Graph Search

CMPUT 366: Intelligent Systems

P&M §3.1-3.4

"Recap": Perfect Rationality vs. Bounded Rationality

Is it feasible for the agent to achieve the **ideal optimum**, or must it trade off **solution quality** against **computational cost**?

- **Perfect rationality:** The agent can derive the best course of action without accounting for **computational limitations**.
- **Bounded rationality:** Agent decides on best action that **it can find** within its computational limitations
    - **Anytime algorithm:** Solution quality improves with time

Recap: Dimensions

- Static vs. sequential action
- Goals vs. complex preferences
- Episodic vs. continuing
- State representation scheme
- Perfect vs. bounded rationality

1. Uncertainty
2. Interaction
3. Number of agents

Different dimensions **interact**; you can't just set them arbitrarily

"Recap": Course Topics Breakdown

Offline
Single agent

Deterministic

Stochastic

Uncertainty
Causality
Supervised Learning
Deep Learning

Search

Multiple agents

Online

Multiagent Systems

Reinforcement Learning

Lecture Outline

1. Recap
2. Search Problems
3. Graph Search

Search

- It is often easier to **recognize** a solution than to **compute** it
- For **fully-observable, deterministic, offline, single-agent** problems, search exploits this property!
- Agent searches **internal representation** to find solution
  - All computation is purely internal to the agent.
  - Environment is fully deterministic, so no need for observations, just remember actions
- 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.
- A state is an assignment of values to one or more **variables**
  - A single variable called "state"
  - *x* and *y* coordinates, temperature, battery charge, etc.
- **Actions** change the environment from one state to another

