Artificial Intelligence (CS6380)

State Space Search

State space search: notation

Abstract the problem as a state space search problem

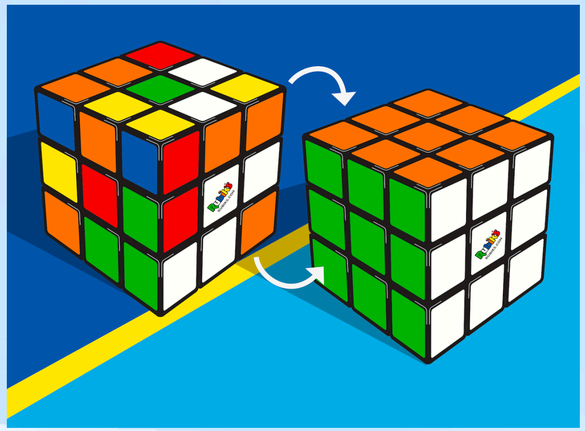
Level of abstraction: too detailed vs too coarse

Path finding : CSE dept to main gate. How to model?

- Set of states (state space)
- Initial state
- Goal states / goalTest / is goal function
- Actions
- Transition function
- Action cost function

Problem 1

- Do you **know** how to solve the cube?
- Can you try to solve the Rubik's cube?
- How good is the solve?



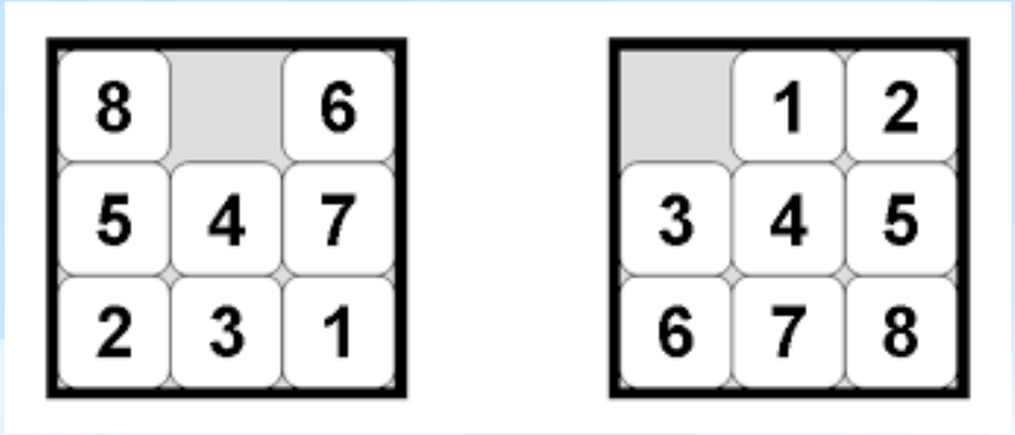
Rubiks cube

- **Set of states (state space):** all configurations of the cube. Depending on the representation. ~10^19 states.
- **Initial state:** given configuration
- **Goal states / goalTest / is goal function:** solved cube
- **Actions:** face = T / Bt / L / R / F / Bk; rotate = 90 / 180 / 270
- **Transition function:** config-1 T-180 config-2, …
- **Action cost function:** unit cost

Trivia

- Designed by sculptor and architect Rubik (1974)
- Initially called as Magic cube

Problem 2



Eight puzzle

- Set of states (state space)
- Initial state
- Goal states / goalTest / is goal function
- Actions
- Transition function
- Action cost function

Problem 3

- Set of states (state space)
- Initial state
- Goal states / goalTest / is goal function
- Actions
- Transition function
- Action cost function

- Man, goat, wolf, cabbage puzzle.
- Boat can carry at most 2 things and there are constraints on who can stay together safely.
- How does the man take everyone across the river?

