CS143: Artificial Intelligence

Project assignment#1

Posing AI problem solving as search

Drake UNIVERSITY

Agenda

- Project assignment#1 has been released
  - Street-sweeping agent (mapbot) in Python
  - discussing randomly moving vacuum agent
  - discussion on how to make it intelligent

- Posing AI problem solving as search
  - Introduction

Recap: Intelligent Agents

- An **agent** perceives and acts in an environment. It has an architecture and is implemented by a program.

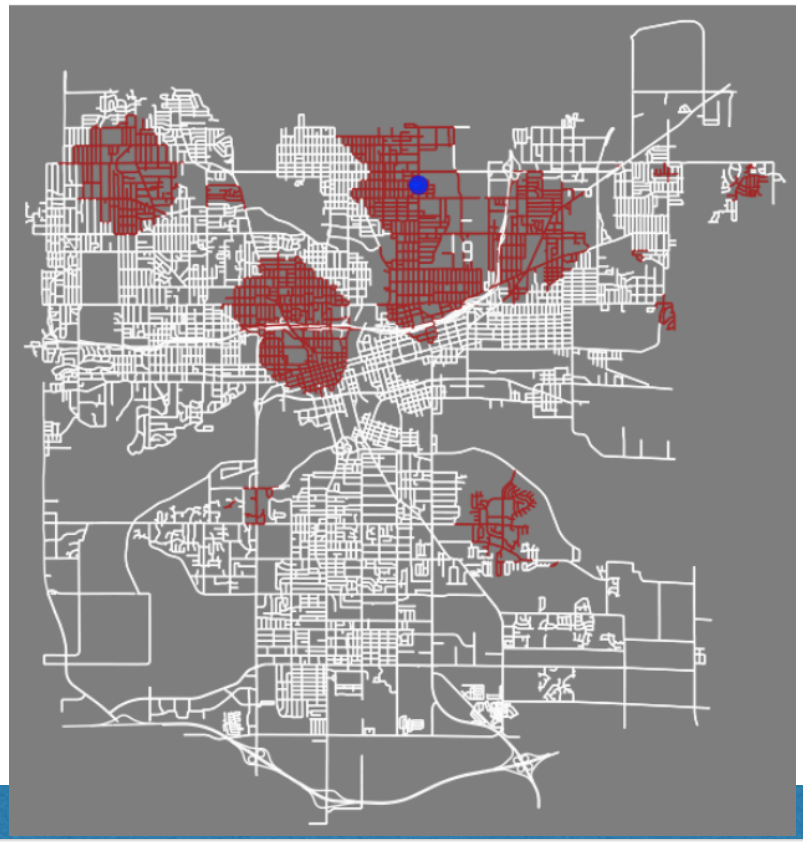
- A **rational agent** always chooses the action which maximizes its expected performance, given the percept sequence received so far and its knowledge about the environment.

- An **agent program** maps from percepts to actions.
  - Reflex agents respond immediately to percepts.
  - Model-based agents use a model of the world to handle partial observability.
  - Goal-based agents act in order to achieve their goal(s).
  - Utility-based agents maximize their own utility function.

Project assignment#1: street sweeping agent (mapbot)

Simple street-sweeping agent

- Create an mapbot in Python that will sweep dirts on the streets of a particular city.
  - We have played with the street_sweeper_demo.ipynb code in our last class



Simple street-sweeping agent

- The agent moves across the 2D map shown above. It starts at a random location on the map (indicated by a blue circle)

- You are provided with code for implementing reflex agents in the Python notebook

Simple street-sweeping agent

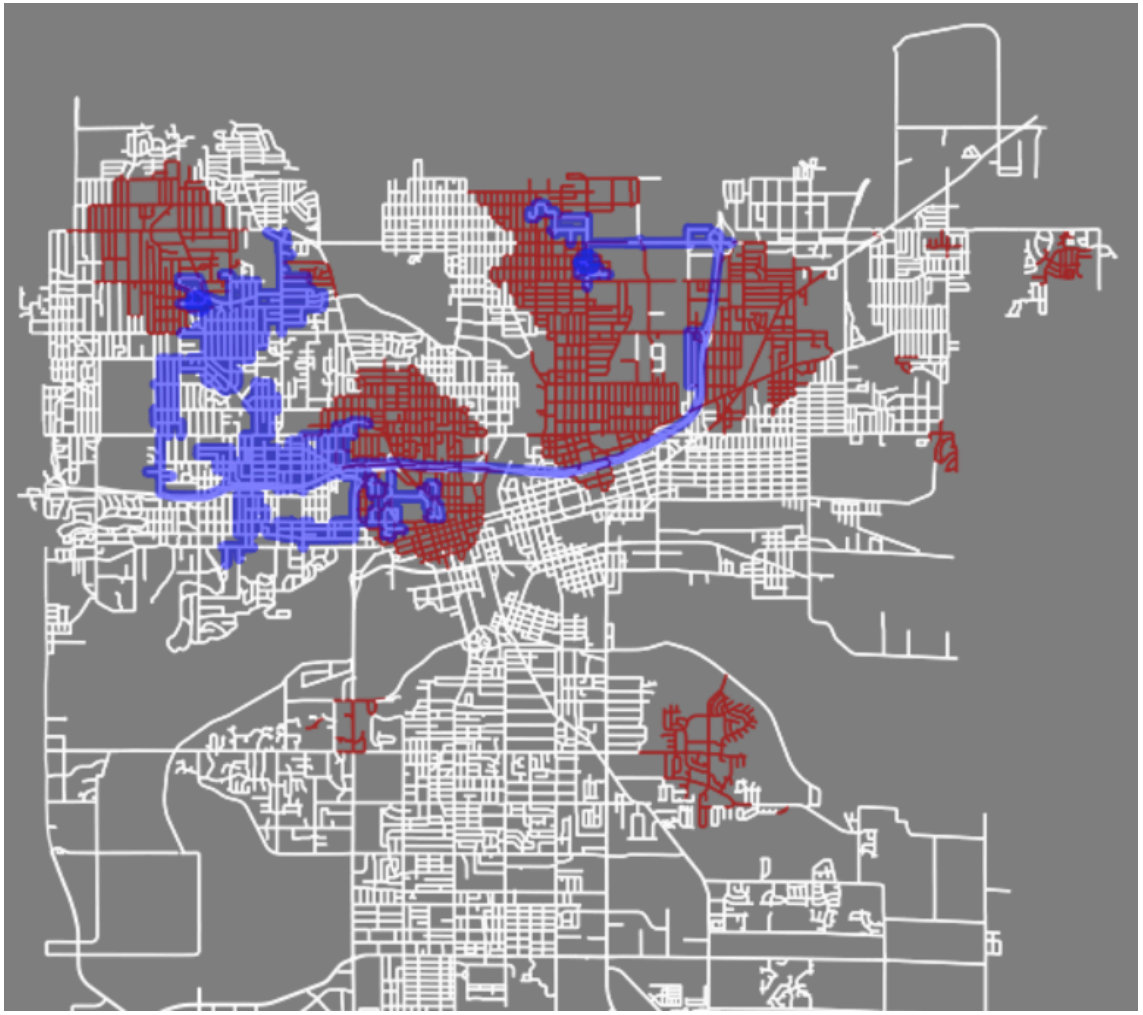
- A simple reflex-agent function which is making a random selection of streets to be cleaned

```
import random
random.seed(10)
while dsm_bot.get_battery_life() > 0:

    move_options = dsm_bot.scan_next_streets()
    if move_options == []:
        dsm_bot.backup(how_many=5)
    else:
        chosen_street = random.choice(move_options)
        dsm_bot.clean_and_move_to(chosen_street["end"]["location_id"])

print("Final location:",dsm_bot.get_current_location())
print("Final battery life:",dsm_bot.get_battery_life())
print("Total distance cleaned (meters):",dsm_bot.get_meters_cleaned())
dsm_bot.display_map()
```

```
Final location: {'y': 41.619102, 'x': -93.674368, 'street_count': 3, 'location_id': 160913126}
Final battery life: -6.526023239631801
Total distance cleaned (meters): 39716.44023858075
```