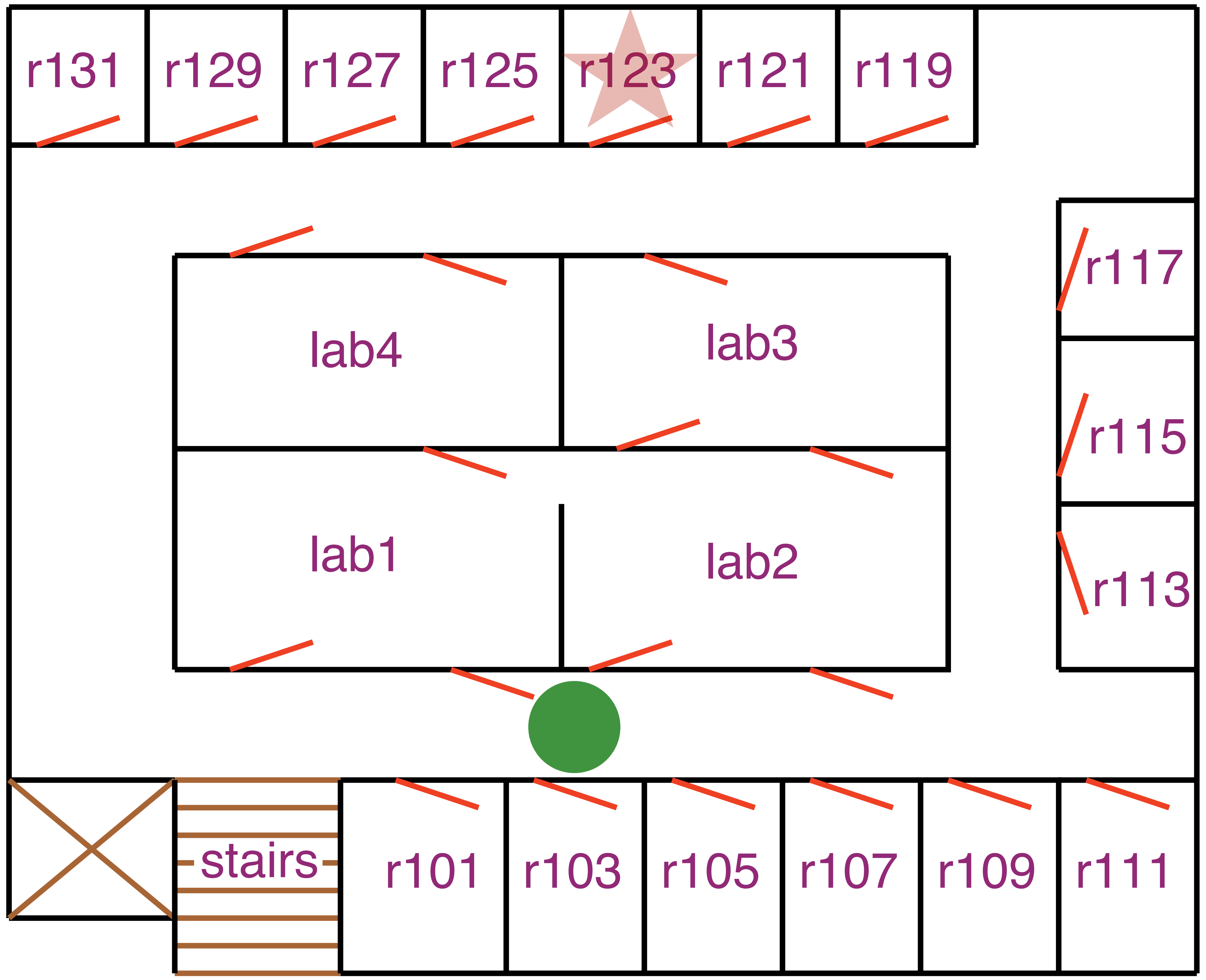
Search Problem

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 next states
  - The textbook calls this an **action function**
- A **goal function** that returns true when a state satisfies the goal

