

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 **T BW-1**

- In person

eClass: <https://eclass.srv.ualberta.ca/course/view.php?id=81622>

- 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

- There is **no lab** for this course
- TA office hours instead (announced next class)

Lecture Outline

1. Recap
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
 - All computation is purely internal to the agent.
 - Outcomes are **known** and **deterministic**, so no need for observations
- 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

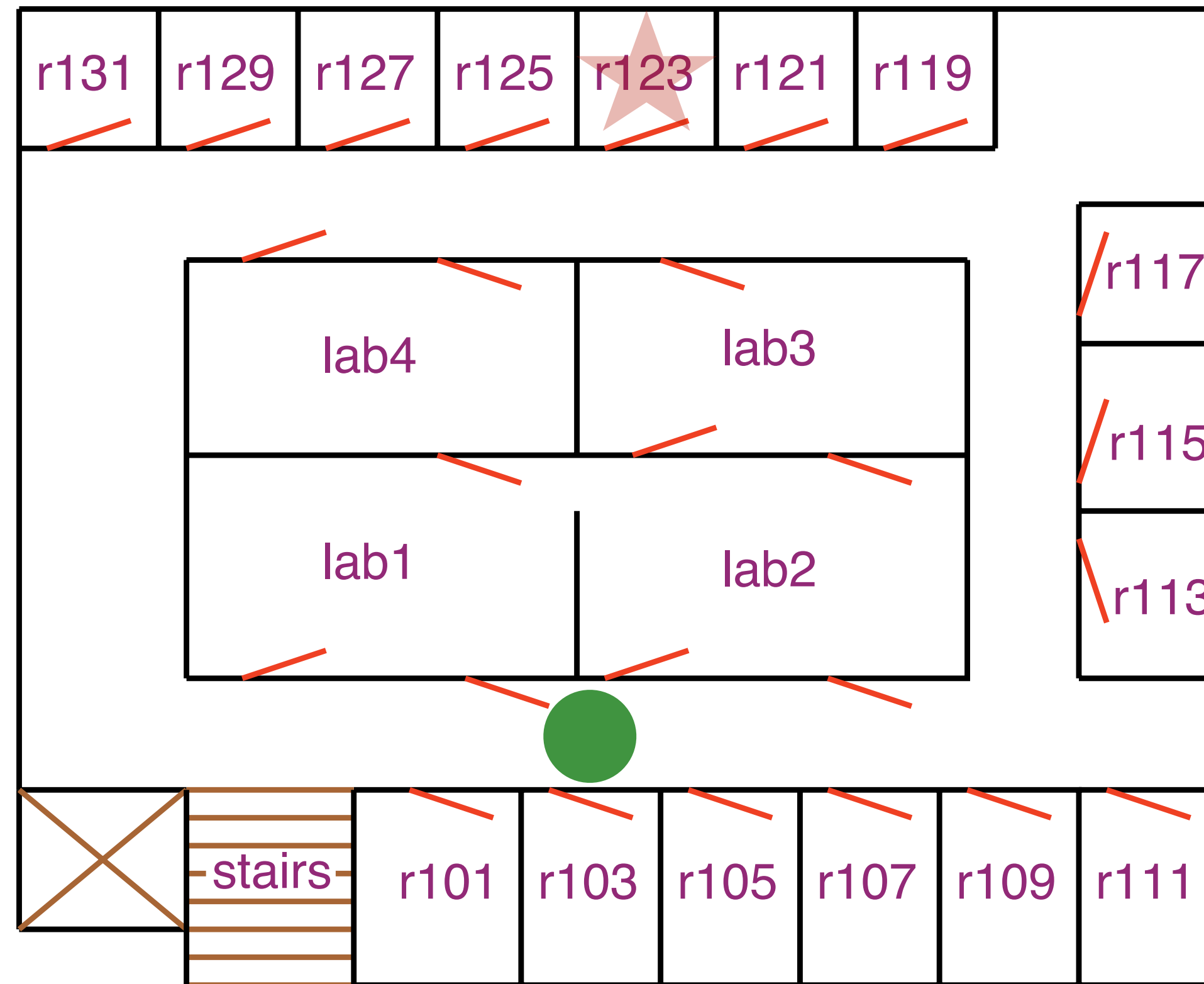
Search Problem

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 neighbour 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