Problem 4

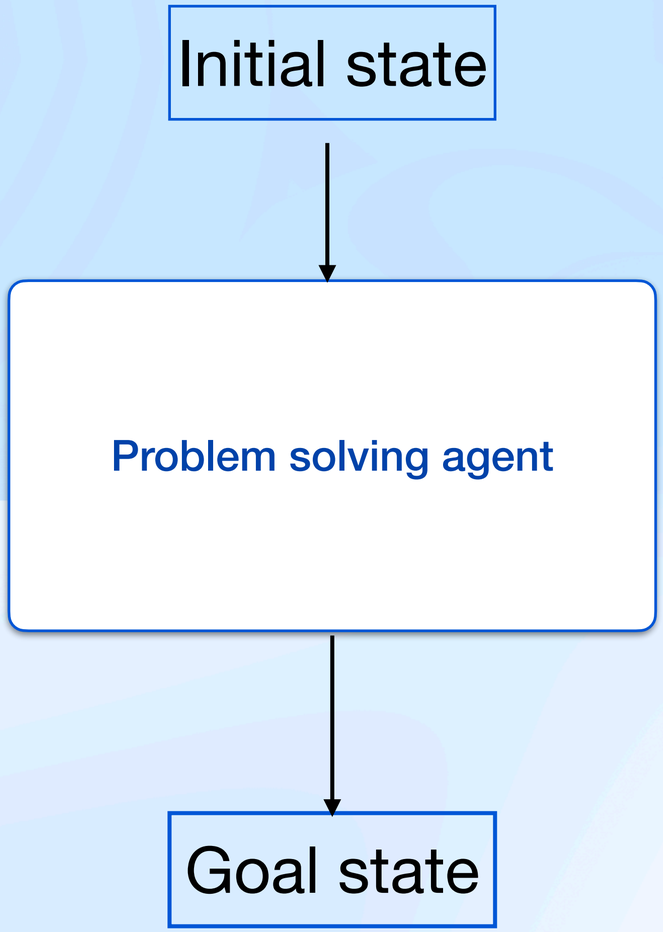
Knuth's conjecture

- Start with integer 4.
- Apply sq. root, floor, factorial.
- Goal : to obtain the desired integer.

- Set of states (state space)
- Initial state
- Goal states / goalTest / is goal function
- Actions
- Transition function
- Action cost function

https://oisinmoran.com/projects/root_floor_fact_four

Problem solving agent



Single agent solves/ attempts to solve all problems

- Rubik's cube
- 8 puzzle
- Man, cabbage, goat, fox
- 4 → 5 (Knuths conjecture)

State space is a graph: nodes as states and arcs representing transitions.
State space graph is not given to us. It is present implicitly

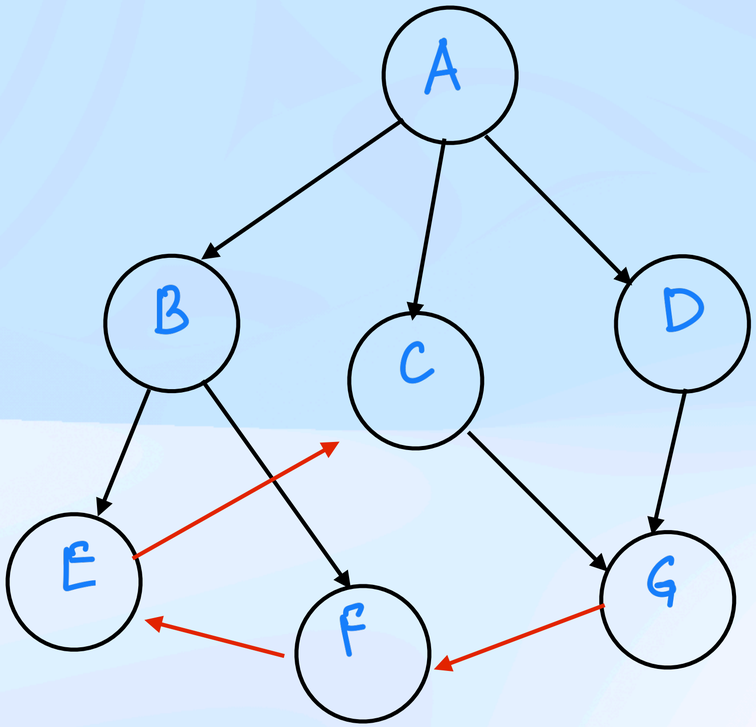
Search algorithm : evaluation

How do we evaluate different search algorithms?