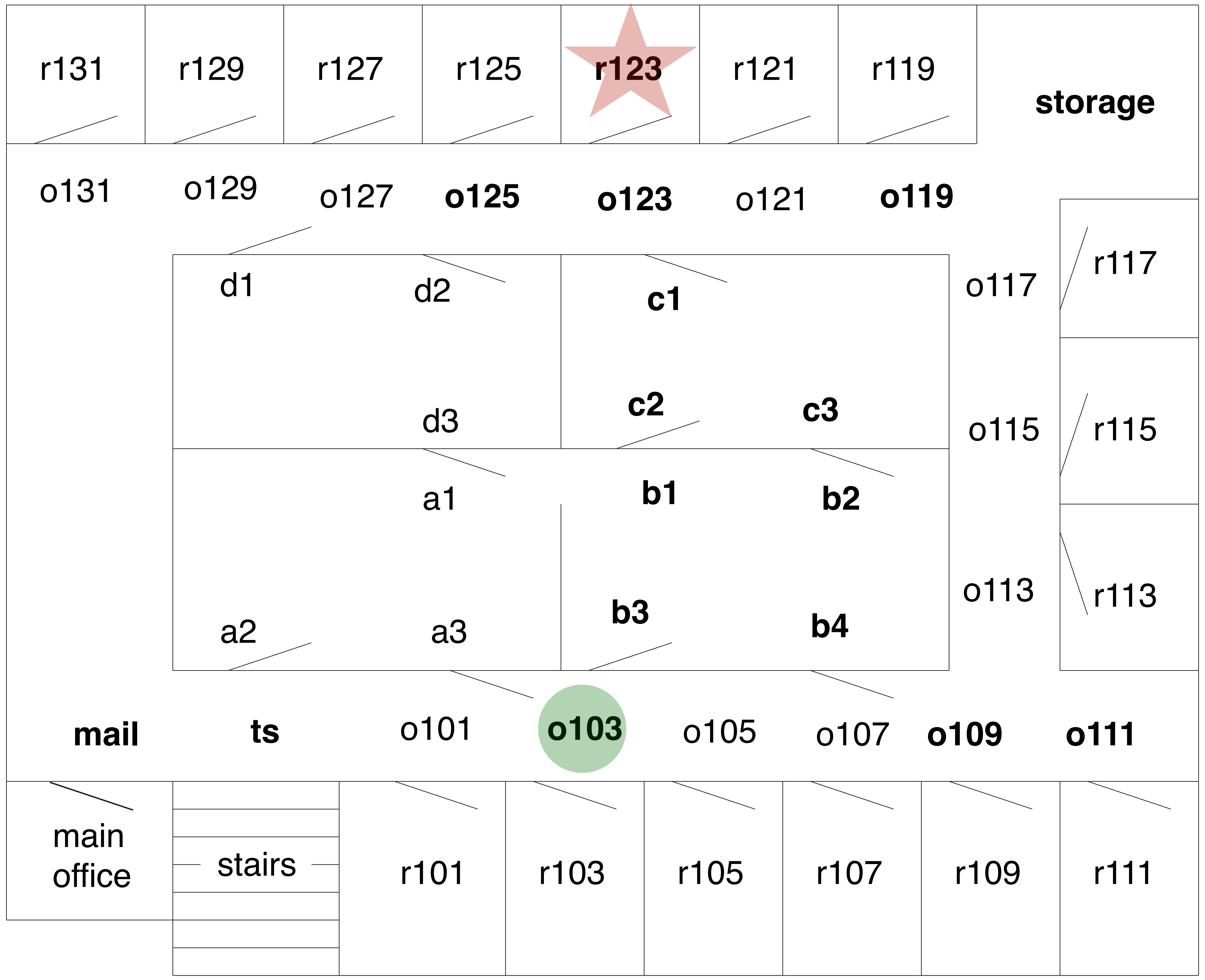
Example: DeliveryBot

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	succ(r101) = {r101, o101}, succ(o101) = {o101, lab1, r101,o105, ts}, ...
Goal function	goal(state): (state == r123)



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

Questions:

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

3. **Stop** when you encounter a **goal 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 $\langle n_1, n_2, \ldots, n_k \rangle$ with $\langle n_{i-1}, n_i \rangle \in A$
- A **solution** is a path $\langle n_1, n_2, \ldots, n_k \rangle$ from a **start node** to a **goal node**