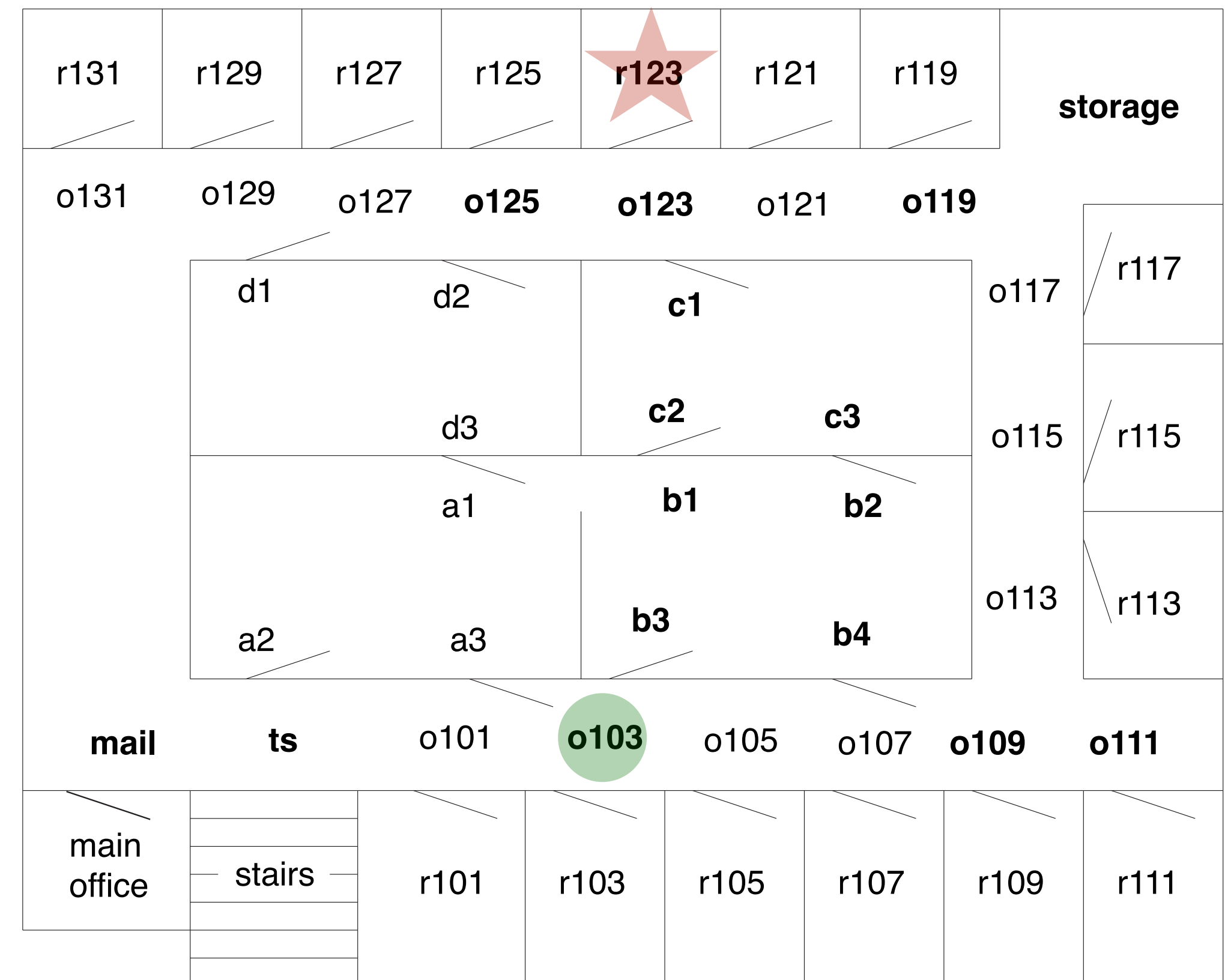
Example: DeliveryBot

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



DeliveryBot as a Search Problem

States	{r131, o131, r129, o129, ...}
Actions	{go-north, go-south, go-east, go-west}
Start state	o103
Successor function	$\text{succ}(r101) = \{r101, o101\}$, $\text{succ}(o101) = \{o101, \text{lab1}, r101, o105, \text{ts}\}$, ...
Goal function	goal(state): (state == r123)