Simple street-cleaning agent

- As a result the agent cleaned a part of the map until its battery ran out



Simple street-cleaning agent

- The agents used in the demo are **reflex agents**—they make decisions based solely on their current perceptions. The ultimate goal of this assignment is to implement a more intelligent version of the provided reflex-agent function. More specifically, you should come up with a strategy for a model-based agent:

  - this means that you should keep track of something - for example, what locations you have been to or how dirty of an area you seem to be in (e.g., how many dirty streets have you scanned in the last 20 actions?)

  - make rational decisions based on the current percepts and the state that you're keeping track of

Simple street-cleaning agent

- The assignment is worth 8 points and will be graded as follows:
  - You can get up to **1 point** if you turn in the starter notebook and you ran all the code yourself. Create a markdown cell at the top where you explain that's what you did.
  - You can get up to **4 points** if you turn in the starter notebook and you attempted each of the lab exercises.
  - You can get up to **8 points** if you implemented a model-based agent, performed an experiment where you compared it against one of the reflex-only agents, and wrote-up your description and results as directed above. Partial credit (5-7 points) will be given if any of these parts are incomplete.

Posing AI problem solving as searches

An old puzzle

Image source: made with the help of ChatGPT

Puzzle

Puzzles and games have long been considered a challenge for human intelligence:

- Chess in Persia and India 4000 years ago
- Checkers in 3600-year-old Egyptian paintings
- Go in China over 3000 years ago

Old puzzle

Problem Statement:

- There's a man (M), a goat (G), a wolf (W), and a cabbage (C) on one side of a river.
- There's a boat, and it can only hold the man plus another thing.
- Unfortunately, the man can't leave the wolf and goat unattended or else the wolf will eat the goat.
- Similarly he can't leave the cabbage and goat unattended or else the goat will eat the cabbage.
- How can the man get everything (including himself) across the river safely?

Old Puzzle

Let's introduce some notations. We can use the following to represent the entities:

- C stands for the cabbage
- G stands for the goat
- M stands for the man
- W stands for the wolf

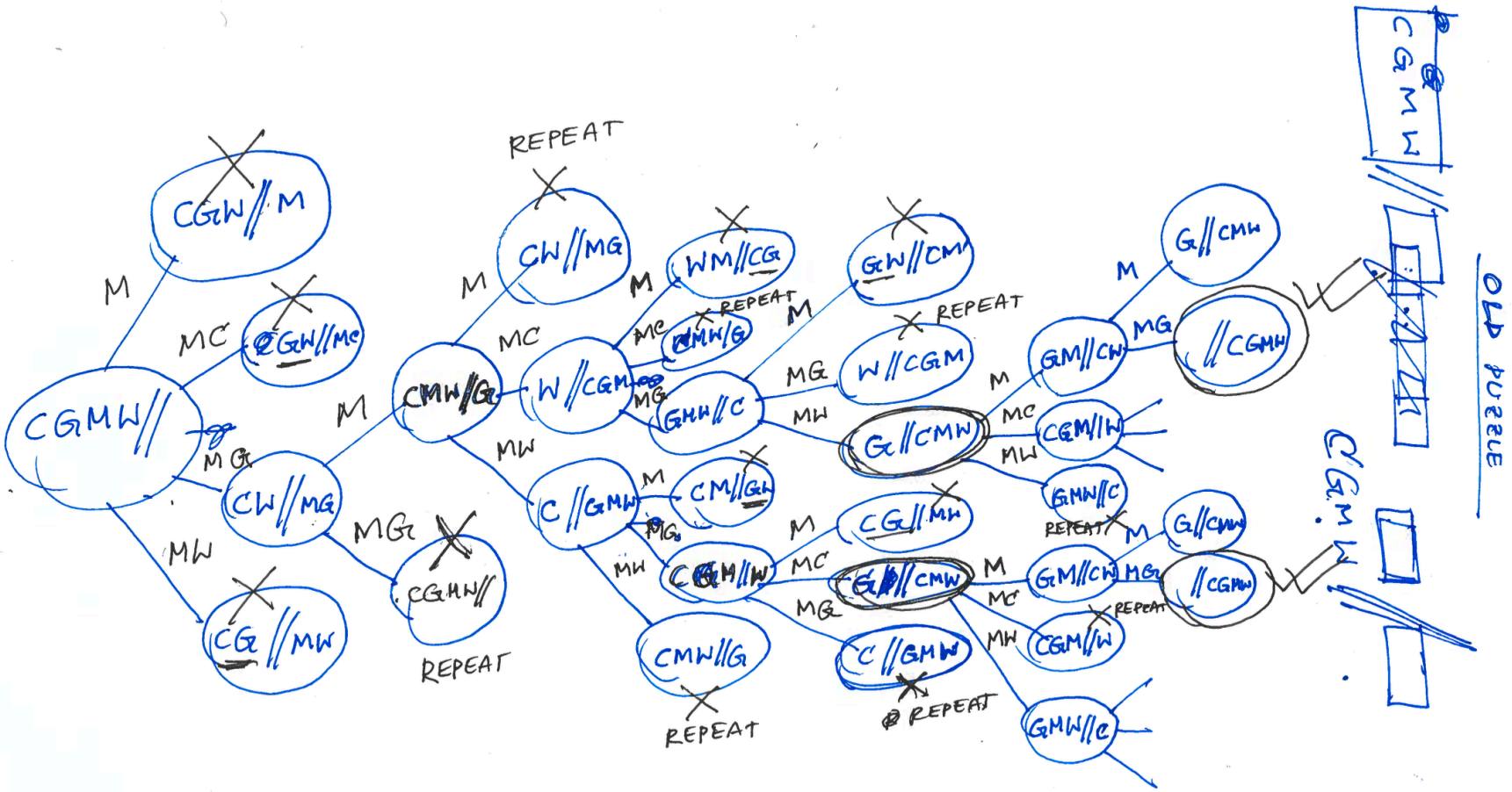
Old Puzzle

- Let's use a slash / to represent the river. Then we can write down symbolic representations of the current state of the whole system.
  - notation like CGMW/ to mean that everything is on the left side of the river
  - notation like /CGMW to mean that everything is on the right side of the river
  - GW/MC means the wolf and goat are on the left side of the river and the man and cabbage are on the other side.