Search Graph

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

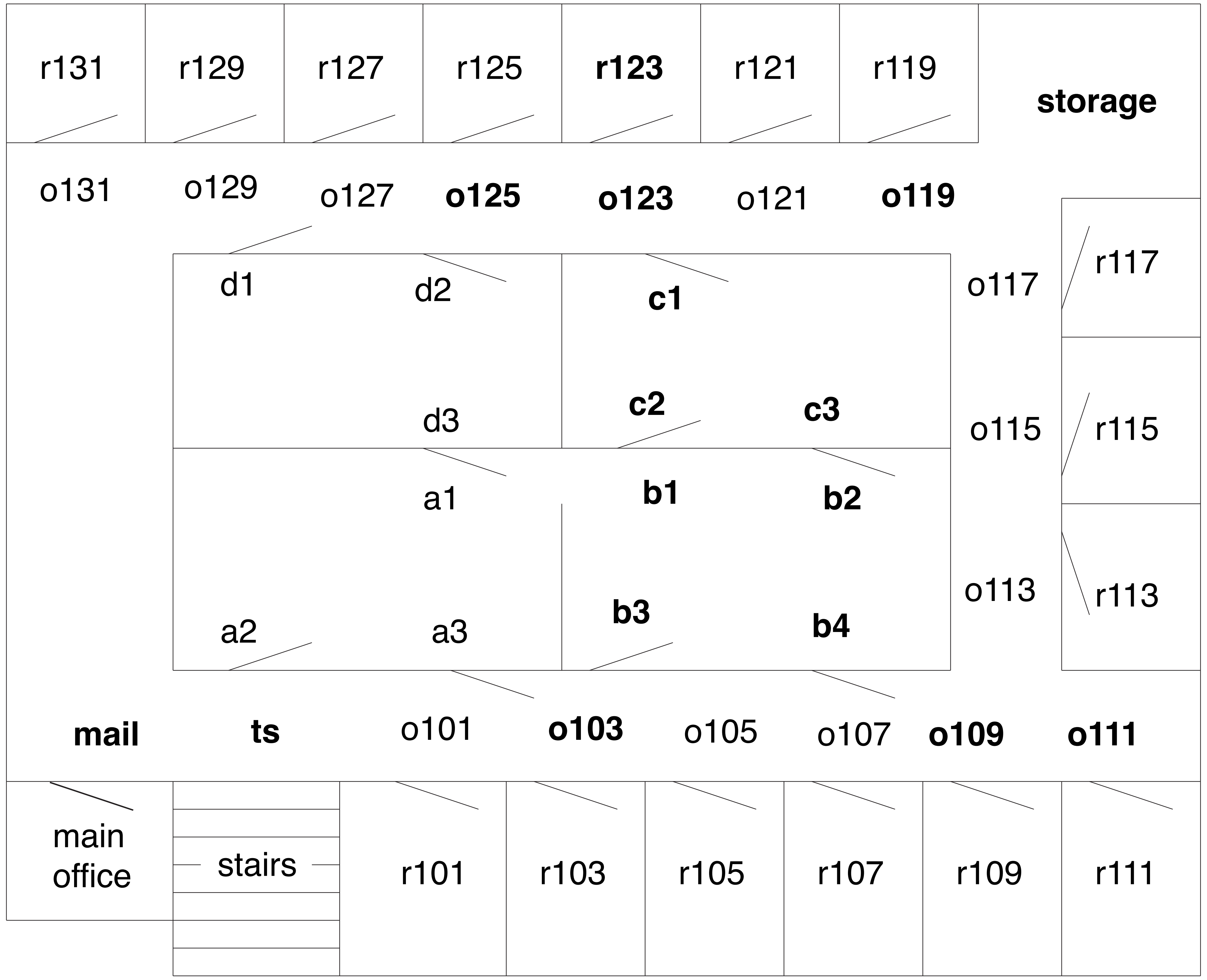
1. Nodes are the **states**

2. Neighbours are the **successors** of a state

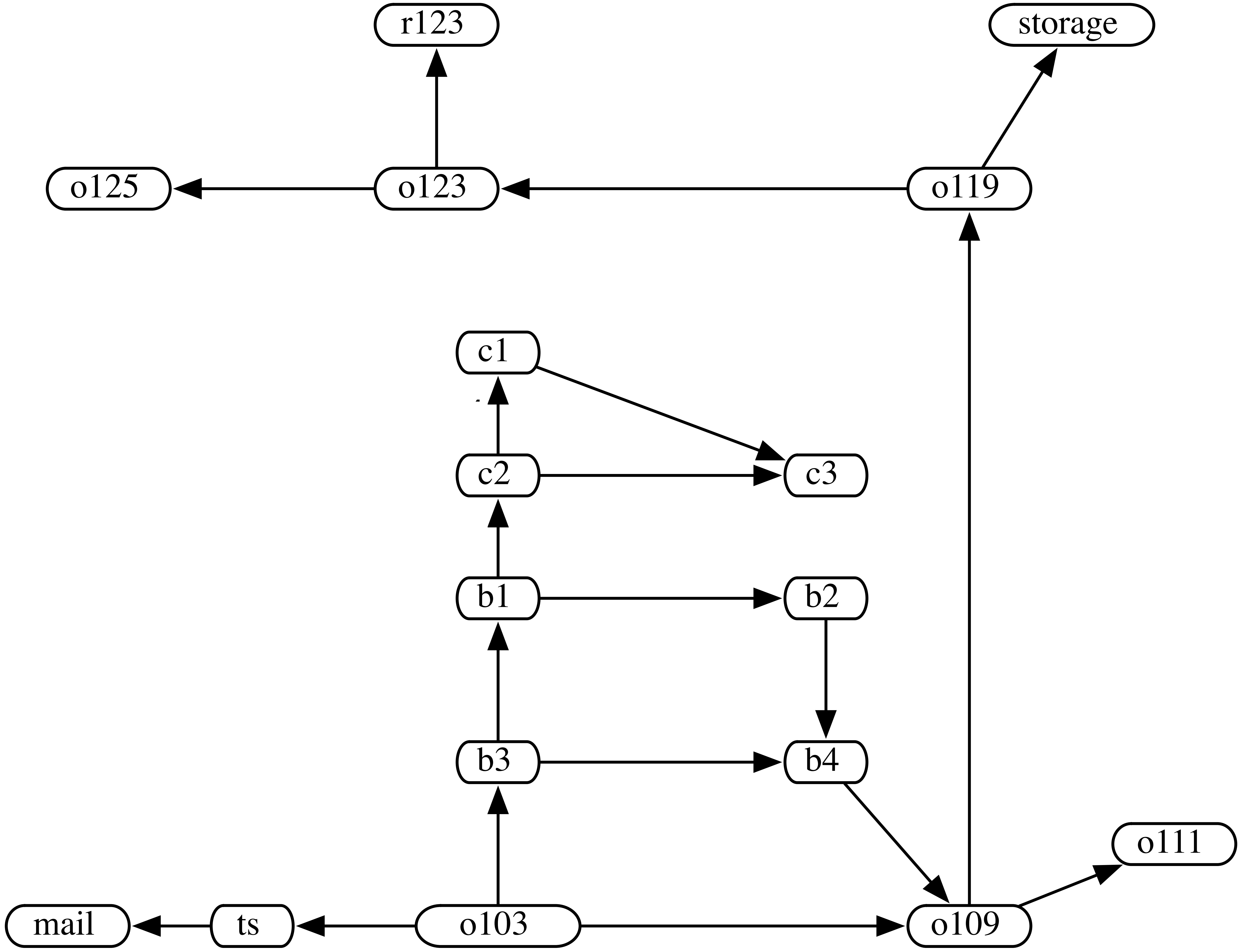
    - i.e., add one **arc** from state s to each of s's **successors**

