Graph Search

CMPUT 261: Introduction to Artificial Intelligence

P&M §3.1-3.4

Recap: Course Essentials

Course information: https://jrwright.info/introai/

- This is the main source of information about the class
- Syllabus, slides, readings, deadlines

Lectures: Tuesdays and Thursdays, 9:30-10:50am in **ETLC E2-001**

- In person

Canvas:

- Discussion forum for **public** questions about assignments, lecture material, etc.
- Handing in assignments

Email: james.wright@ualberta.ca for **private** questions

- (health problems, inquiries about grades)

Office hours: By appointment, or after lecture

- TA's are available to help during lab hours
- Labs begin **next week**

Recap: Search

Example: Farmer's raft

A farmer needs to move a hen, fox, and bushel of grain from the left side of the river to the right using a raft.

- The farmer can take one item at a time (hen, fox, or bushel of grain) using the raft.
- The hen cannot be left alone with the grain, or it will eat the grain.
- The fox cannot be left alone with the hen, or it will eat the hen.

- We want to compute a sequence of actions:
  - from a **starting state** (all of the animals on the left bank)
  - to a **goal state** (all of the animals on the right bank)
  - while satisfying **constraints** (nothing gets eaten)
- Every action has a **known** and **deterministic** result and cost
- **Search:** efficiently compute a cost-optimal solution based on known rules

Lecture Outline

1. Recap & Logistics
2. Search Problems
3. Graph Search
4. Markov Assumption

After this lecture, you should be able to:

- Represent a search problem formally
- Represent a search problem as a search graph
- Implement a generic graph search
- Identify whether a representation satisfies the Markov assumption

Search

- It is often easier to **recognize** a solution than to **compute** it
  - Search exploits this property!
- Agent searches **internal representation** to find solution
  - Outcomes are **known** and **deterministic**, so no need for observations
  - All computation is purely internal to the agent.
- Formally represent as searching a **directed graph** for a path to a goal state
- **Question:** Why might this be a good idea?
  - Because it is very **general**. Many AI problems can be represented in this form, and the same algorithms can solve them all.

State Space

- A **state** describes all the relevant information about a possible configuration of the environment
- **Markov assumption**: How the environment got to a given configuration doesn't matter, just the current configuration.
  - It is always possible to construct such a representation (**how?**)
- A state is an assignment of values to one or more **variables**, e.g.:
  - A single variable called "state"
  - x and y coordinates, temperature, battery charge, etc.
- **Actions** change the environment from one state to another

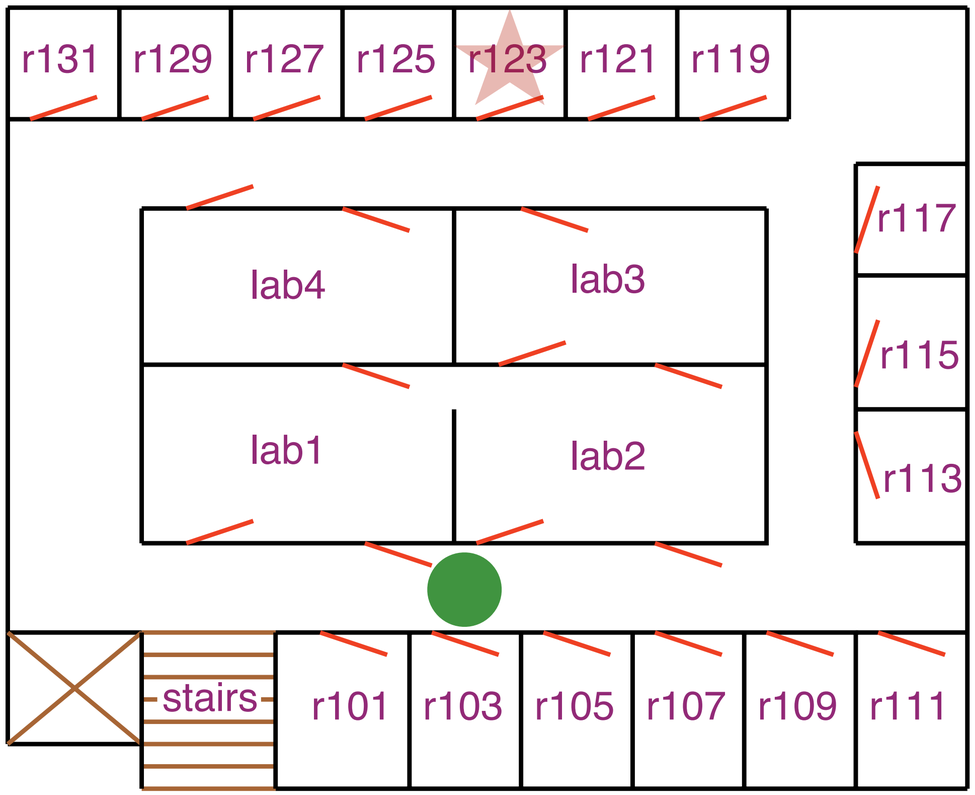
Search Problem

Definition: Search problem (textbook: state-space problem)