How can you solve this old puzzle?

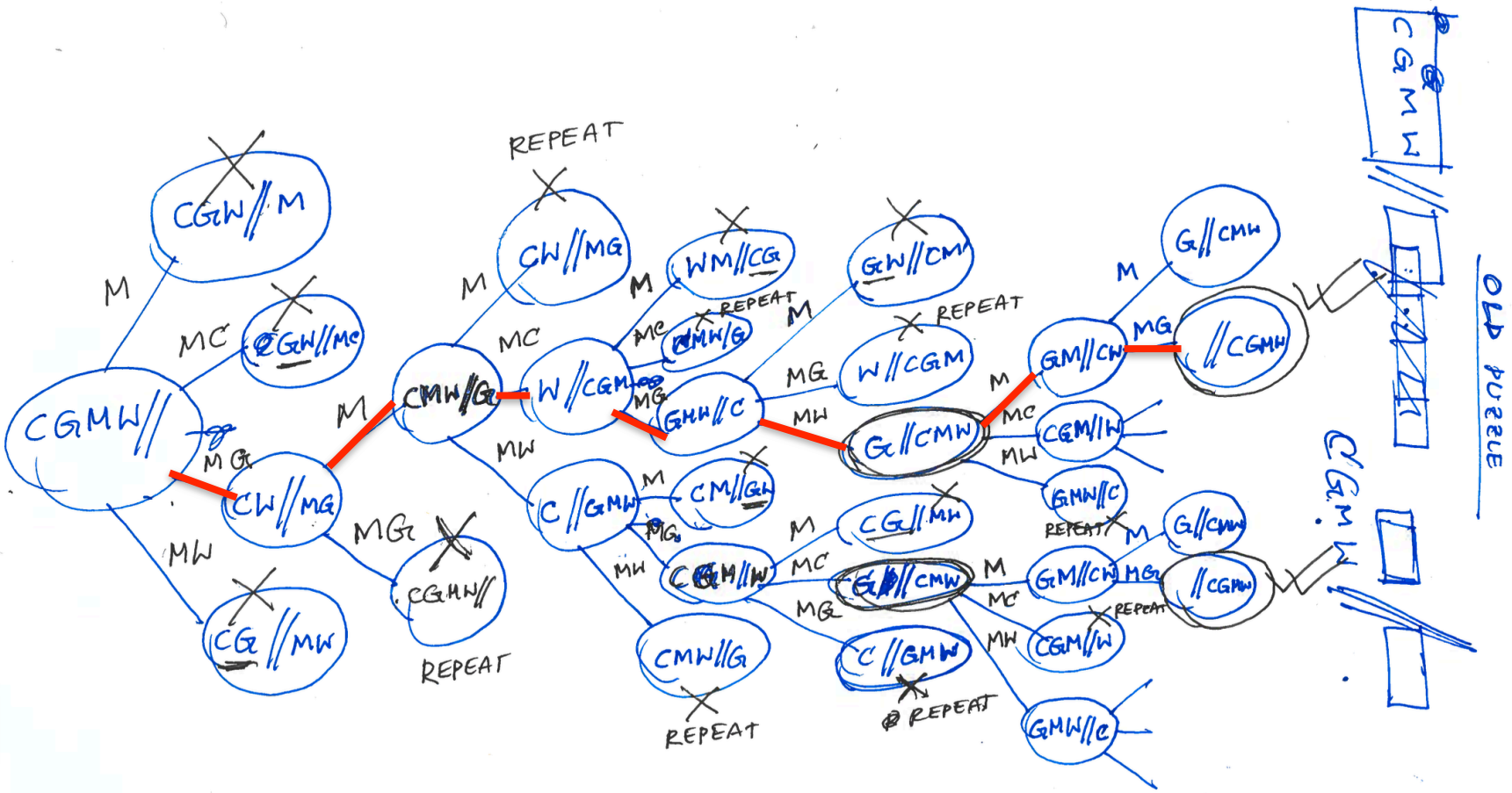
Solution: Old Puzzle

- How many possible states of the system are there?