- **Completeness:** does the algorithm find a solution when there is one and report failure when a solution does not exist?
- **Cost optimality:** does it find the lowest cost path amongst all solutions?
- **Time complexity:** how long does it take to find a solution?
  - Graph is not given explicitly. Cannot use usual parameters like |V|, |E|
- **Space complexity:** how much memory is required to perform the search?

Uninformed search

How to search?



For a particular node expand its neighbourhood / moveGen function (filter invalid states if generated)

- Generates child nodes.
- Pick one of the child nodes for further expansion
  - Which one to select?
  - Need some way to order child nodes. Use a **function f**
- Maintain set of nodes to expand. Call it **frontier / open list**
- What is missing in the state space graph beside?
  - **Cycles / loops**.

Generic search algorithm

Create an empty **frontier** and insert start-node to **frontier**
Create an empty **reachedList** and insert (start-node, initial-cost) to **reachedList**
While frontier not empty do

- Extract **best** node from frontier. // **best** is decided via **function f**
- If node is goal node return node
- Expand node to get child nodes
- For each child node do
  - newCost = node.path-cost + transition-cost
  - If (child not in **reachedList**)
    - Add child to **frontier**
    - child.parent = node
    - Add (child, new cost) to **reachedList**
  - Else if newCost < child.path-cost // child is in **reachedList**
    - Add child to **frontier**
    - child.parent = node
    - Replace occurrence of child in **reachedList** by (child, newCost)

Return failure

Generic search algorithm

Create an empty **frontier** and insert start-node to **frontier**
Create an empty **reachedList** and insert (start-node, initial-cost) to **reachedList**
While frontier not empty do

- Extract **best** node from frontier. // **best** is decided via **function f**
- If node is goal node return node
- Expand node to get child nodes
- For each child node do
  - newCost = node.path-cost + transition-cost
  - If (child not in **reachedList**)
    - Add child to **frontier**
    - child.parent = node
    - Add (child, new cost) to **reachedList**
  - Else if newCost < child.path-cost // child is in **reachedList**
    - Add child to **frontier**
    - child.parent = node
    - Replace occurrence of child in **reachedList** by (child, newCost)

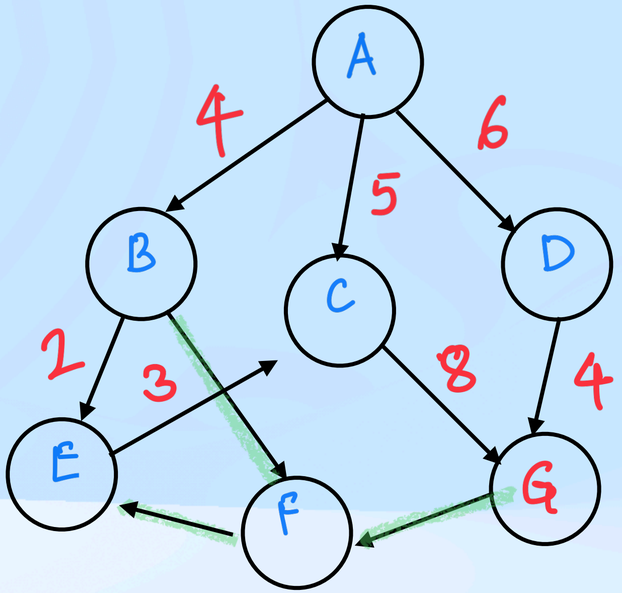
Return failure

problem specific

late goal test.

How can we use **function f** to get familiar algorithms?