- A set of **states**
- A **start state** (or set of start states)
- A set of **actions** available at each state
- A **successor function** that maps from a state to a set of reachable states
  - The textbook calls this an "action function"
- A **cost** for moving from each state to each successor state
- A **goal function** that returns true when a state satisfies the goal

Example: DeliveryBot

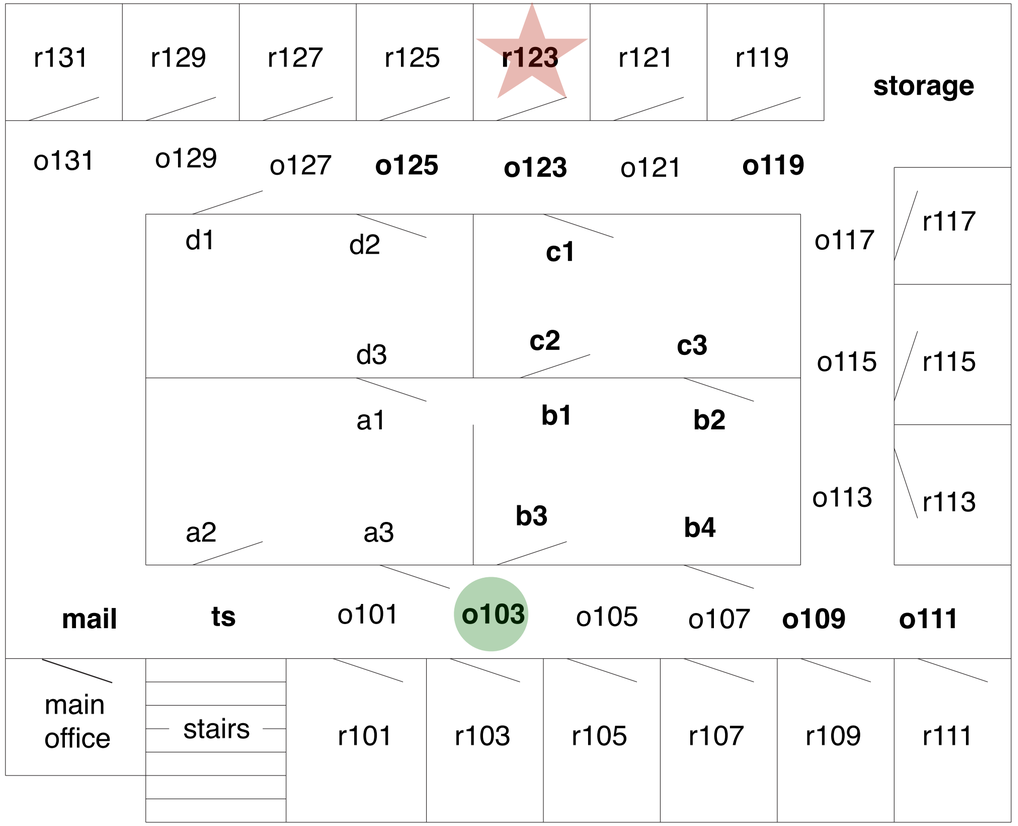
DeliveryBot wants to get from outside room 103 to inside room 123



Question: What might be a better representation for states?

DeliveryBot as a Search Problem

States	{r131, o131, r129, o129, ...}
Actions	{go-north, go-south, go-east, go-west}
Start state	o103
Successor function	succ(r101) = {r101, o101}, succ(o101) = {o101, lab1, r101,o105, ts}, ...
Goal function	$\text{goal}(s) = \begin{cases} 1 & \text{if } s = \text{r123}, \\ 0 & \text{otherwise.} \end{cases}$