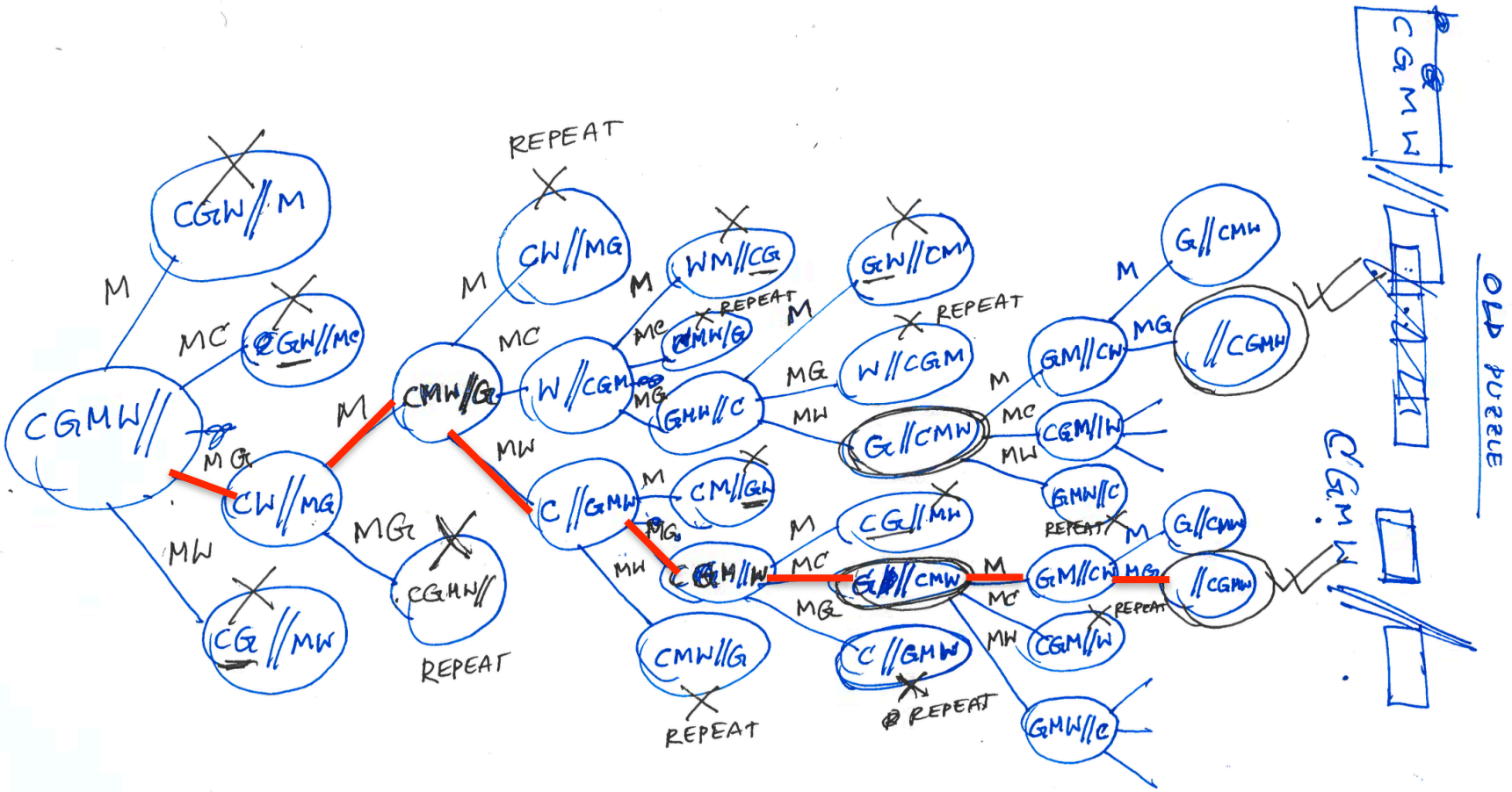
Solution: Old Puzzle

- One path to the solution (from the starting state to goal state)



Solution: Old Puzzle

- Another path to the solution (from the starting state to goal state)



Exploring Choices

- Many AI problems that seem to require intelligence usually require exploring multiple choices.

- Search: a systematic way of exploring choices.

Exploring Choices

- This also means that if we think about a new AI problem properly, we can use existing search algorithms to solve it without much additional effort!

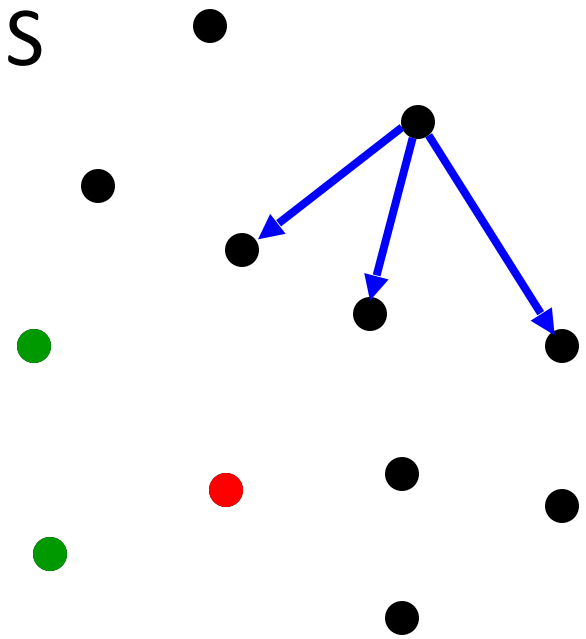
- In other words, if we approach a new problem correctly, we don't need a specific new algorithm to solve:
  - Eg, the problem of the cabbage, man, goat, and wolf can be solved with an existing search algorithm

- We just need to think about the "cabbage, man, goat, and wolf" problem in the right way, and then we can use existing algorithms to solve it automatically.

These abstractions have 5 parts

1. Set of states S
2. Initial state s_0
3. A function SUCC: S $\rightarrow 2^S$ that encodes possible transitions of the system
4. Set of goal states
5. A cost function that calculates how "expensive" a given set of moves is

Defining a search problem



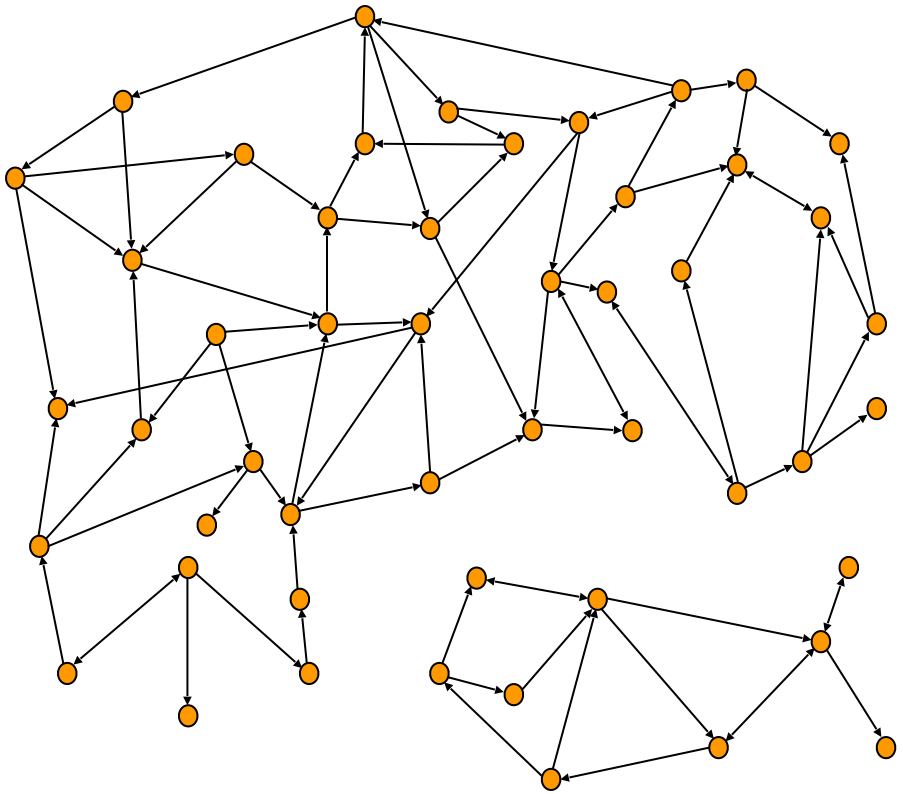
- State space S
- Successor function:

  $x \in S \rightarrow \text{succ}(x) \in 2^S$
- Initial state s_0
- Goal test:

  $x \in S \rightarrow \text{GOAL?}(x) = T \text{ or } F$
- Cost

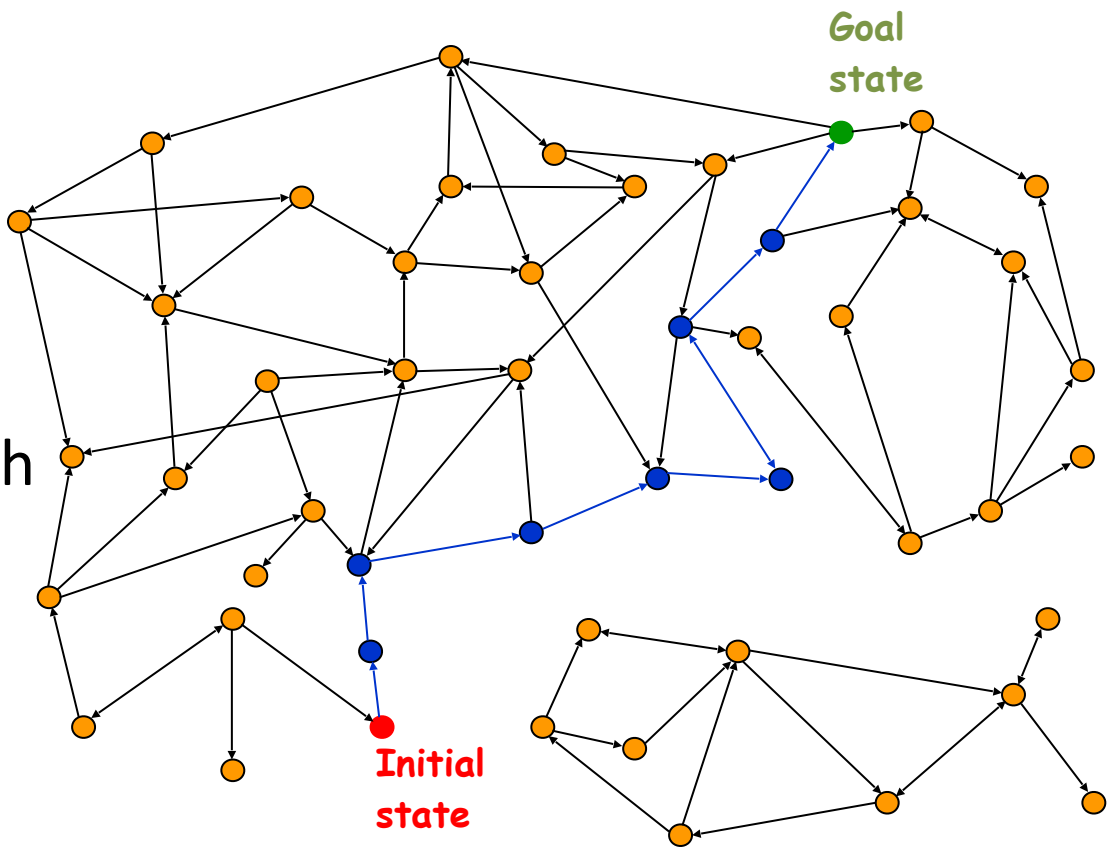
State Graph

- Each state is represented by a distinct node
- An edge connects a node s to a node s' if $s' \in \text{SUCC}(s)$
- The state graph may contain more than one connected component



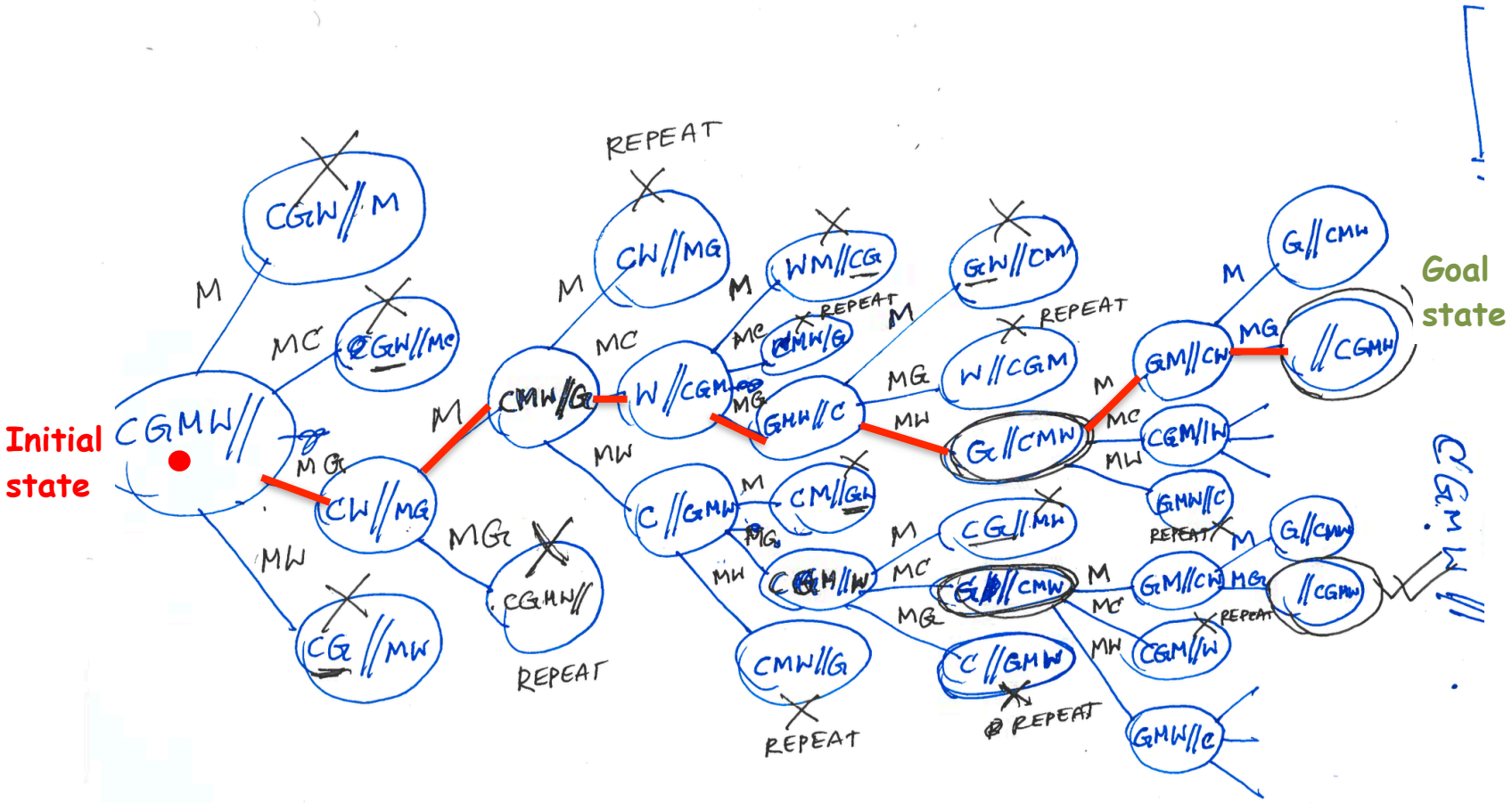
Solution to the Search Problem

- A solution is a path from the initial to any goal node
- Path cost is sum of the edge costs along the path
- An optimal solution has minimum cost
  - There might be no solution!
  - There might be multiple solutions

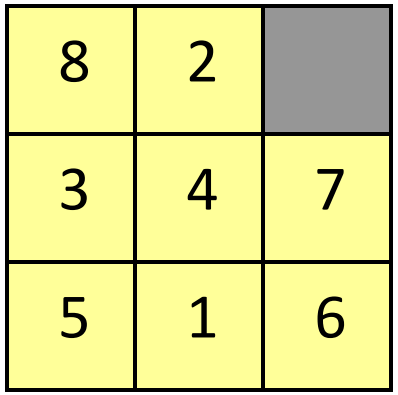


Example: solution to old puzzle

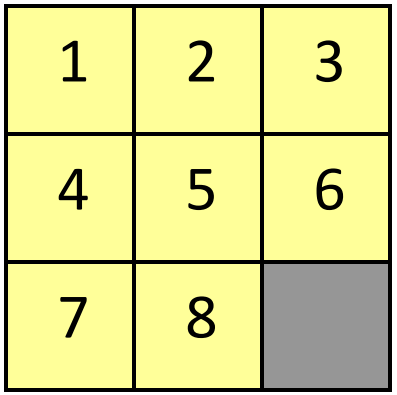
- One path to the solution (from the starting state to goal state)