Example continued

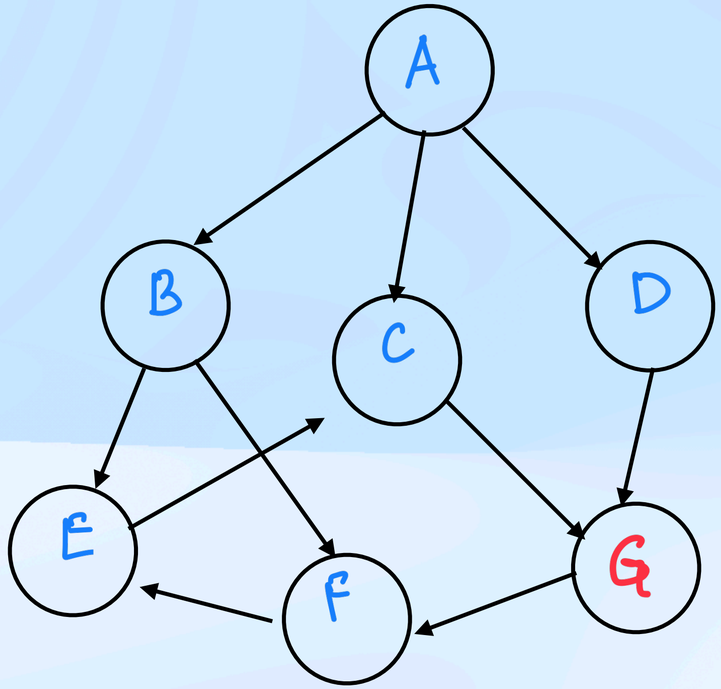


Paths from A ⇝ G

1. A → B → E → C → G
2. A → B → F → E → C → G
3. A → C → G
4. A → D → G

- **b** : branching factor or maximum number of successors of a node
- **d** : number of actions in an optimal solution
- **m** : maximum number of actions in any path
- **C*** : cost of optimal solution
- $\epsilon > 0$: cost of the cheapest action

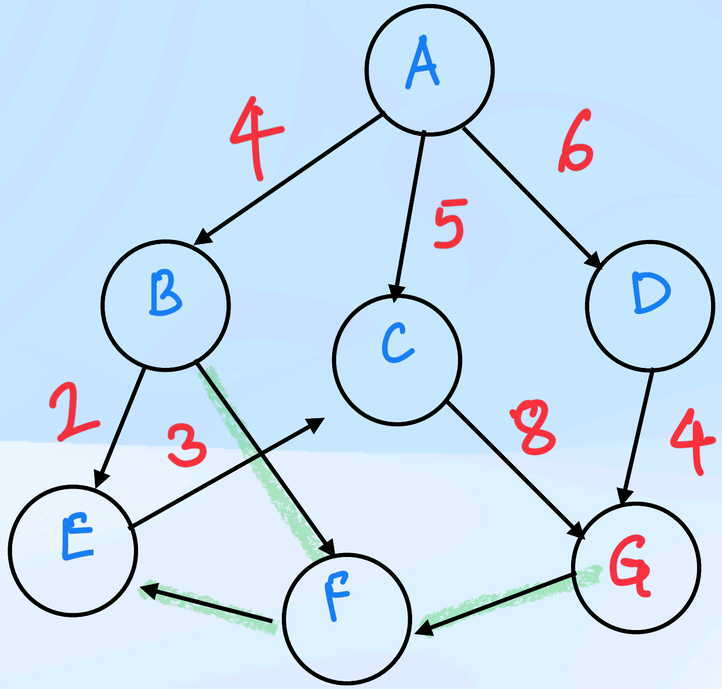
Breadth First search (BFS)



- **Completeness:** yes if the space is finite or the goal is reachable from the start state.
- **Cost optimality:** yes for uniform cost actions. No when the costs are not uniform.
- **Total number of nodes generated:** **1+ b + b^2 + b^3 + … + b^d**
- **Time complexity:** for finite state spaces : O (|V| + |E|). For inifinite spaces when goal is reachable: proportional to the total number of nodes generated.

function f : depth of node.