3. *Optional:* Label each arc with the **action** that leads to the successor state

DeliveryBot: State Space Graph



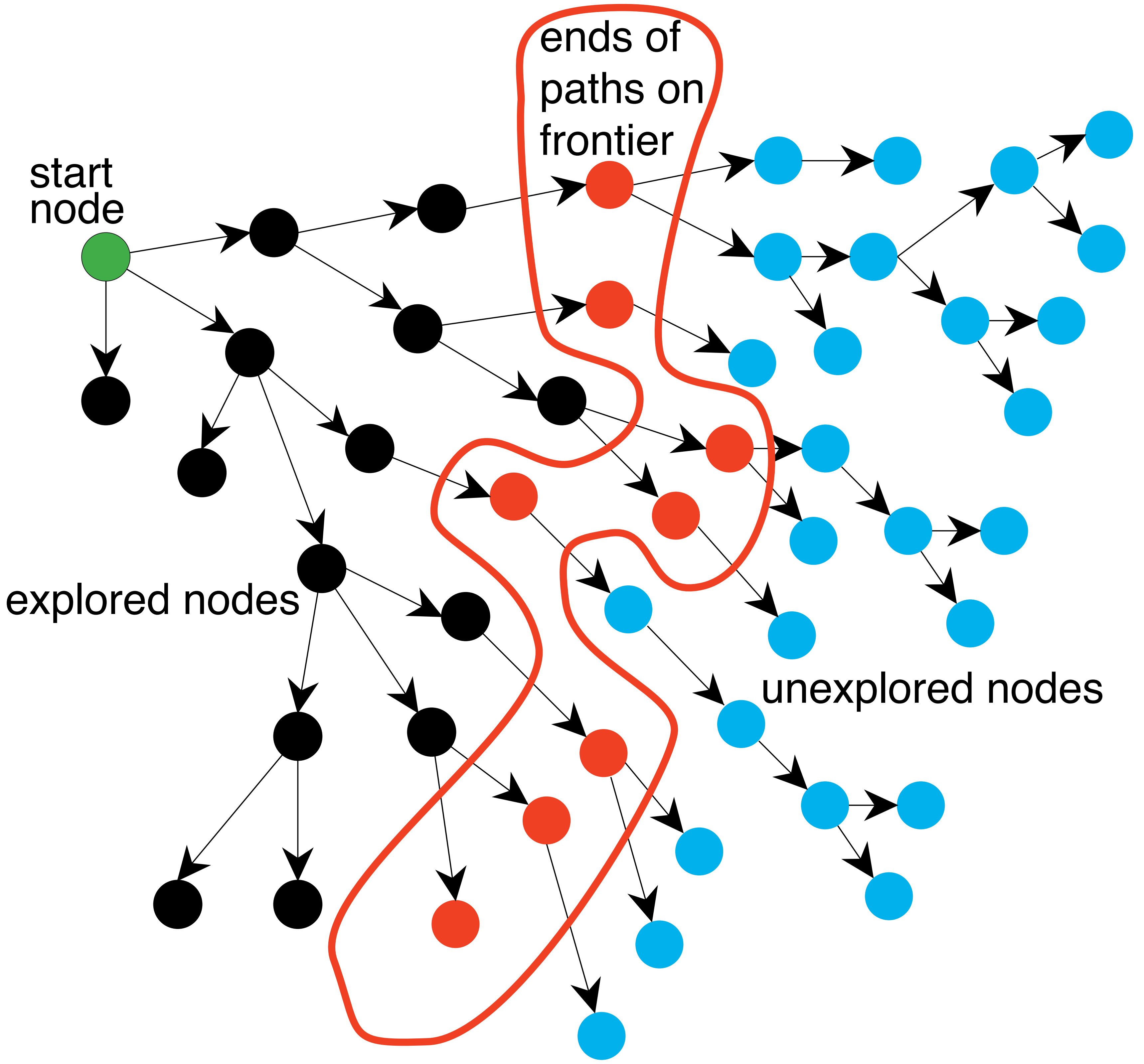
https://artint.info/2e/html/ArtInt2e.Ch3.S2.html



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

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 := { <*s*> | *s* is a start node}
while *frontier* is not empty:
    select and **remove** a path <n_1, n_2, ..., n_k> from *frontier*
    if *goal*(n_k):
        return <n_1, n_2, ..., n_k>
    for each neighbour *n* of n_k:
        add <n_1, n_2, ..., n_k, *n*> to *frontier*
end while

- Can continue the procedure after algorithm returns
- Which value is **selected** from the frontier defines the **search strategy**

Search Problem with Costs

What if solutions have differing qualities?