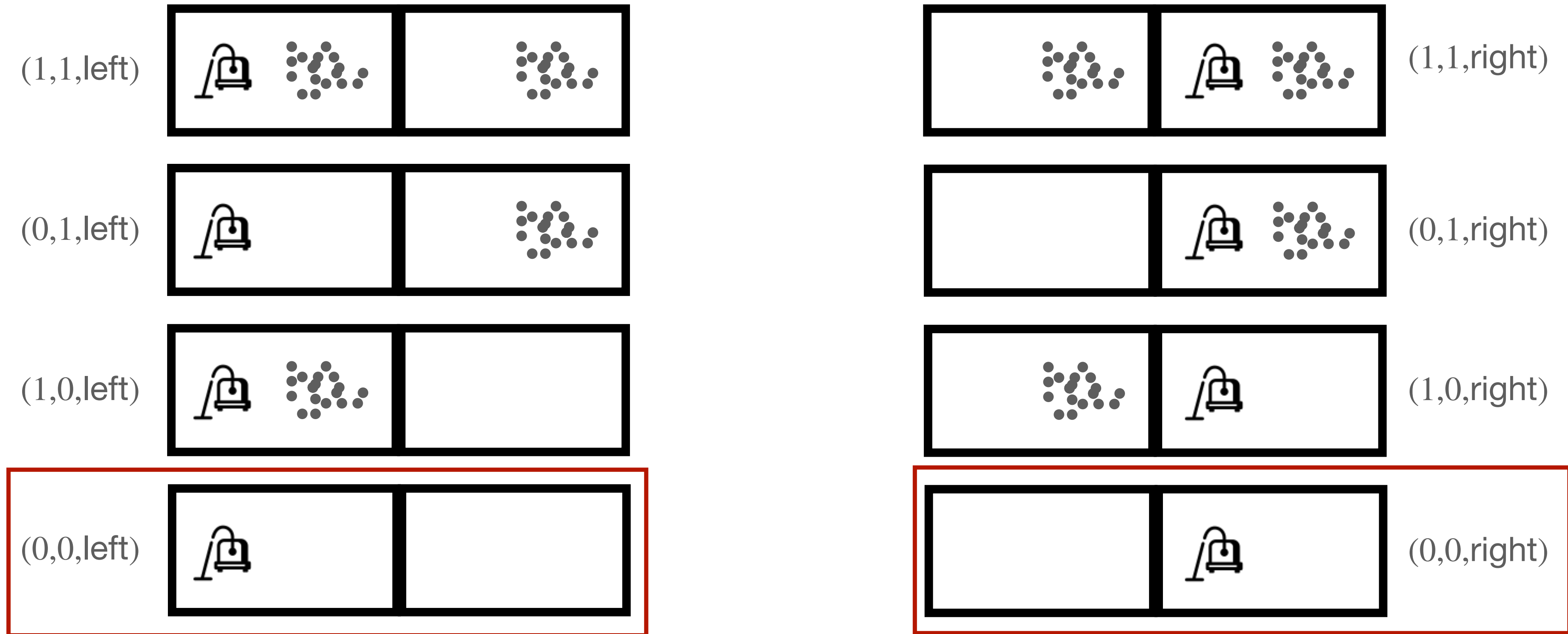
Example: VacuumBot

- 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

Questions:

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

VacuumBot as a Search Problem: States



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

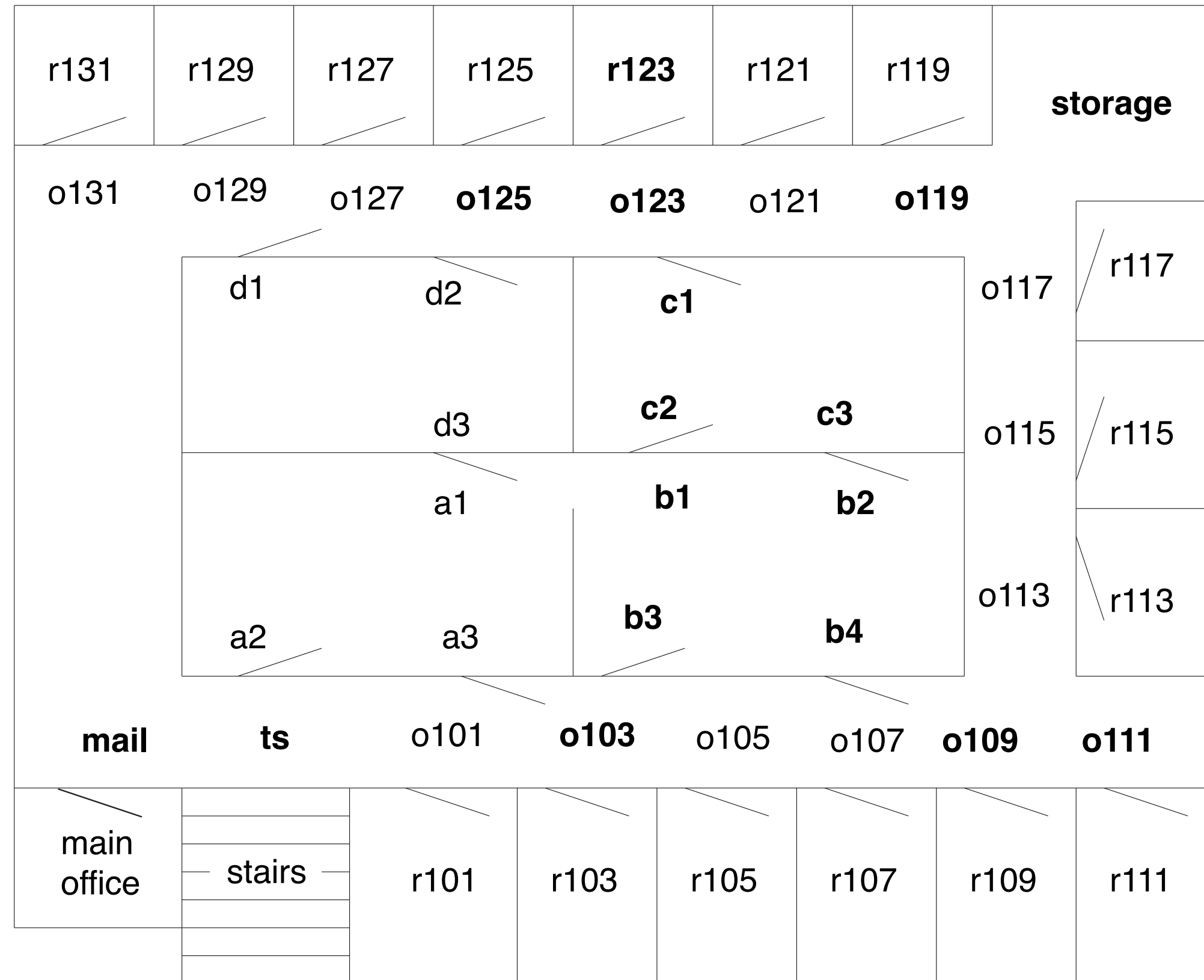
- A **directed graph** is a pair $G = (N, A)$
 - N is a set of **nodes**
 - A is a set of ordered pairs called **arcs**
- Node n_2 is a **neighbour** of n_1 if there is an arc from n_1 to n_2
 - i.e., $\langle n_1, n_2 \rangle \in A$
- A **path** is a sequence of nodes $\langle n_0, n_1, \dots, n_k \rangle$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