Uniform cost search (Dijkstra)



Paths from A ⇝ G

① A → B → E → C → G

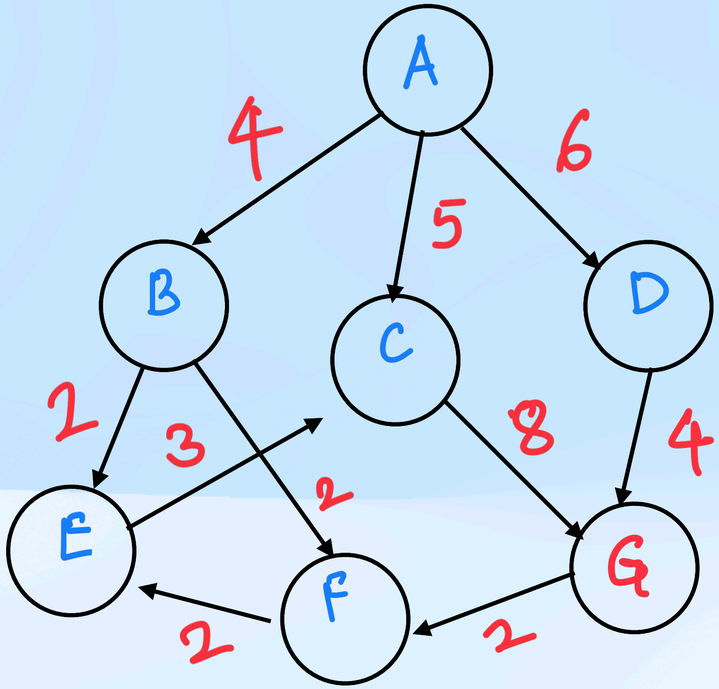
② A → B → F → E → C → G

③ A → C → G

④ A → D → G

function f : path cost of node.

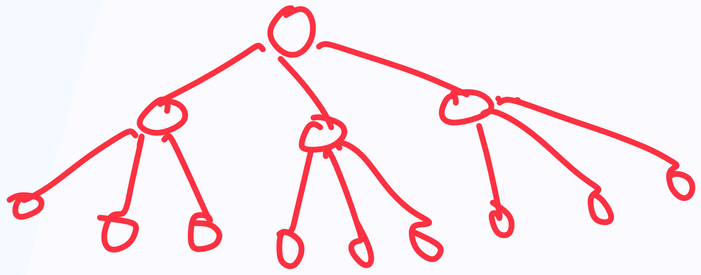
Uniform cost search (Dijkstra)

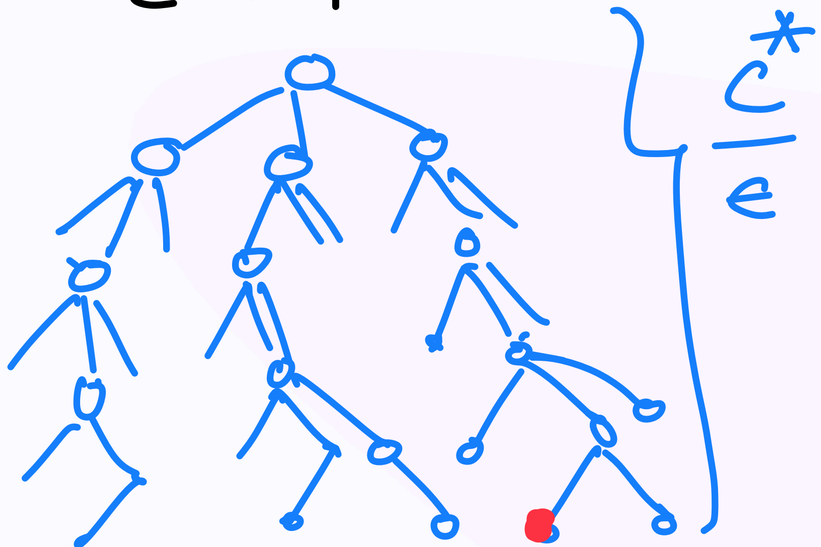


- **Completeness:** yes if the space is finite or the goal is reachable from the start state.
- **Cost optimality:** yes for uniform cost actions. No when the costs are not uniform.
- **Time and space :** for finite state spaces : O ((|V| + |E|) log(|V|)), O (|V| + |E|). For infinite spaces when goal is reachable: proportional to the total number of nodes generated.

function f : path cost of node.