Example: 8-Puzzle

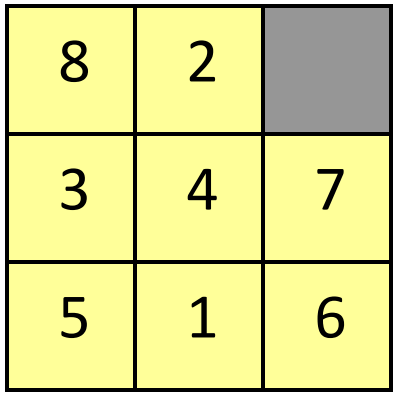


Initial state

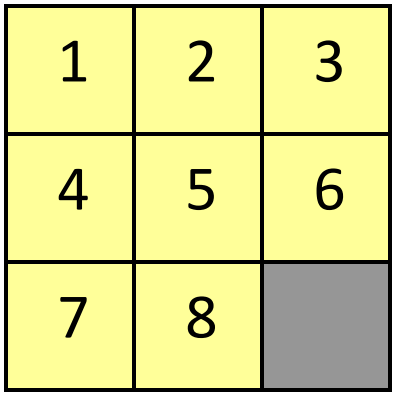


Goal state

Example: 8-Puzzle



Initial state



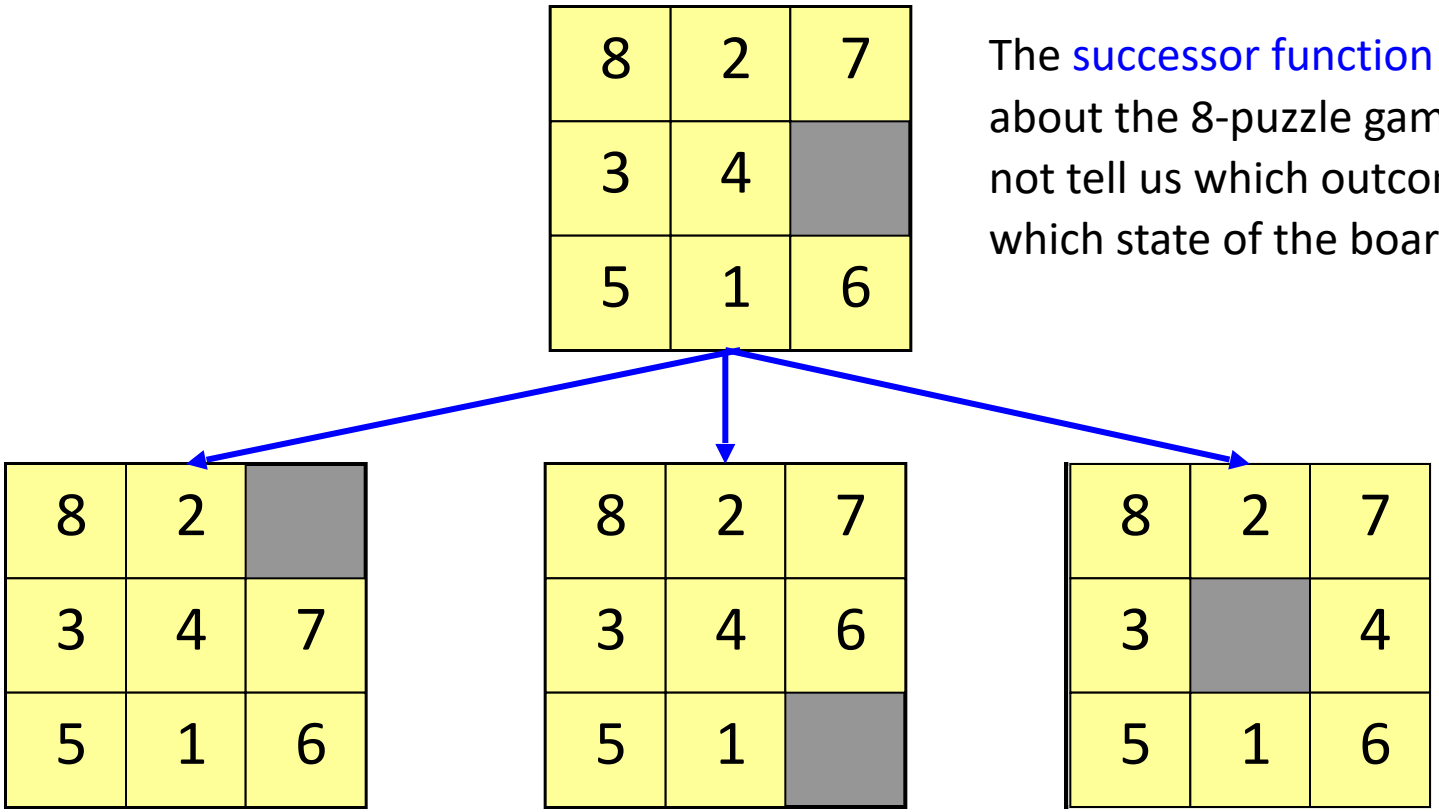
Goal state

State: Any arrangement of 8 numbered tiles and an empty tile on a 3x3 board
Successor function: given by available actions (sliding tiles) L, R, U, D.
Cost: How many moves were performed