https://artint.info/2e/html/ArtInt2e.Ch3.S2.html

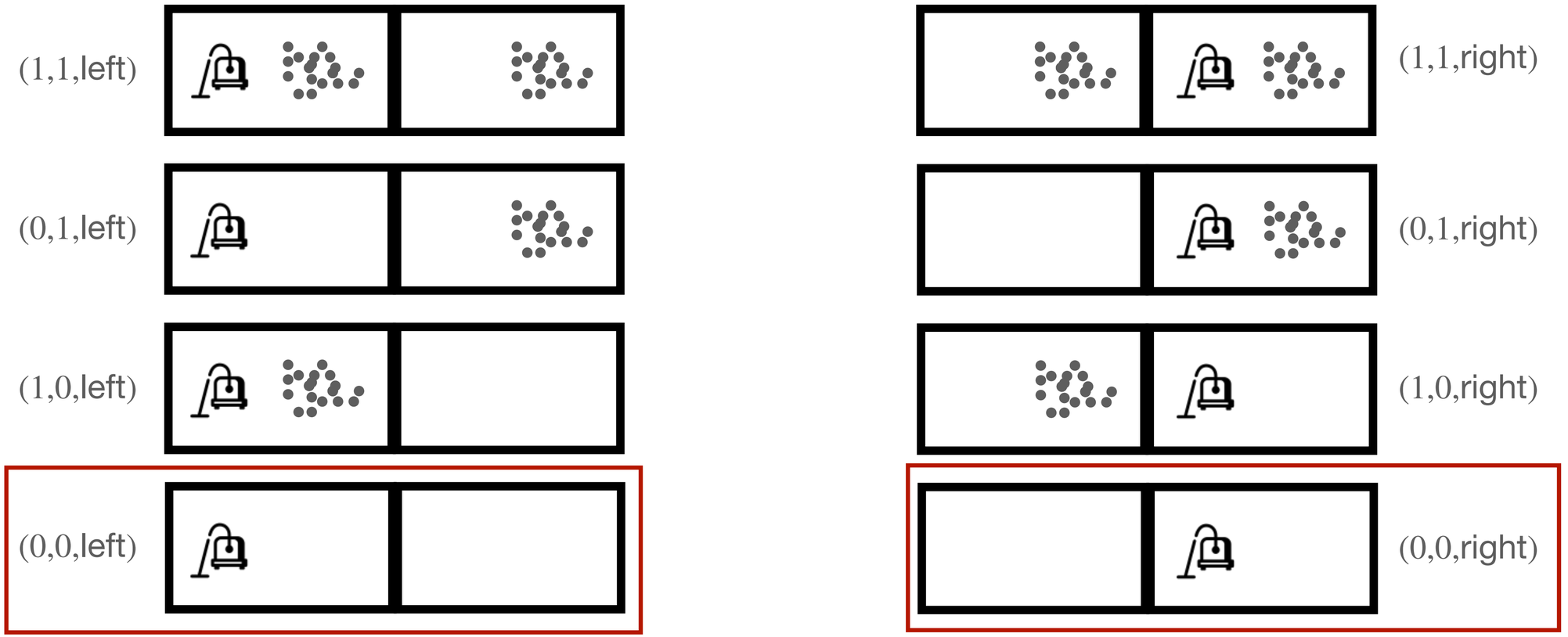
Example: VacuumBot

- Two rooms, one cleaning robot
- Each room can be clean or dirty
- Robot has two actions:
  - **clean**: makes the room the robot is in clean
  - **move**: moves to the other room
- Robot's goal: All the rooms are clean

Questions:

1. How many **states** are there?
2. How many **goal states**?

VacuumBot as a Search Problem: States



A visual description of eight possible states.
There are two rooms; each can be either dirty or clean, and the robot can be in either the left or the right room.
The state with both rooms clean and robot in left room, and both rooms clean and robot in right room, are each outlined in red.

icons by Icons8

VacuumBot as a Search Problem: Formally

States: $\{(x, y, p) \mid x \in \{0,1\}, y \in \{0,1\}, p \in \{\text{left}, \text{right}\}\}$

Start state: $(1,1,\text{left})$

Actions: $\{\text{clean}, \text{move}\}$

Successor function:
$$\text{succ}((x, y, p)) = \begin{cases} \{(x,0,\text{right}), (x, y, \text{left})\} & \text{if } p = \text{right} \\ \{(0,y, \text{left}), (x, y, \text{right})\} & \text{otherwise.} \end{cases}$$

Goal function:
$$\text{goal}(x, y, p) = \begin{cases} 1 & \text{if } x = y = 0 \\ 0 & \text{otherwise.} \end{cases}$$

Cost function:
$$\text{cost}\left((x, y, p), (x', y', p')\right) = 1$$

Solving Search Problems, informally

1. Consider each **start state**
2. Consider every state that can be **reached** from some state that has been previously considered (and remember how to reach the state)
3. **Stop** when you encounter a **goal state**, output plan for reaching the state

Directed Graphs

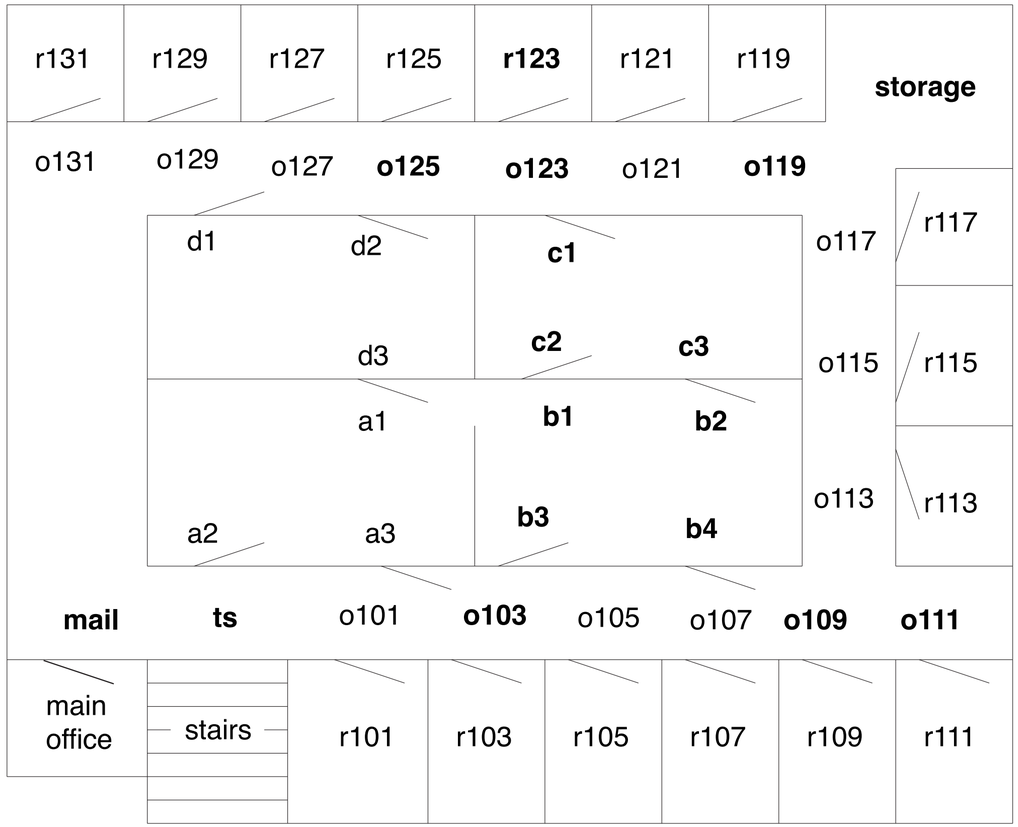
- A **directed graph** is a pair $G = (N, A)$
  - N is a set of **nodes**
  - A is a set of ordered pairs called **arcs**
- Node n_2 is a **neighbour** of n_1 if there is an arc from n_1 to n_2
  - i.e., $\langle n_1, n_2 \rangle \in A$
- A **path** is a sequence of nodes $\langle n_0, n_1, \ldots, n_k \rangle$ with $\langle n_{i-1}, n_i \rangle \in A$
  - **Length** of a path is number of **arcs** (not nodes)