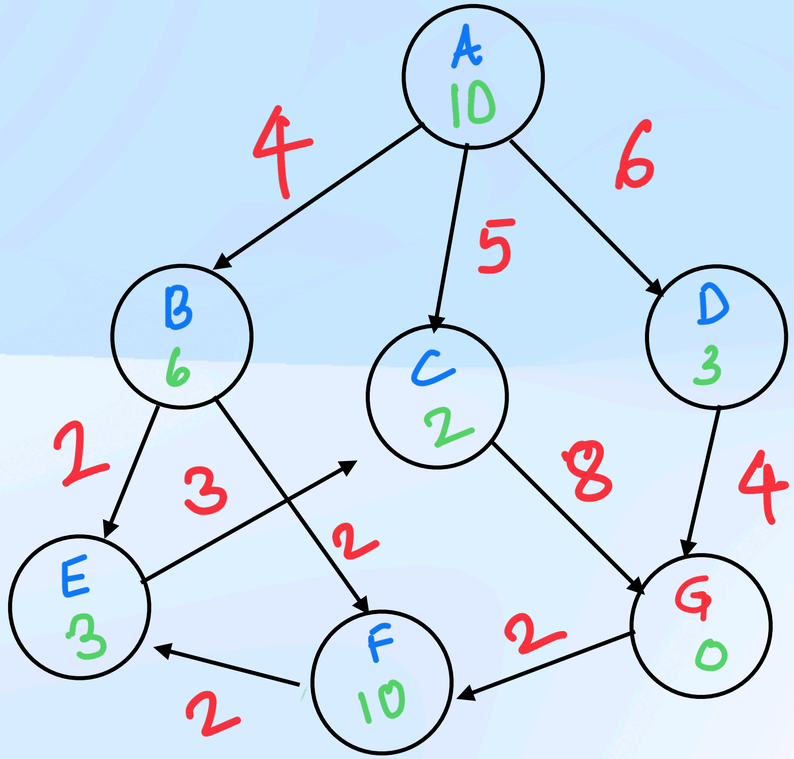
$b = 3$, $d = 2$, $C^* = 10$ $\epsilon = 2$





Informed search

Greedy best-first search



Number inside the node : estimate of cheapest cost from node to goal. Call this h(n).

Recall is path-cost. Call it g(n).

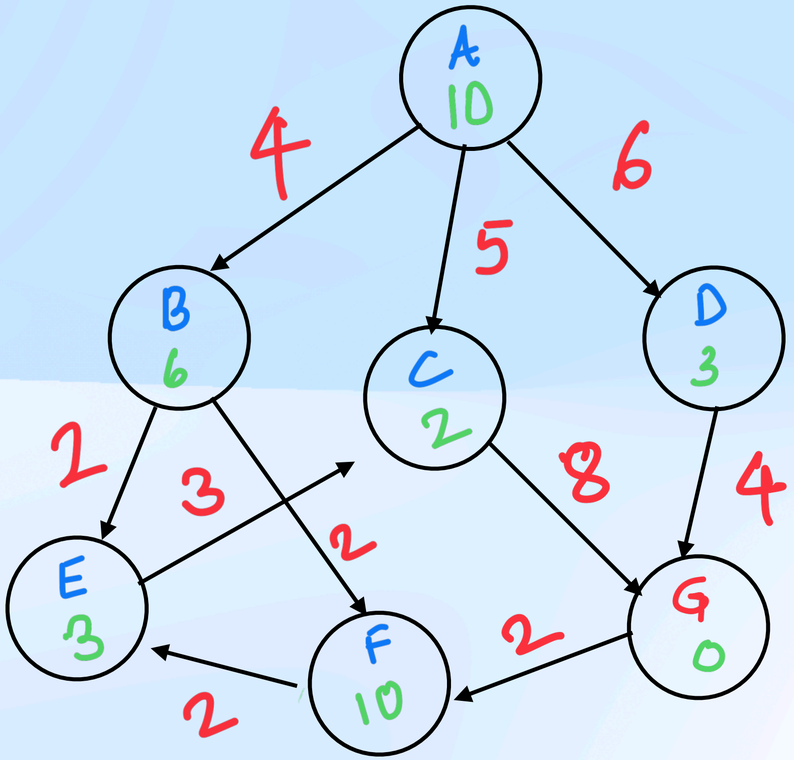
Run the generic search algorithm with f(n) = h(n)

Does it return optimal cost path?