Search Graph

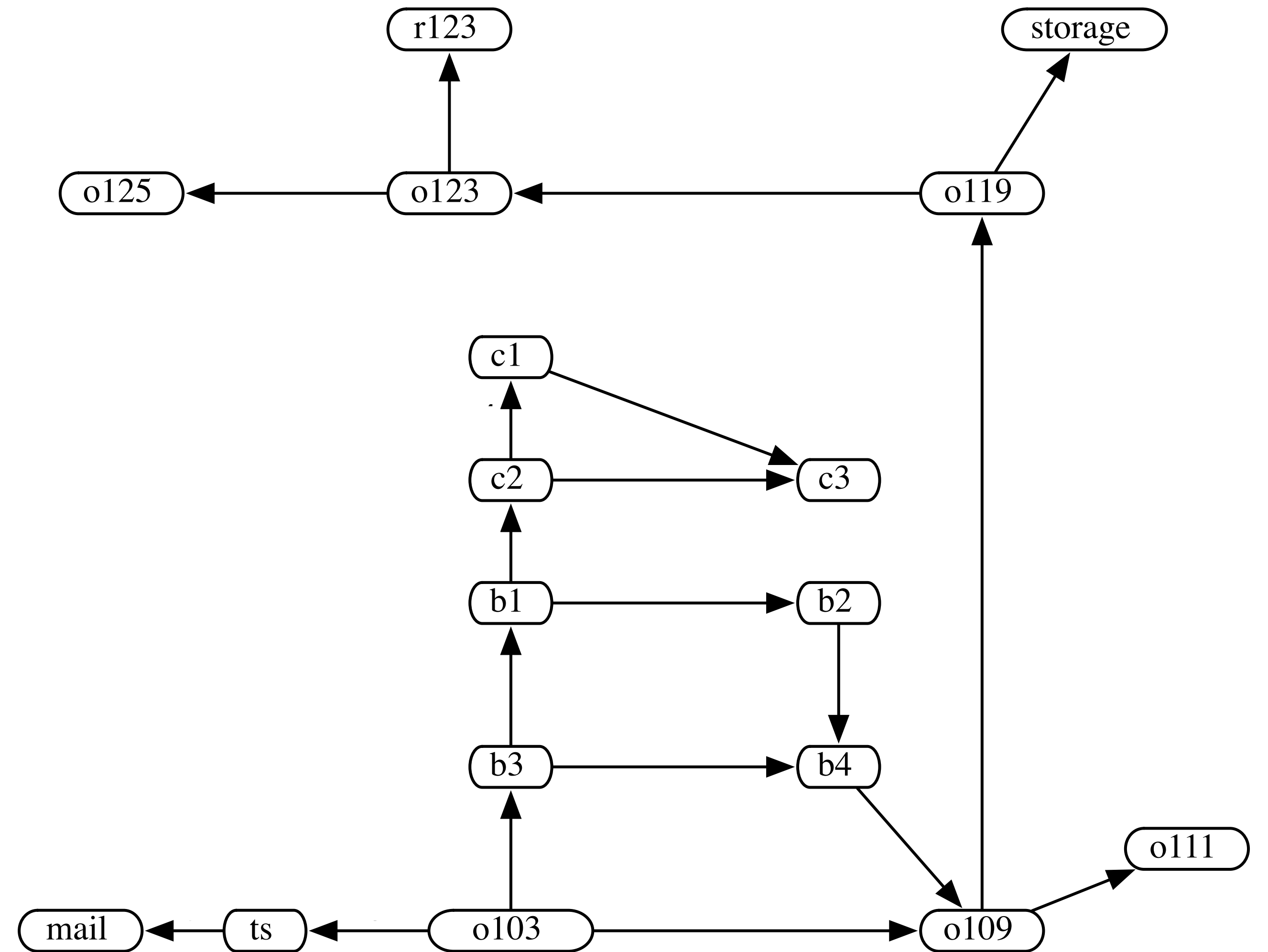
We can represent any search problem as a **search graph**:

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 $\langle n_0, n_1, \dots, n_k \rangle$ 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