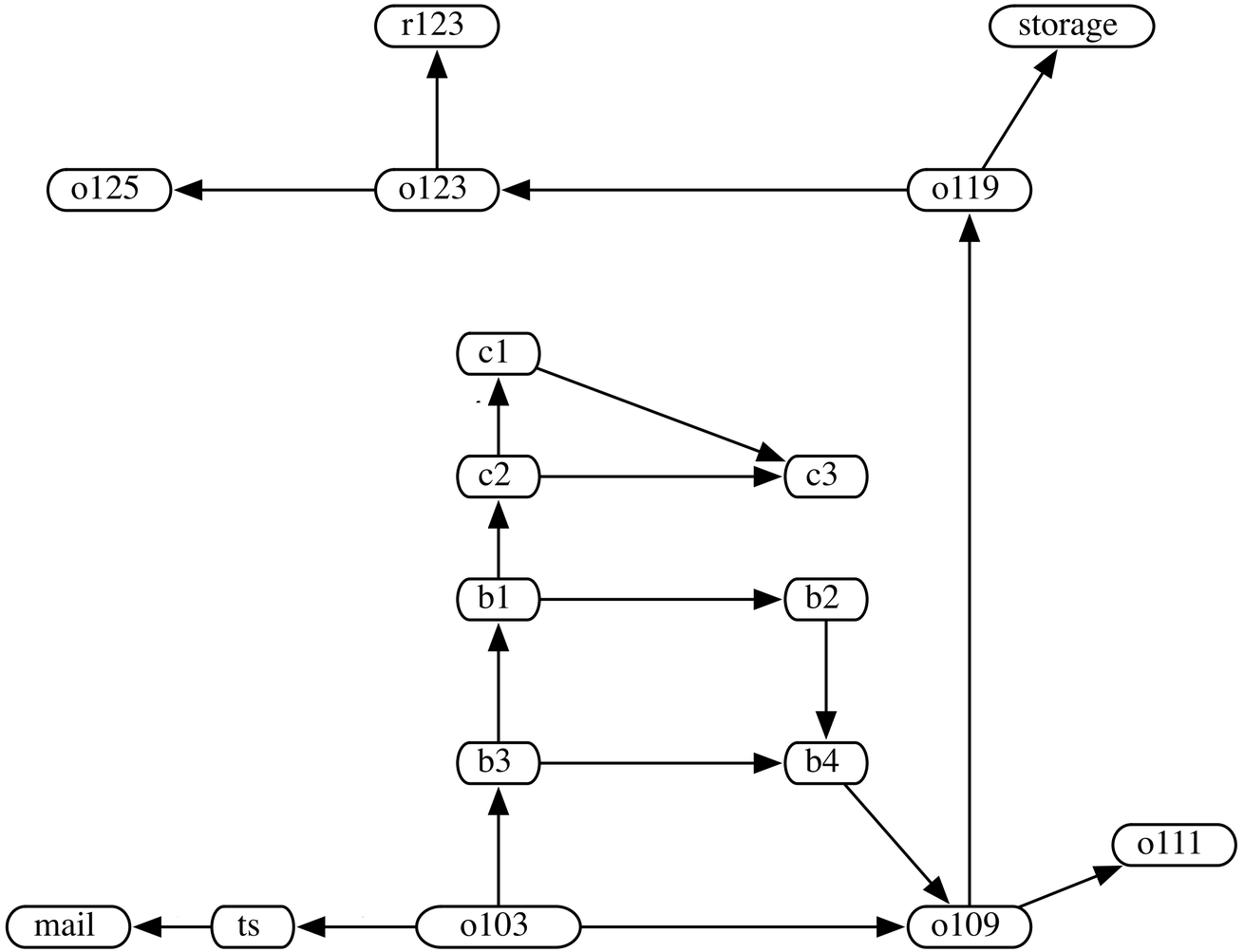
Search Graph

We can represent any search problem as a **search graph**:

1. Nodes are the **states**
2. Neighbours are the **successors** of a state
   - i.e., add one **arc** from state s to each of s's **successors**
3. A **solution** is a path $\langle n_0, n_1, \ldots, n_k \rangle$ from a **start node** to a **goal node**
4. Label each arc with the **cost** for transitioning to the successor state
5. *Optional:* Label each arc with the **action** that leads to the successor state
   - **Question:** Why is this optional?