Successor function: 8-Puzzle

SUCC(state) → subset of states

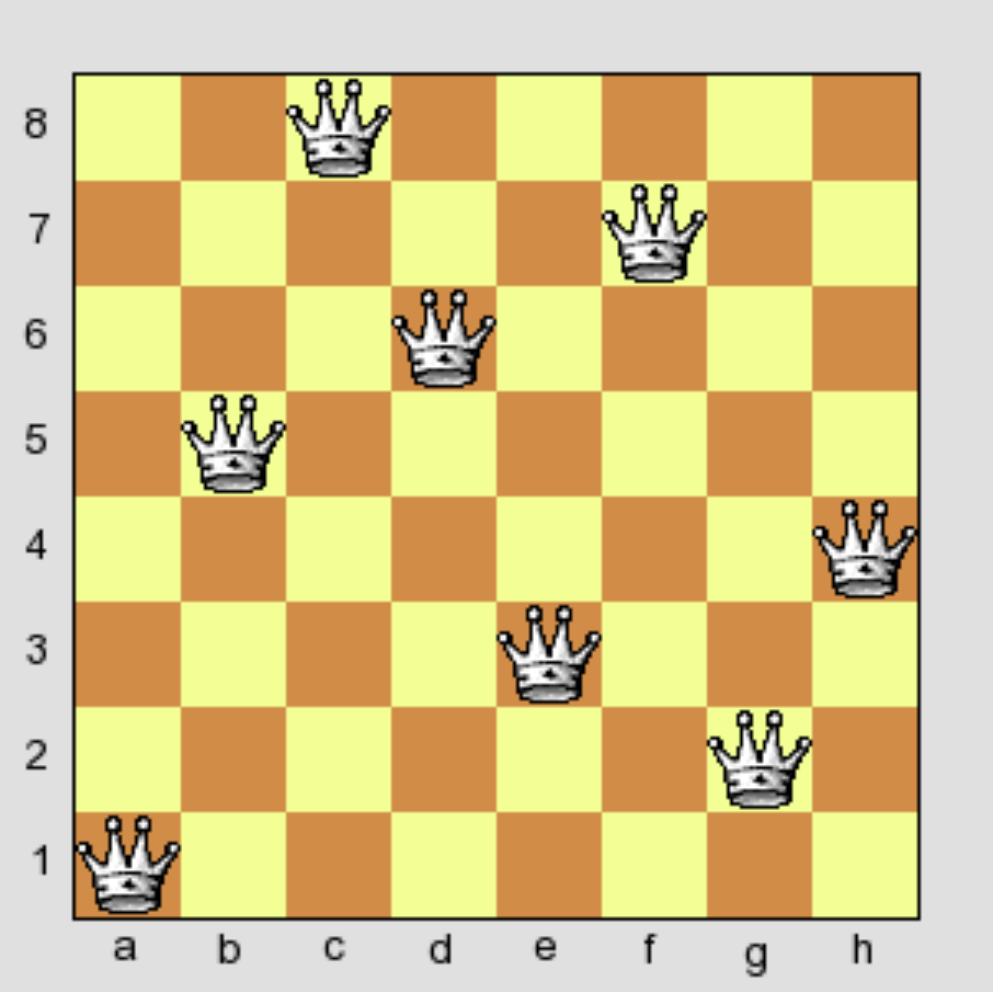
The successor function is knowledge about the 8-puzzle game, but it does not tell us which outcome to use, nor to which state of the board to apply it.



Example: 8-Queens Problem

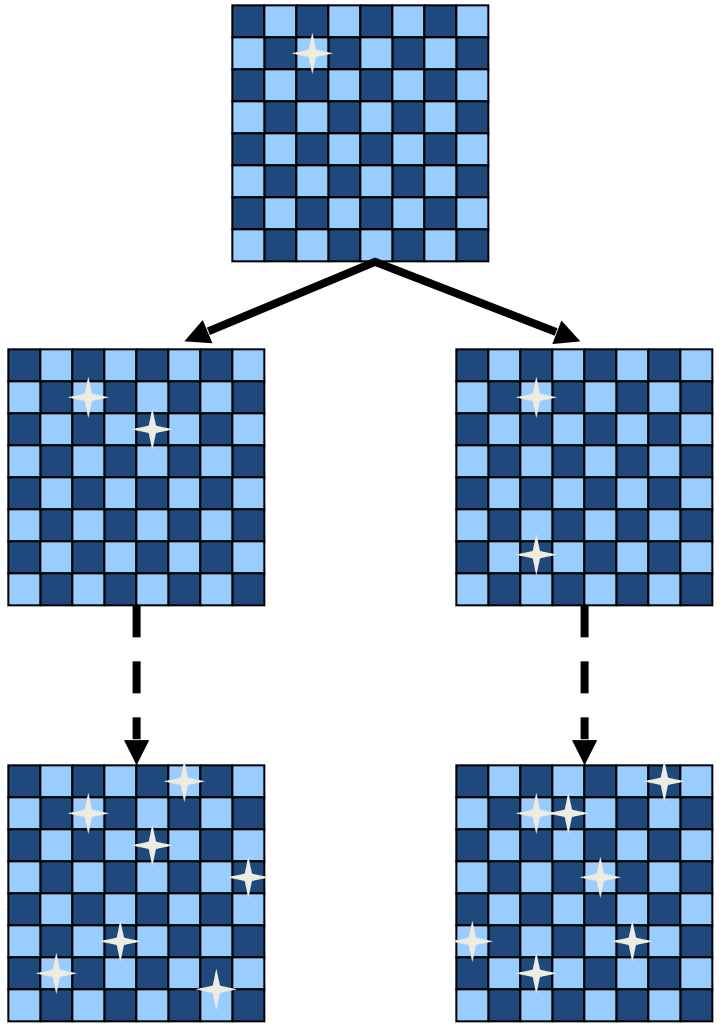
Place 8 queens in a chessboard so that no two queens are in the:

- same row or
- same column or
- same diagonal



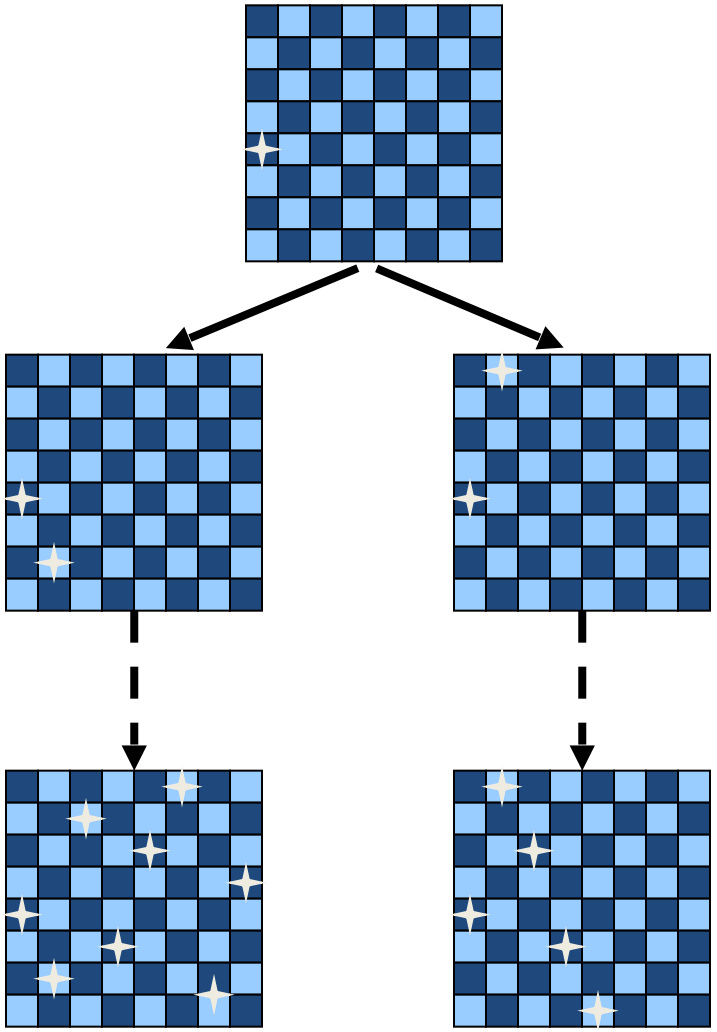
Formulation-1: 8-Queens Problem

- **State:** configuration of 0-8 queens
- **Initial state:** 0 queens
- **Successor function:** add queen to an empty square
- **Goal states:** non-conflicting placement of 8 queens
- **Cost:** irrelevant