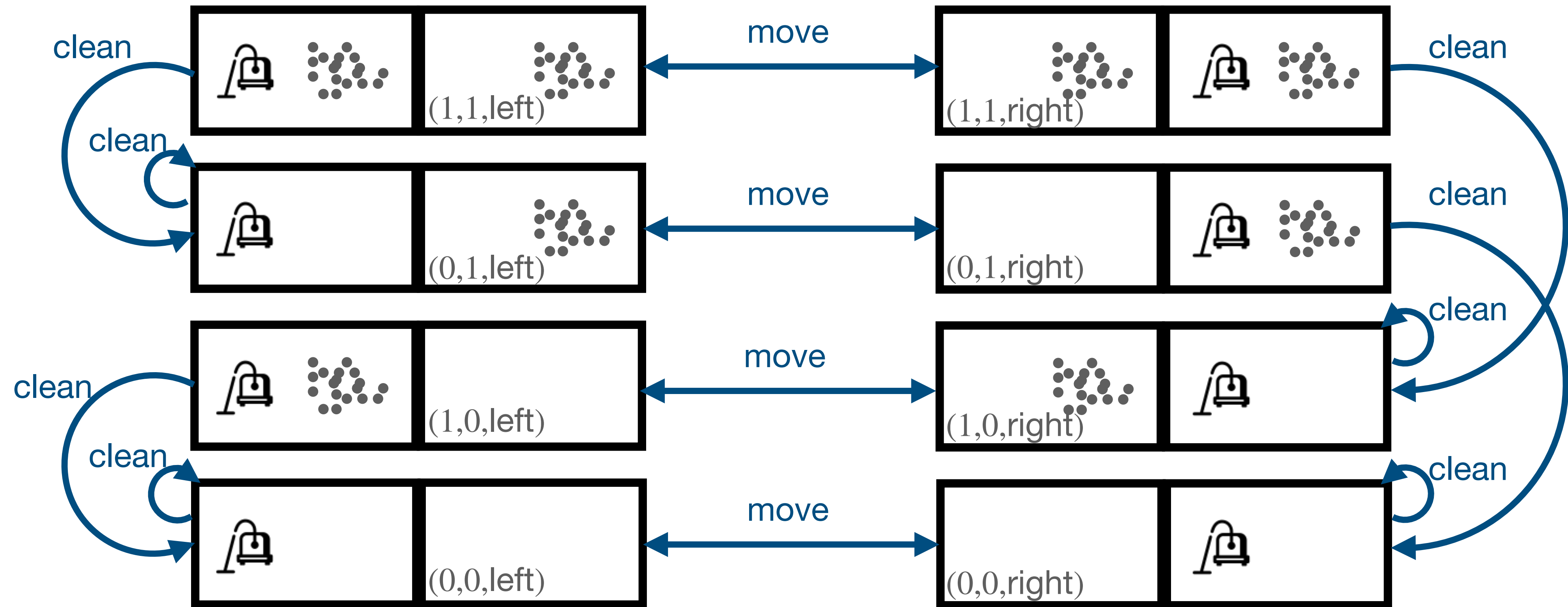


<https://artint.info/2e/html/ArtInt2e.Ch3.S2.html>



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

VacuumBot: Search Graph



VacuumBot: Search Graph

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

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

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

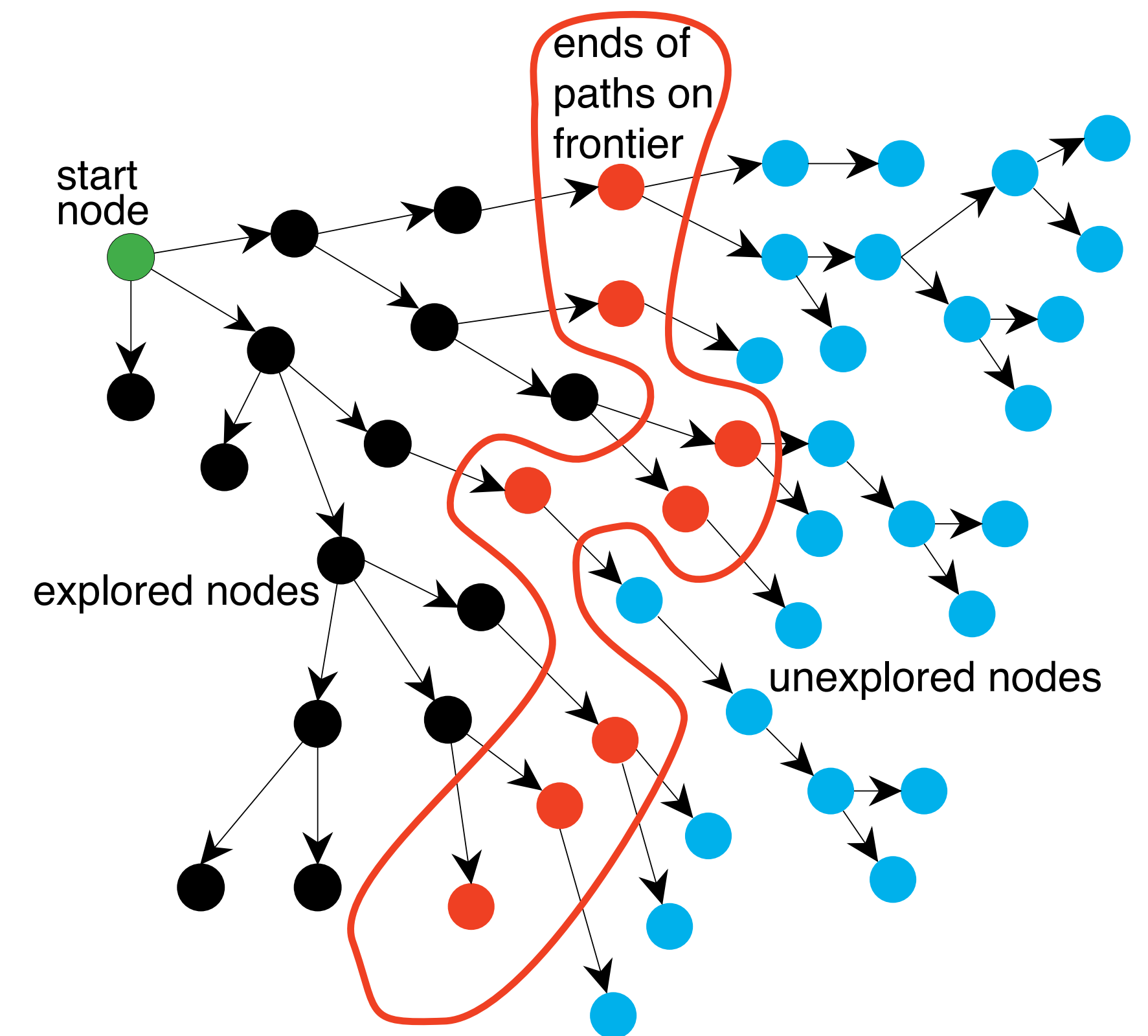
$$g(x, y, p) = \begin{cases} (x, y, \text{right}) & \text{if } p = \text{left} \\ (x, y, \text{left}) & \text{if } p = \text{right} \end{cases}$$