DeliveryBot: Search Graph

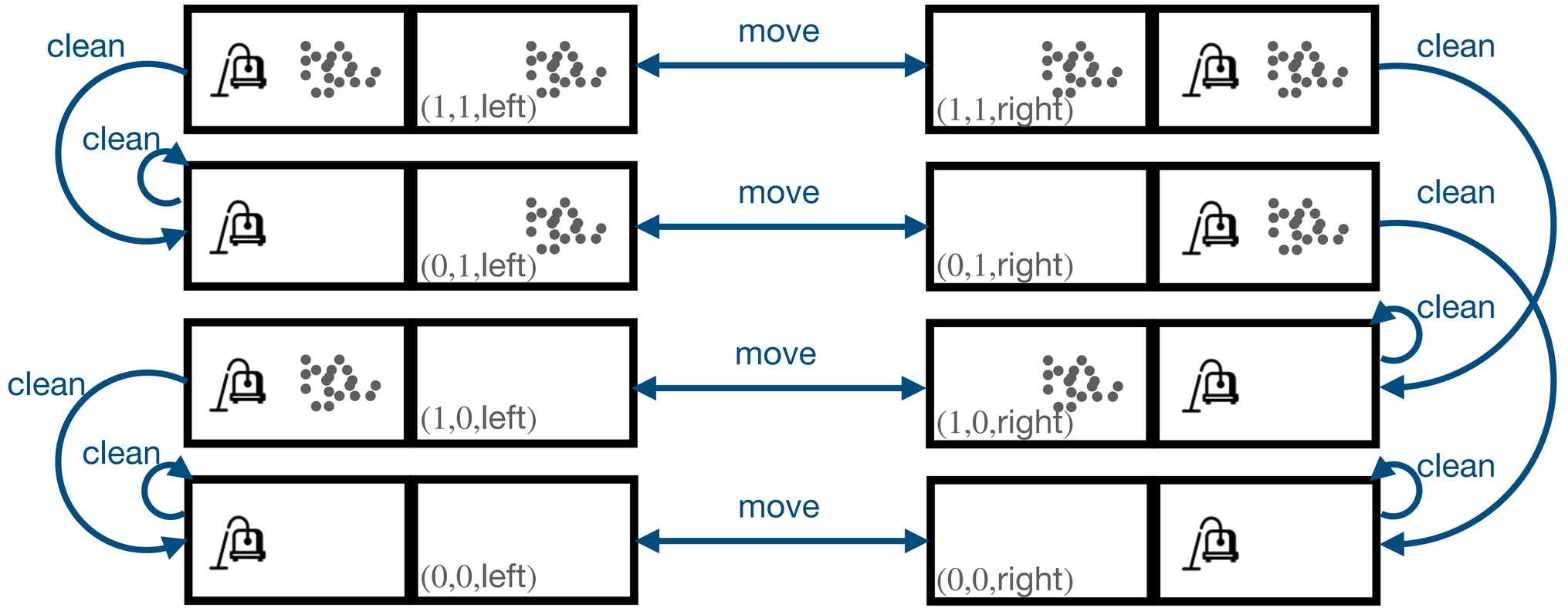




https://artint.info/2e/html/ArtInt2e.Ch3.S2.html

https://artint.info/2e/html/ArtInt2e.Ch3.S3.SS1.html

VacuumBot: Search Graph



A visual description of eight possible states.
There are two rooms; each can be either dirty or clean, and the robot can be in either the left or the right room.
The state with both rooms clean and robot in left room, and both rooms clean and robot in right room, are each outlined in red.
Each state has two outgoing edges, one labeled "clean" and one labeled "move".
From a state where the robot is in a dirty room, the "clean" edge points to a state that is identical except that the room is now clean.
From a state where the robot is in a clean room, the "clean" edge points back to the same state.
Each state has a "move" edge that points to a state that is identical except that the robot is in the opposite room.

icons by Icons8

VacuumBot: Search Graph

$$V = \{(0,0,\text{left}), (0,1,\text{left}), (1,0,\text{left}), (1,1,\text{left}), (0,0,\text{right}), (0,1,\text{right}), (1,0,\text{right}), (1,1,\text{right})\}$$

$$A = \{\langle (x,y,p), (x',y',p') \rangle \mid (x',y',p') = f(x,y,p) \vee (x',y',p') = g(x,y,p)\}$$

$$f(x,y,p) = \begin{cases} (0,y,p) & \text{if } p = \text{left} \\ (x,0,p) & \text{if } p = \text{right} \end{cases}$$