No! It gives A → C → G.

How many states were explored?

Combining h(n) and g(n)



Number inside the node : estimate of cheapest cost from node to goal. Call this h(n).

Recall is path-cost. Call it g(n).

Run the generic search algorithm with f(n) = h(n) + g(n)

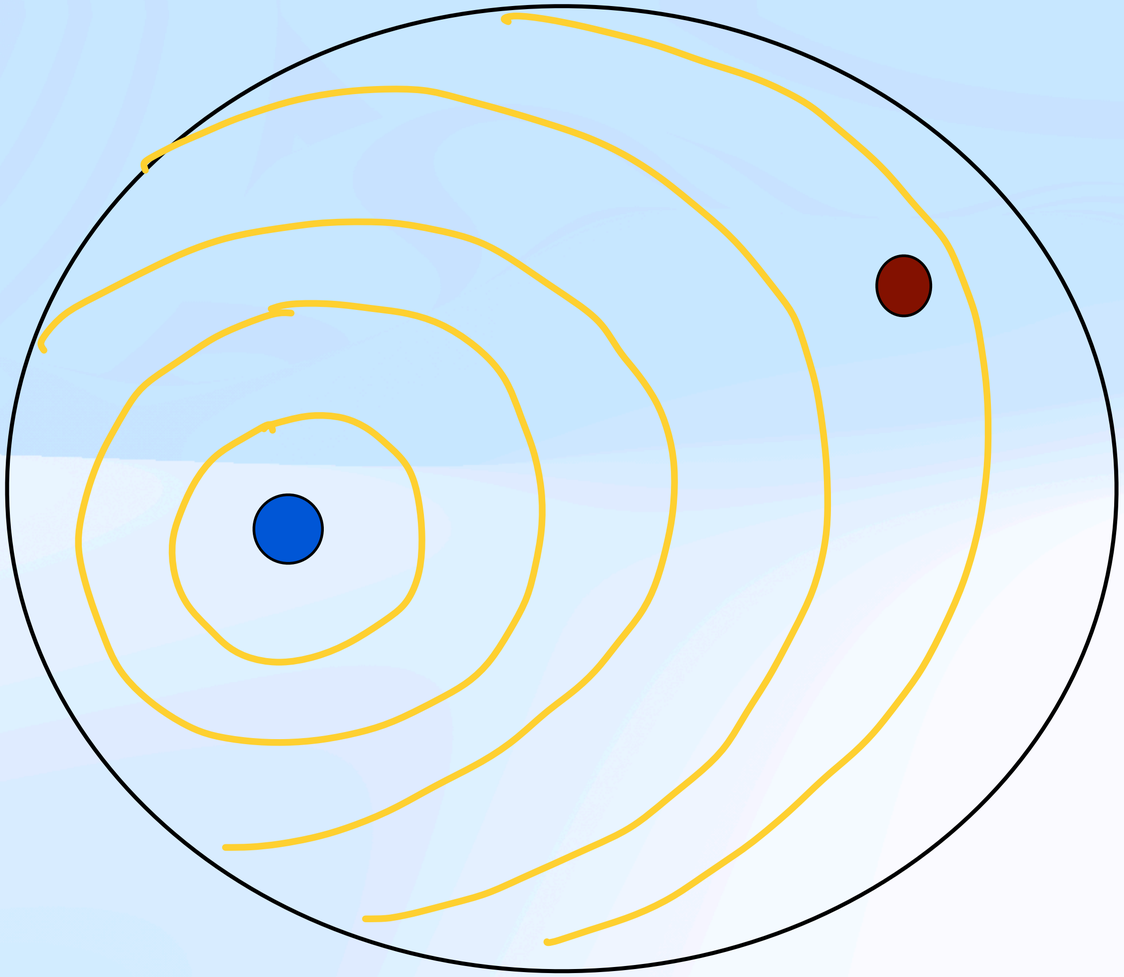
Does it return optimal cost path?