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

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

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**



Generic Graph Search Algorithm

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

frontier := { $\langle s \rangle$ | *s* is a start node }

while *frontier* is not empty:

select a path $\langle n_0, \dots, n_k \rangle$ from *frontier*

remove $\langle n_0, \dots, n_k \rangle$ from *frontier*

 if *goal*(n_k):

return $\langle n_0, \dots, n_k \rangle$

for each neighbour *n* of n_k :

add $\langle n_0, \dots, n_k, n \rangle$ to *frontier*

end while

Search Problem with Costs

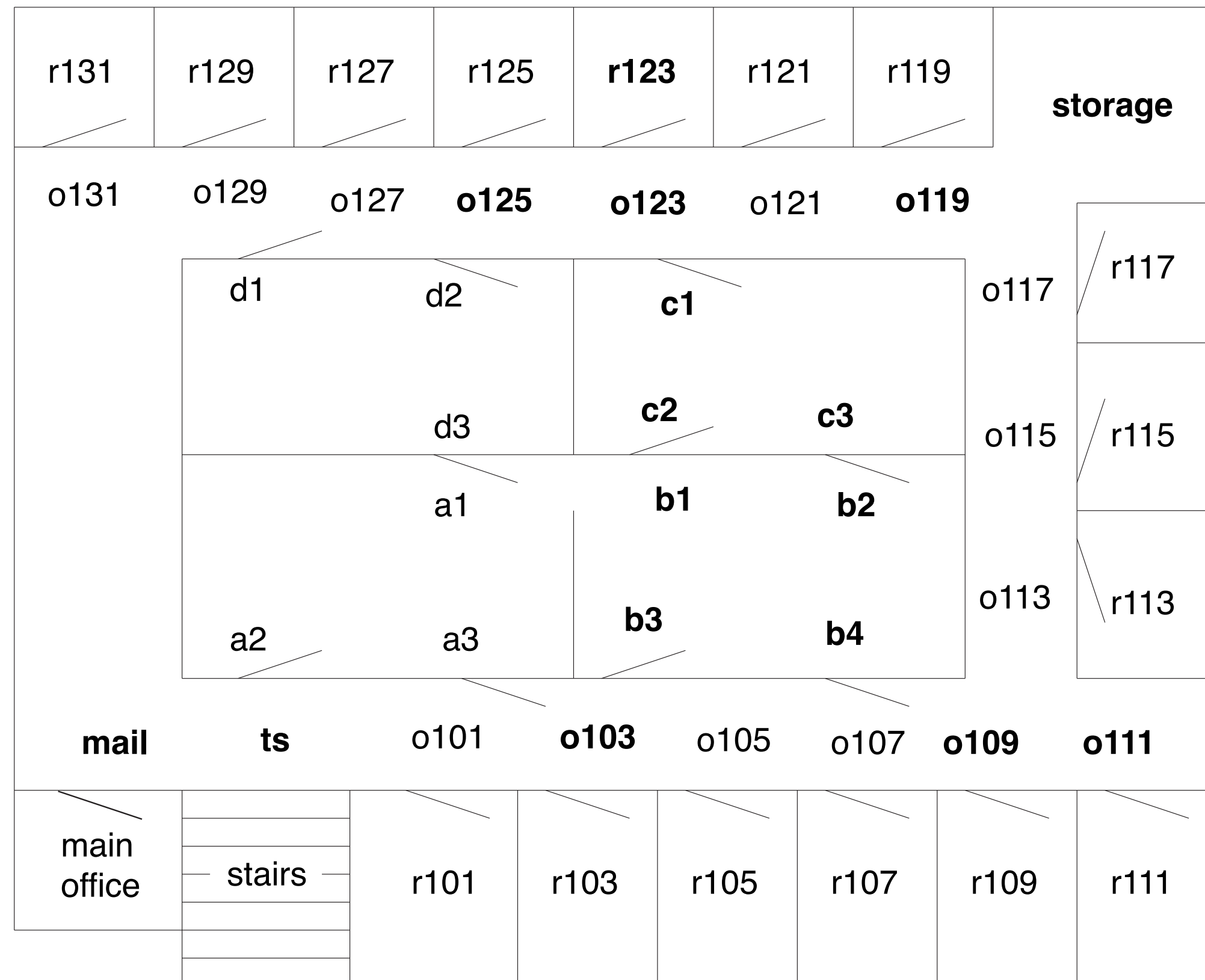
What if solutions have differing qualities?