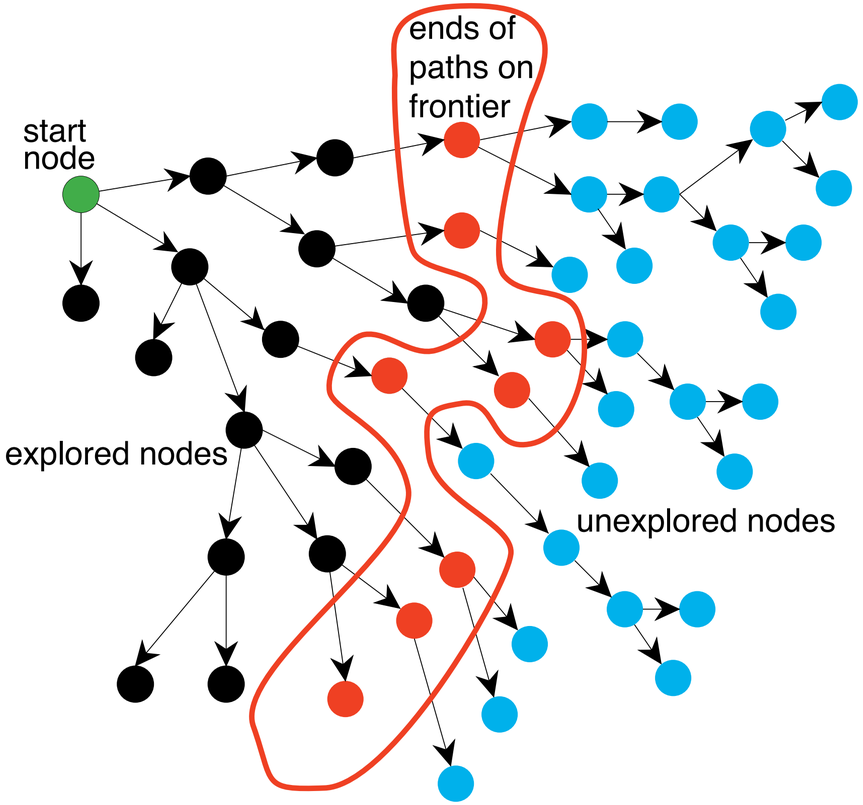
$$g(x,y,p) = \begin{cases} (x,y,\text{right}) & \text{if } p = \text{left} \\ (x,y,\text{left}) & \text{if } p = \text{right} \end{cases}$$

$$\text{goal}(x,y,p) = (x = 0 \wedge y = 0)$$

$$\text{cost}(v_1, v_2) = 1$$

Generic Graph Search Algorithm

- Given a graph, start nodes, and goal, incrementally explore paths from the start nodes
- Maintain a **frontier** of **paths** that have been explored
- As search proceeds, the frontier **expands** into the unexplored nodes until a goal is encountered.
- The **way** the frontier is expanded defines the **search strategy**



https://artint.info/2e/html/ArtInt2e.Ch3.S4.html

Generic Graph Search Algorithm

Input: a *graph*; a set of *start nodes*; a $goal$ function

$frontier := \{\langle s \rangle \mid s \text{ is a start node}\}$
while $frontier$ is not empty:
    select a path $\langle n_0, \ldots, n_k \rangle$ from $frontier$
    remove $\langle n_0, \ldots, n_k \rangle$ from $frontier$
    if $goal(n_k)$:
        return $\langle n_0, \ldots, n_k \rangle$
    for each neighbour n of n_k:
        add $\langle n_0, \ldots, n_k, n \rangle$ to $frontier$
end while

Search Problem with Costs

What if solutions have differing qualities?

- Add **costs** to each arc: $\text{cost}(n_{i-1}, n_i)$
- **Cost of a solution** is the sum of the arc costs:

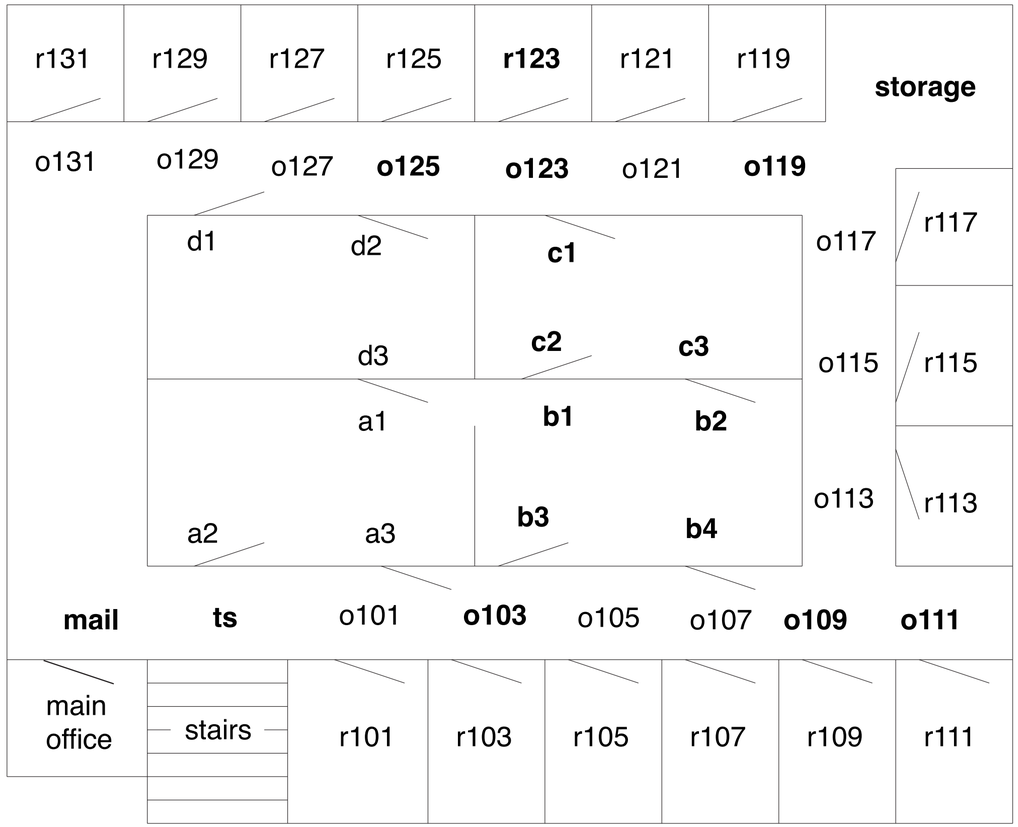
$$\text{cost}\left(\langle n_0, n_1, \ldots, n_k \rangle\right) = \sum_{i=1}^{k} \text{cost}(n_{i-1}, n_i)$$