Yes! It gives A → D → G

function f : path cost (start, node) + estimated cost (node, goal)

This gives us the A* Algorithm

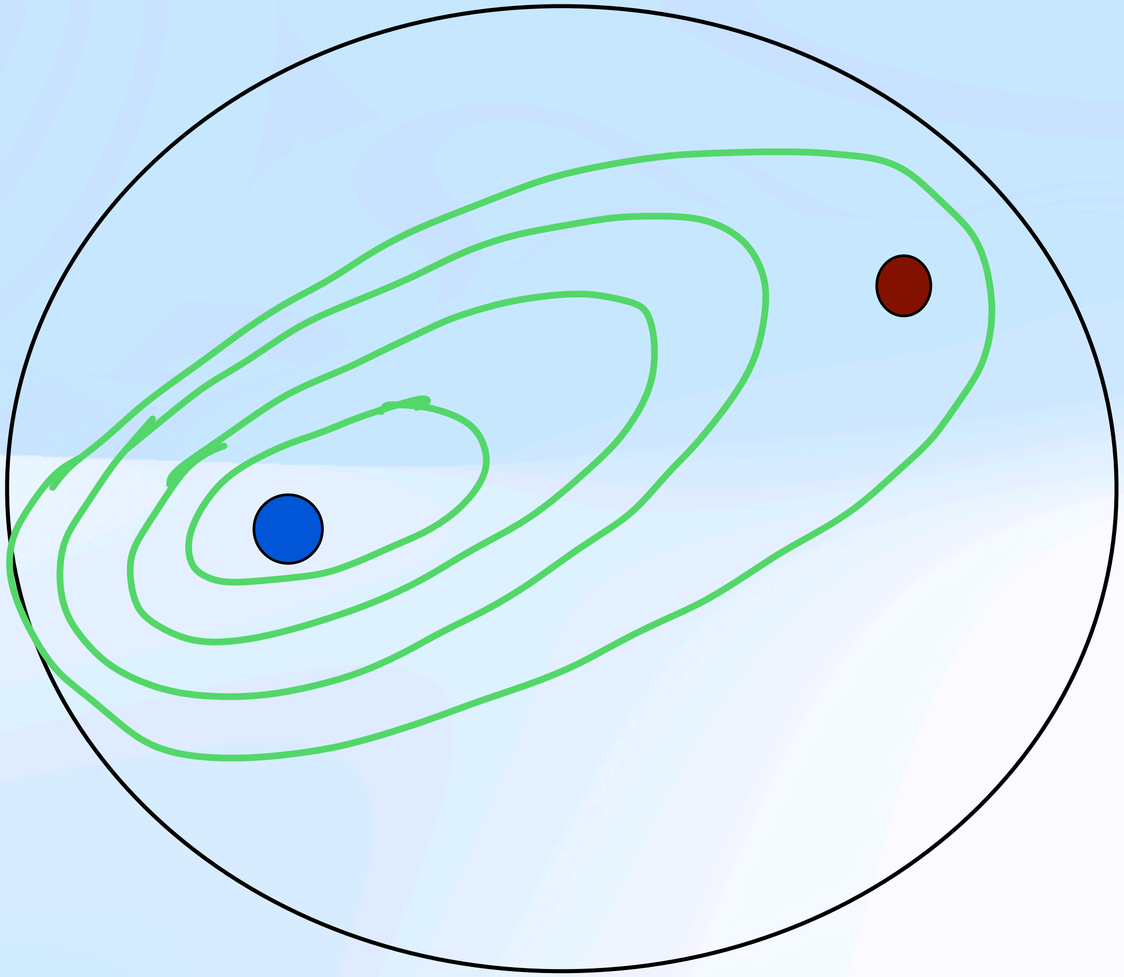
Dijkstra’s algorithm: contours



We have contours of g-cost.

Contours spread equally around the start state with no preference to the goal

A* algorithm: contours..



We have contours of g+h cost.

With a good heuristic function, contours stretch towards a goal and become focused towards goal state.