- Add **costs** to each arc: $\text{cost}(\langle n_{i-1}, n_i \rangle)$
- **Cost of a solution** is the sum of the arc costs:
$$\text{cost}(\langle n_0, n_1, \dots, n_k \rangle) = \sum_{i=1}^k \text{cost}(\langle n_{i-1}, n_i \rangle)$$
- An **optimal solution** is one with the lowest cost

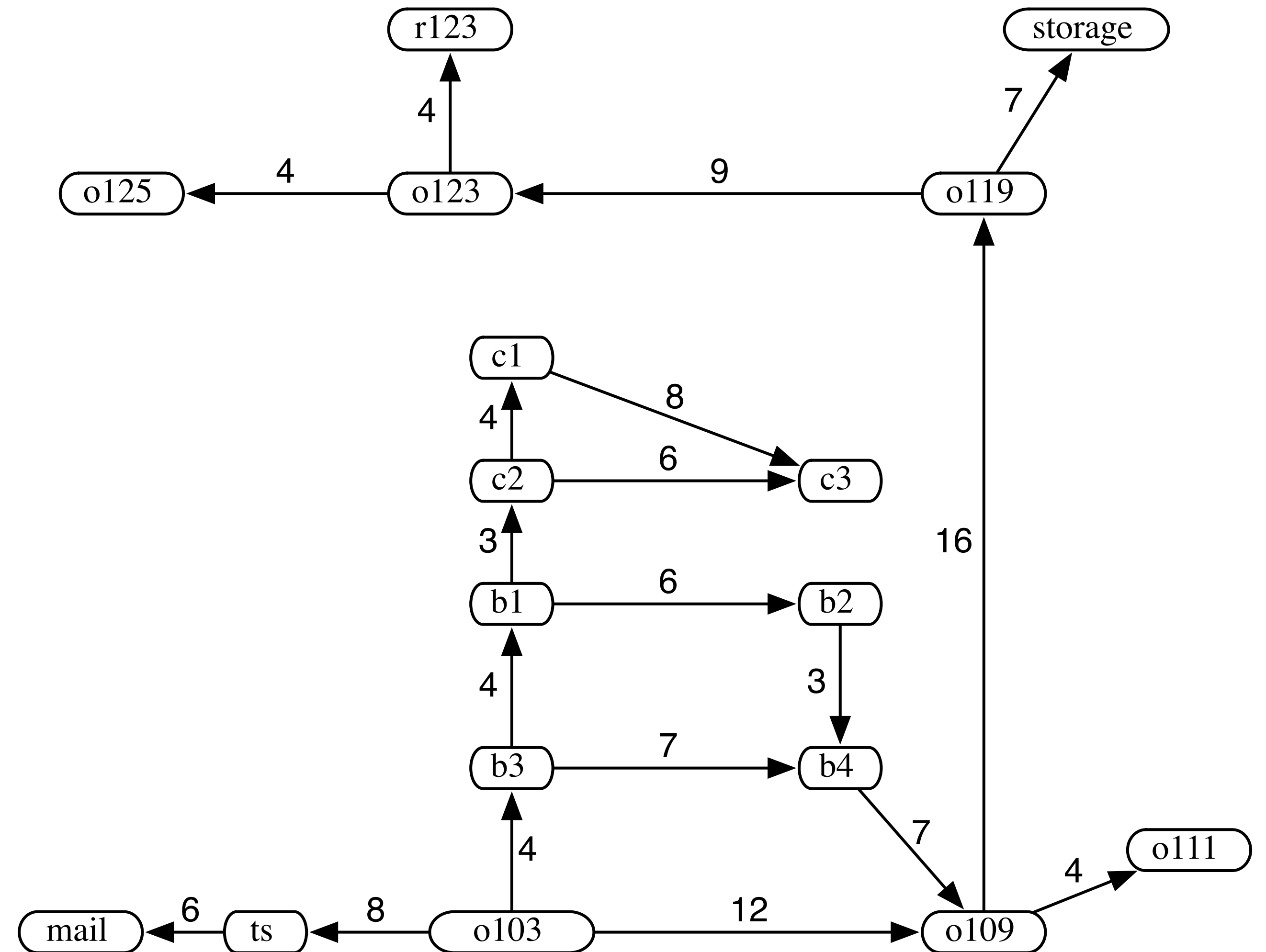
Questions:

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 solution we find?