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

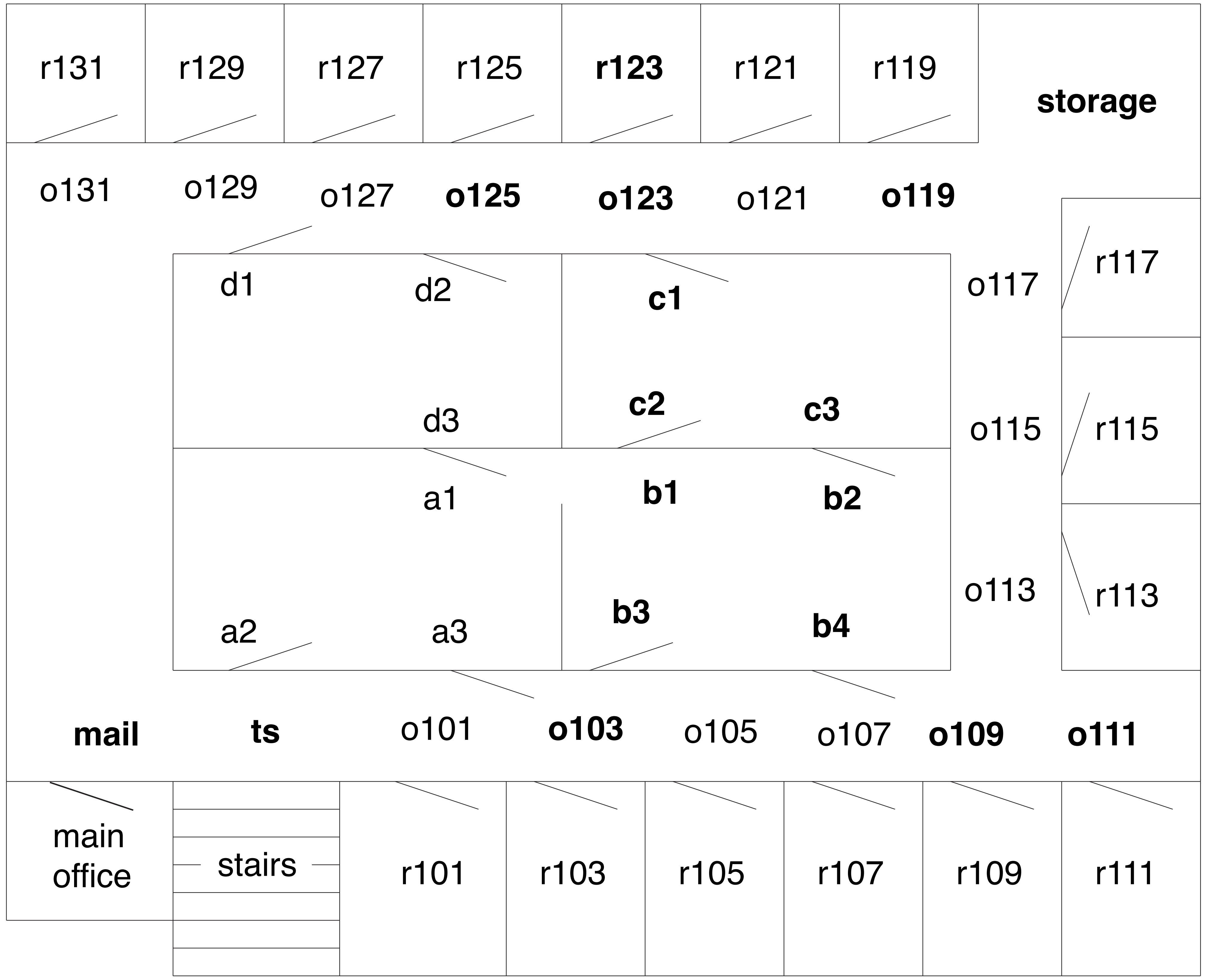
$$\text{cost}\left(\langle n_0, n_1, \ldots, n_k \rangle\right) = \sum_{i=1}^{k} \text{cost}\left(\langle n_{i-1}, n_i \rangle\right)$$