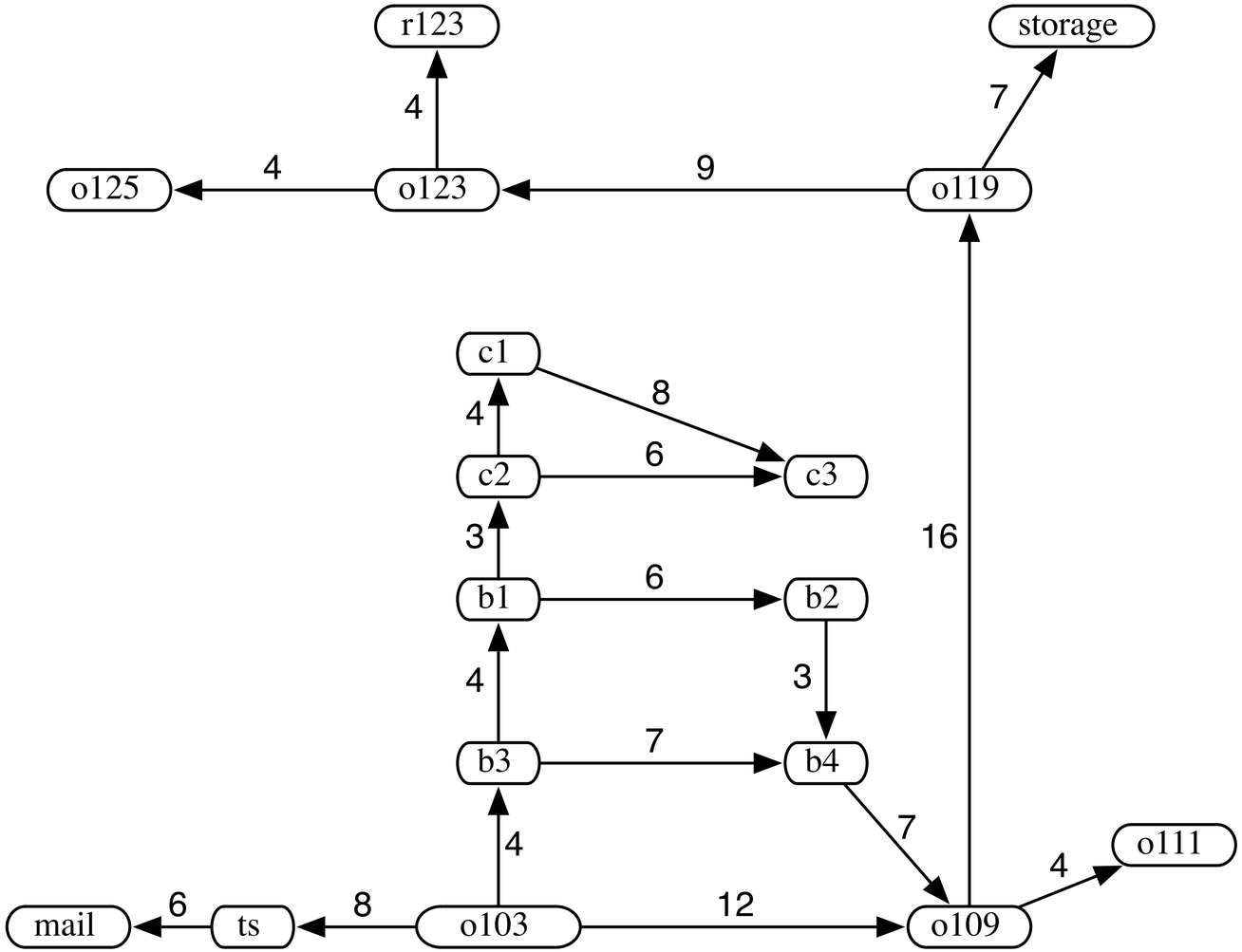
- An **optimal solution** is one with the lowest cost

Questions:

1. Is this scheme sufficiently **general**?
2. What if we only care about the **number of actions** that the agent takes?
3. What if we only care about the **quality** of the end state (i.e., we don't care about the actions)?

