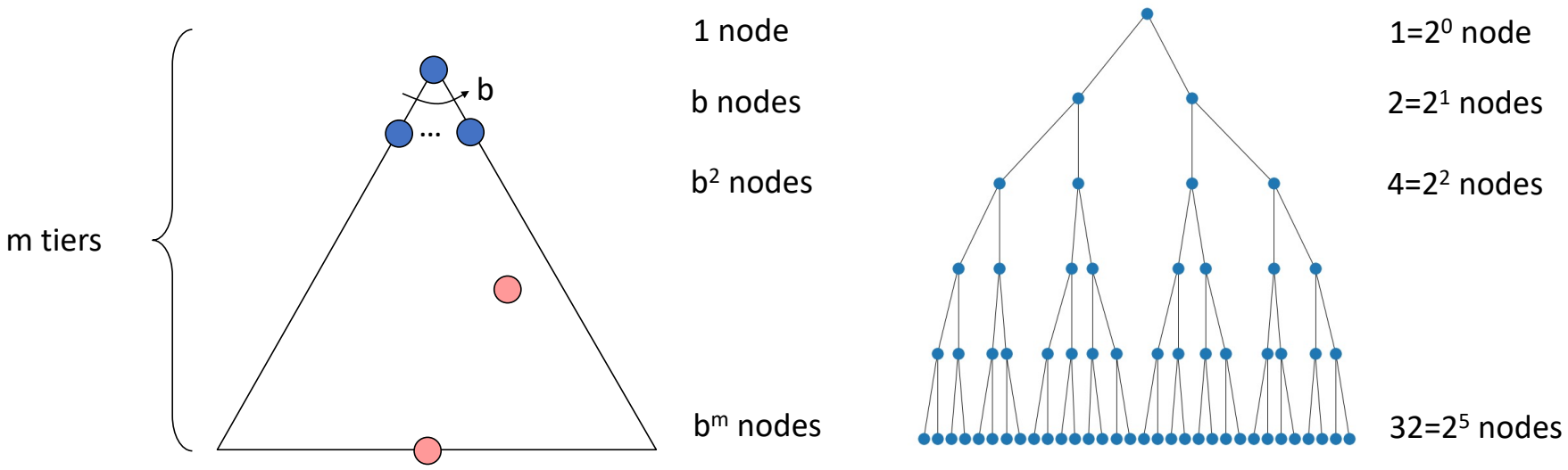
frontier or **fringe**: meaning 'a set of nodes to be expanded. Without causing confusions, we will use them all in the course.

Closedset: is the set of nodes we've already considered.

Sometimes, a Closedset is also called **reached**, meaning 'a set of nodes that are explored already'. Similarly, we will use both terms in this course without causing confusions.

Exploration strategy is the key question!

Search Algorithm Properties



- **Complete**: Guaranteed to find a solution if one exists?
- **Optimal**: Guaranteed to find the least cost path?
- **Time** complexity?
- **Space** complexity?

- Key factors of a search tree
  1. b is the branching factor
  2. m is the maximum depth
  3. solutions at various depths

Node number: $b^0 + b^1 + b^2 + \cdots + b^m = \sum_{i=0}^{m} b^i \backsim \mathcal{O}(b^m)$