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

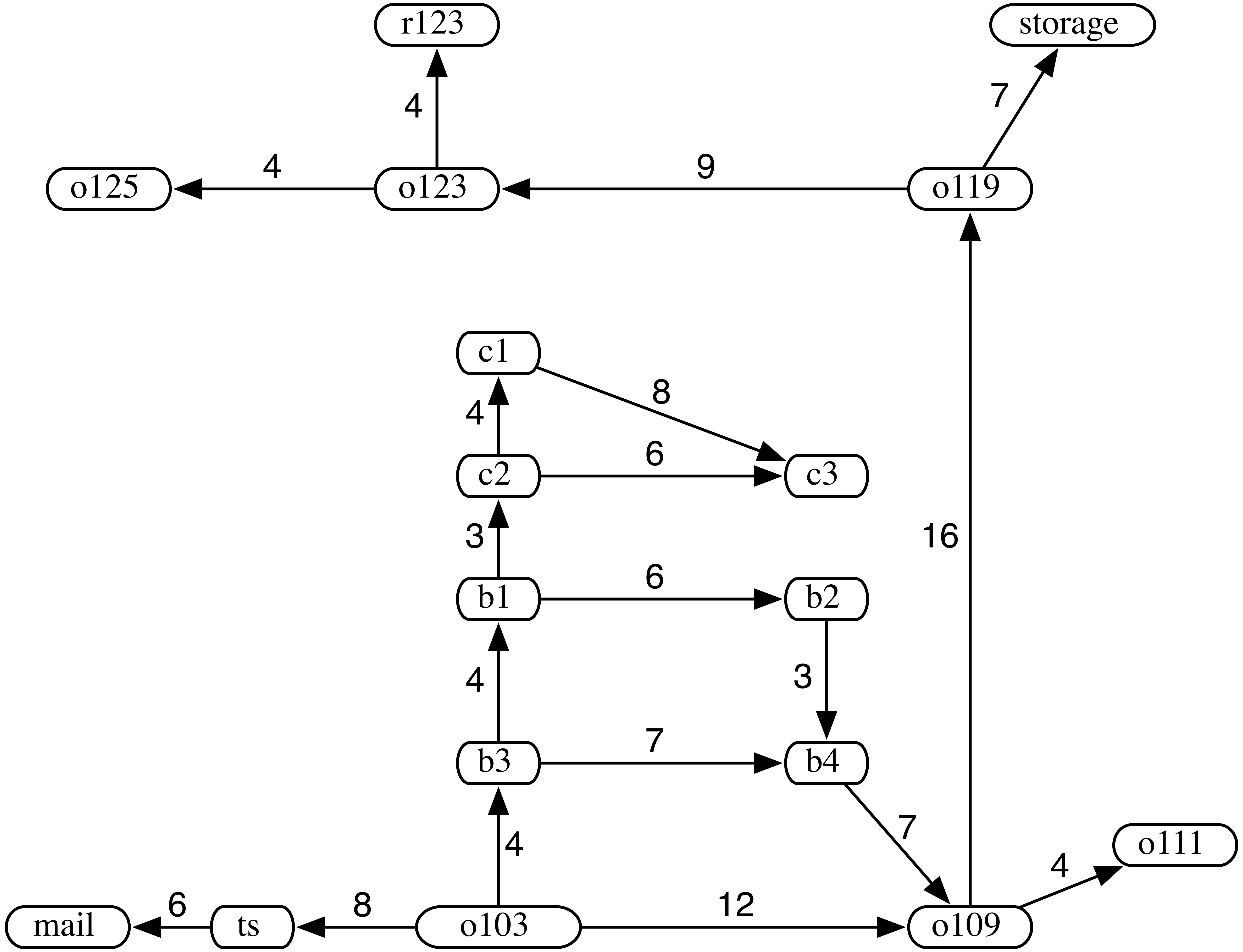
Questions:

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



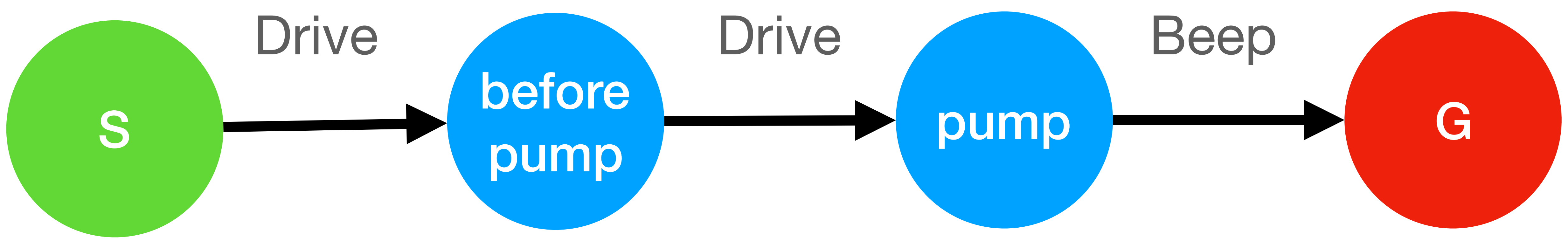
https://artint.info/2e/html/ArtInt2e.Ch3.S2.html



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

Markov Assumption: GasBot

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



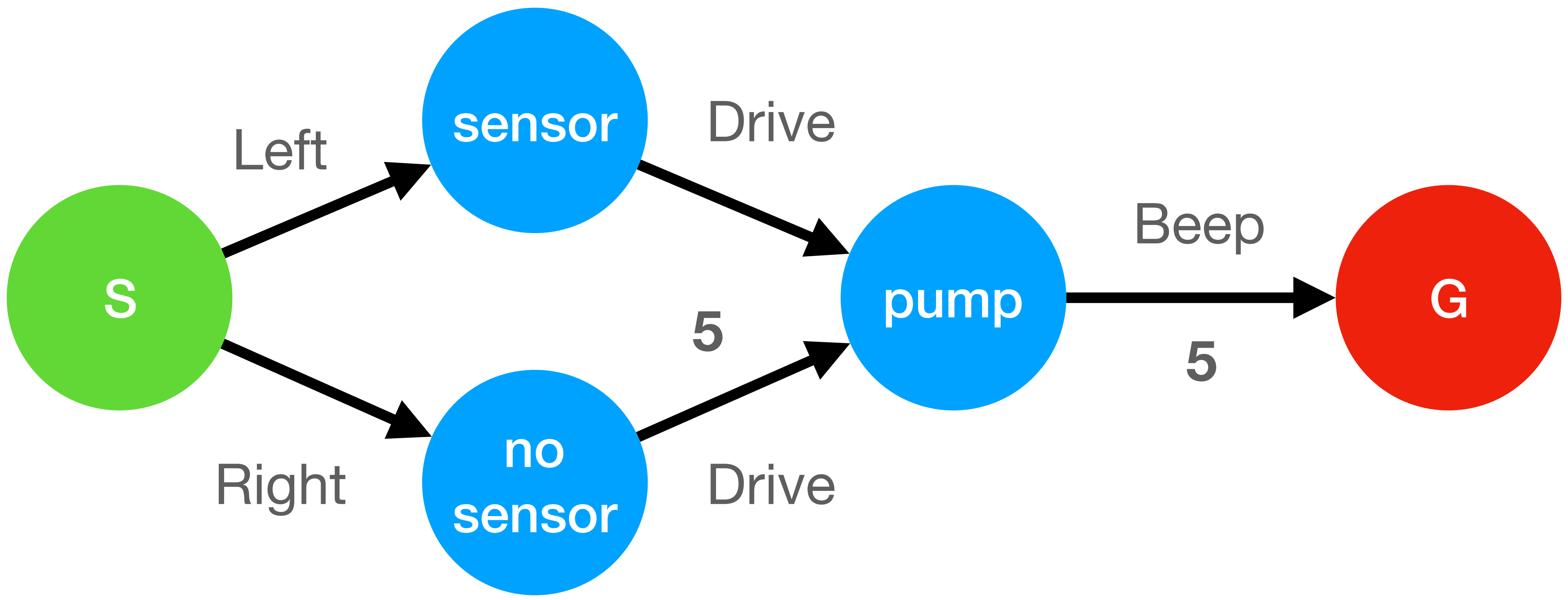
cost(pump, gas):
5 if went through sensor
10 otherwise

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

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

Markov Assumption: GasBot

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



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**