DeliveryBot with Costs





https://artint.info/2e/html/ArtInt2e.Ch3.S2.html

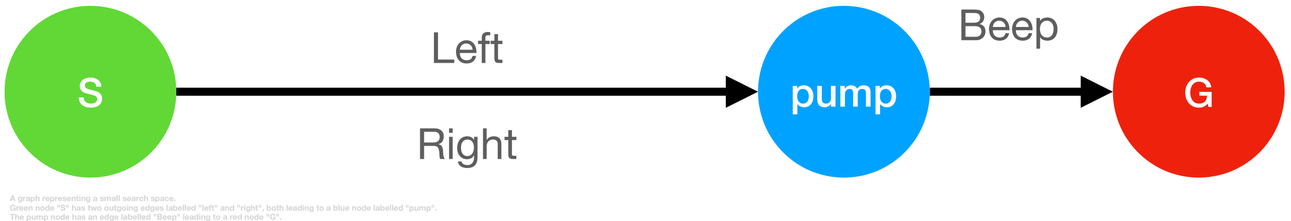
https://artint.info/2e/html/ArtInt2e.Ch3.S3.SS1.html

Markov Assumption

- *Informally:*
  How the environment arrived at the current configuration "doesn't matter"
- **Question:** What does "doesn't matter" mean *formally?*
- Edge costs, available actions, neighbourhoods, all depend only on **starting state** (and maybe action)
  - NOT on "sequence of edges that led to the current state"
- Mathematically, this means that each of these is a **function of** the **state** not the **history**
  - E.g., defining costs as $\mathbf{cost}(s, z)$ instead of $\mathbf{cost}(\langle n_0, n_1, n_2, s \rangle, z)$ **guarantees** that the representation satisfies the Markov assumption (with respect to costs)

Markov Assumption: GasBot

The **Markov assumption** is **crucial** to the graph search algorithm



A graph representing a small search space.
Green node "S" has two outgoing edges labelled "left" and "right", both leading to a blue node labelled "pump".
The pump node has an edge labelled "Beep" leading to a red node "G".

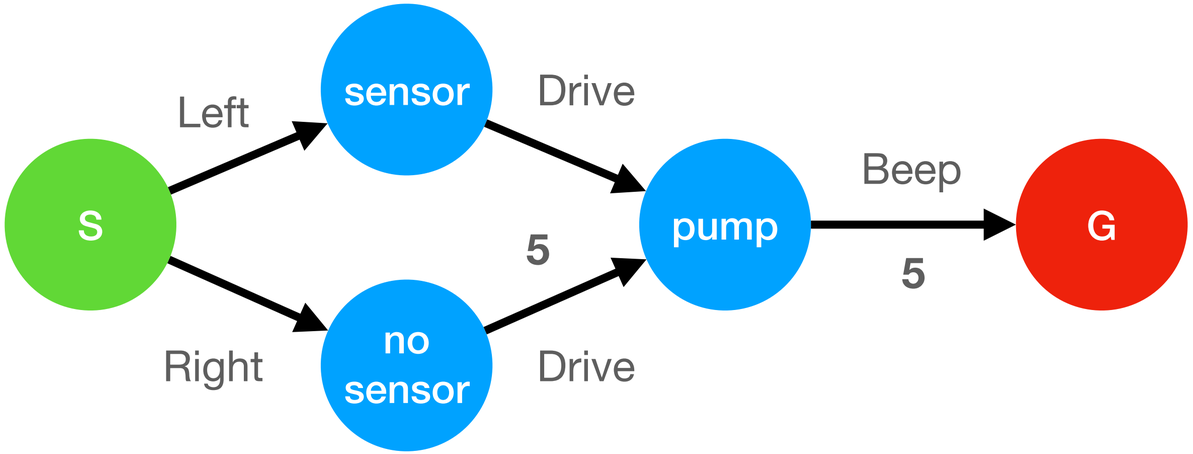
cost(pump, gas):
5 if went through sensor
10 otherwise

Getting to the pump:
from the **left** goes through sensor
from the **right** does not

Question: Does this representation representation satisfy the Markov assumption? Why or why not?

Markov Assumption: GasBot

The **Markov assumption** is **crucial** to the graph search algorithm



A graph representing a small search space.
Green node "S" has two outgoing edges/.
The edge labelled "Left" leads to a blue node "Sensor".
The edge labelled "Right" leads to a blue node "No sensor".
The "no sensor" node has a single edge labelled "Drive" with a cost 5 leading to a node "Pump".
The "sensor" node has an edge labelled "Drive" leading the same "Pump" node.
The pump node has an edge labelled "Beep" with a cost 5 leading to a red node "G".

Questions

1. Does this representation satisfy the Markov assumption? Why or why not?
2. How else could we have fixed up the previous example?

Summary

- Many AI tasks can be represented as **search problems**
  - A single generic **graph search algorithm** can then solve them all!
- A search problem consists of **states**, **actions**, **start states**, a **successor function**, a **goal** function, optionally a **cost** function
- **Solution quality** can be represented by labelling arcs of the search graph with **costs**
- The **Markov assumption** is critical for graph search to work