Formulation-2: 8-Queens Problem

- **State:** configuration of 0-8 non-conflicting queens in columns starting from left
- **Initial state:** 0 queens
- **Successor function:** place queen in leftmost empty column so that there is no conflict
- **Goal :** State with 8 queens
- **Cost:** irrelevant
- Reduced number of states from the previous formulation



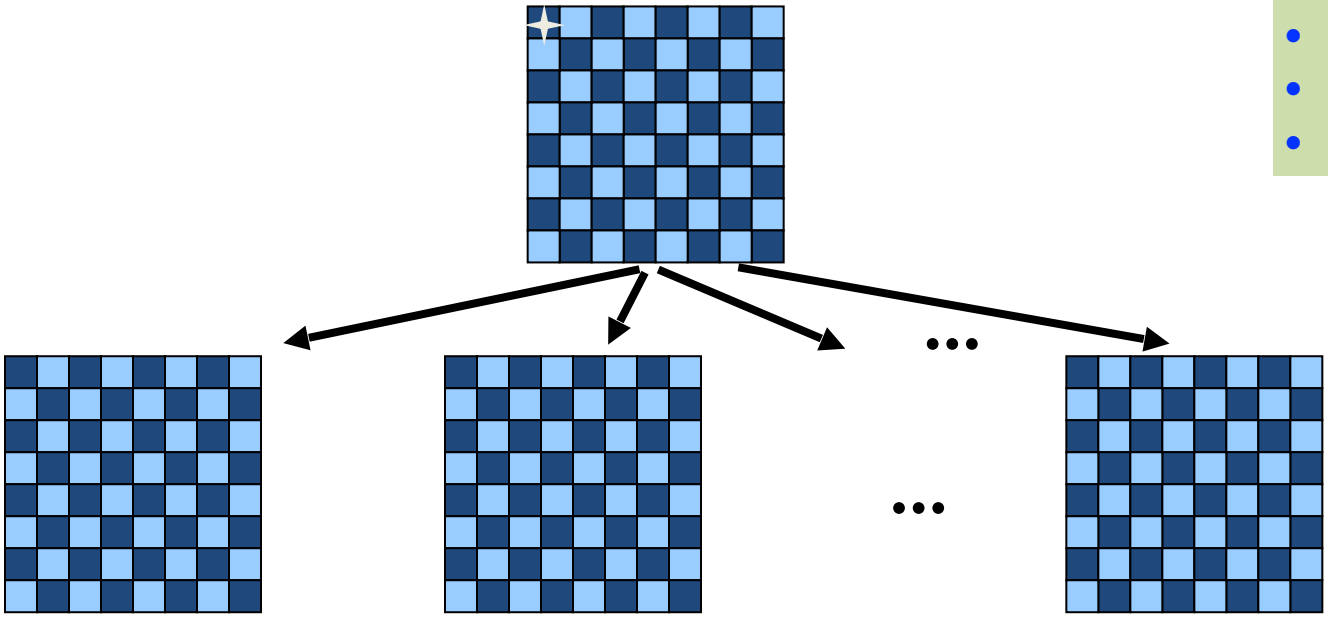
Exercise: Generate the states for 8-Queens problem

Exercise: Generate the states for the following formulation of 8-Queens problem

- **State:** configuration of 0-8 non-conflicting queens in columns starting from left
- **Initial state:** 0 queens
- **Successor function:** place queen in leftmost empty column so that there is no conflict
- **Goal :** State with 8 queens

Recall that conflicts happen in a chessboard when two queens are in the:

- same row or
- same column or
- same diagonal

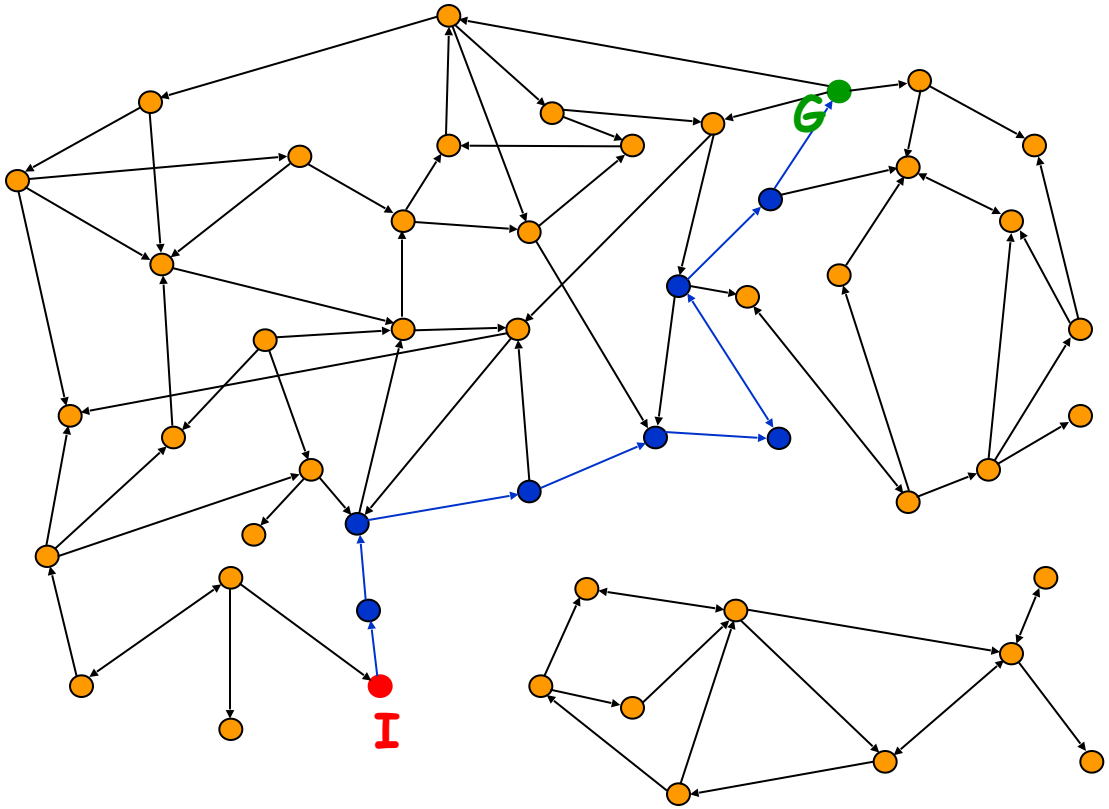


What is a state?

- **A state does:**
  - Represent all information meaningful to the problem at a given "instant in time" – past, present, or future
  - Exist in an *abstract*, *mathematical* sense
- **A state DOES NOT:**
  - Necessarily exist in the computer's memory
  - Tell the computer how it arrived at the state
  - Tell the computer how to choose the next state
  - Need to be a unique representation

Pathless Problems

- Sometimes the path doesn't matter
- A solution is any goal node
- Edges represent potential state transformations
  - E.g. 8-queens, Map coloring



Discussion

- What information would your Street Sweeper agent need to know to be able to do search?