DeliveryBot with Costs



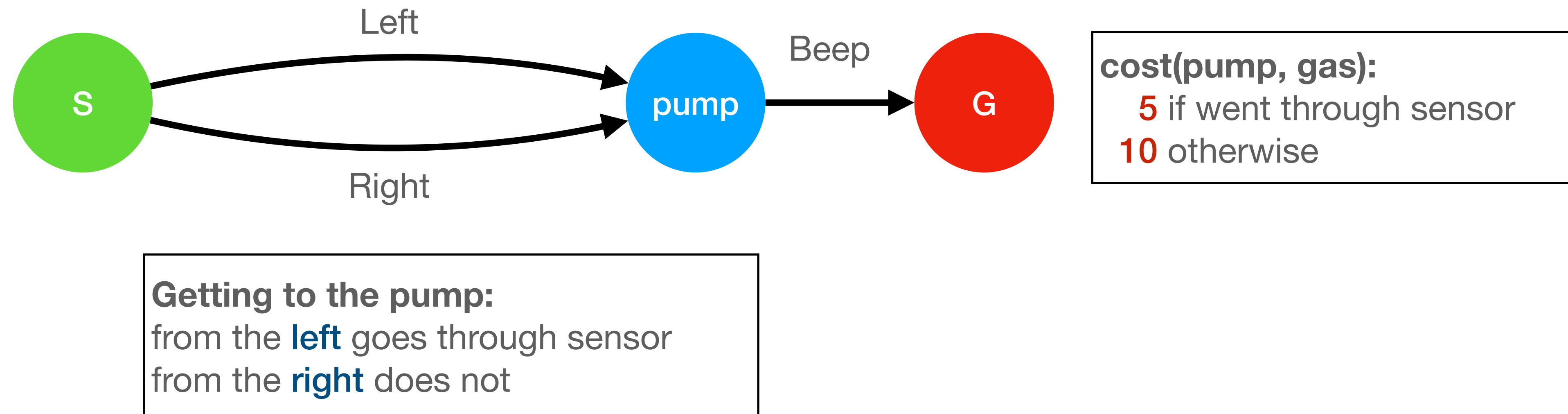
<https://artint.info/2e/html/ArtInt2e.Ch3.S2.html>



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

Markov Assumption: GasBot

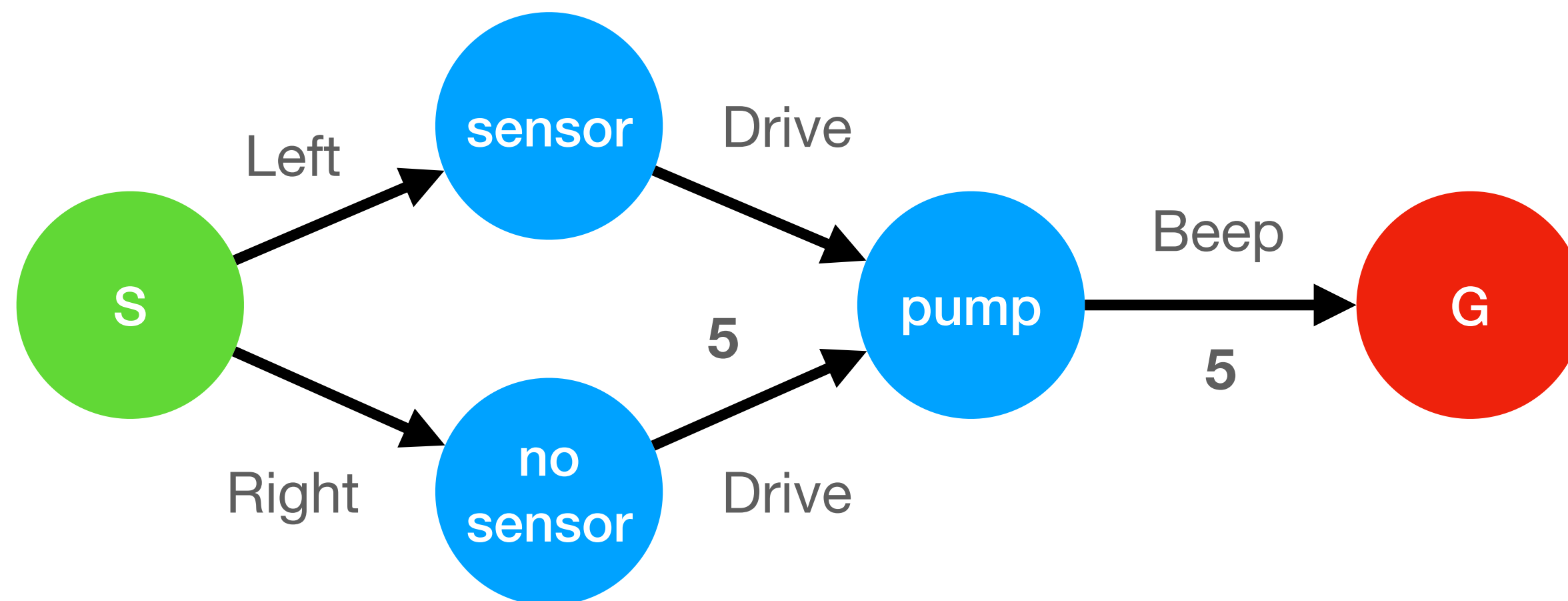
The **Markov assumption** is **crucial** to the graph search algorithm



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

Markov Assumption: GasBot

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



Questions

1. Does this environment 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