Sistemi Intelligenti Avanzati
Corso di Laurea in Informatica, A.A. 2020-2021
Università degli Studi di Milano

Uncertainty in Search

Matteo Luperto
Dipartimento di Informatica
matteo.luperto@unimi.it

Search Algorithms

In the first lessons, we have investigated how to solve search problems:

Initial state: A

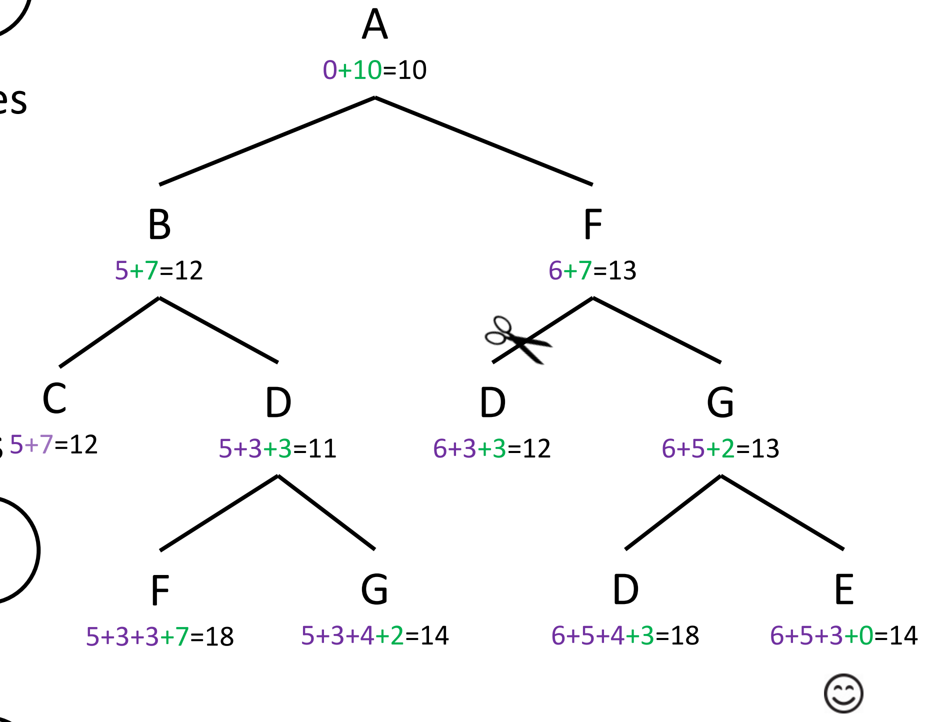
Desired solution: any path to goal state E

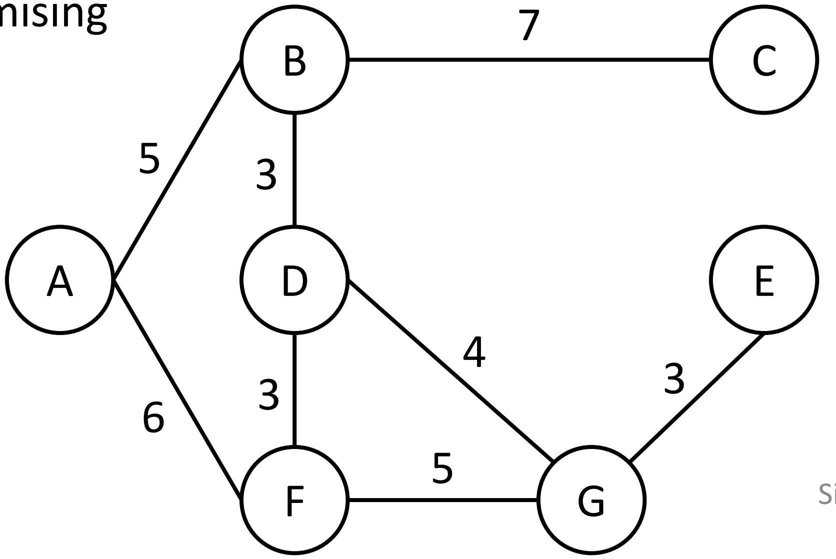
Actions: move between two connected nodes

Transitions: agent is in a new node

Costs: cost for traversing the edge

Heuristic function: estimate of how a state is promising





Search Algorithms

However, we have made several assumption

1. Discrete and finite search space
   vs Continuous and/or infinite search space

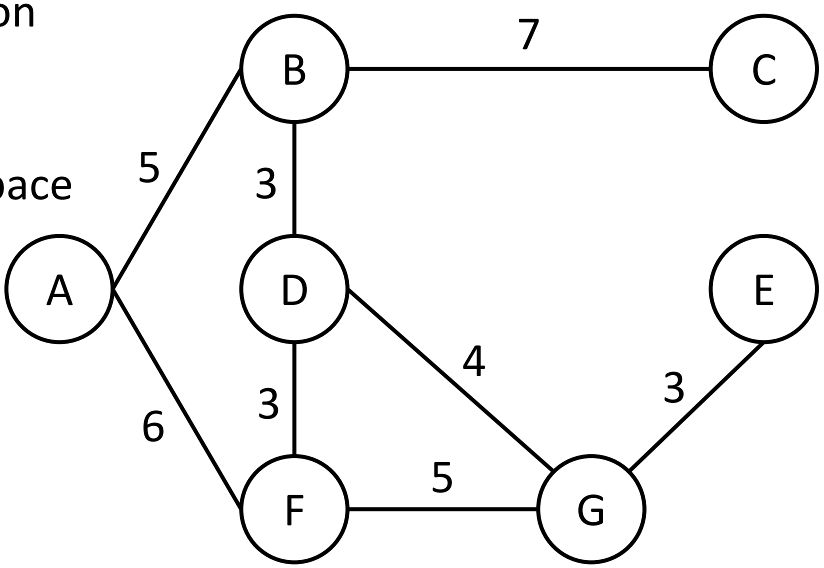
2. Deterministic transitions
   vs Stochastic transitions

3. Observable States
   vs partially-observable states

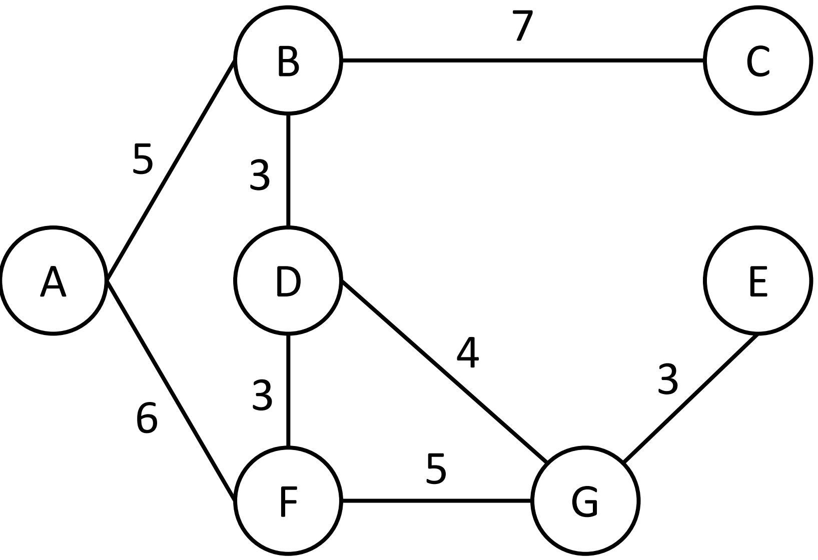
4. Known Search Space
   vs unknown search space

5. Offline search
   vs online search

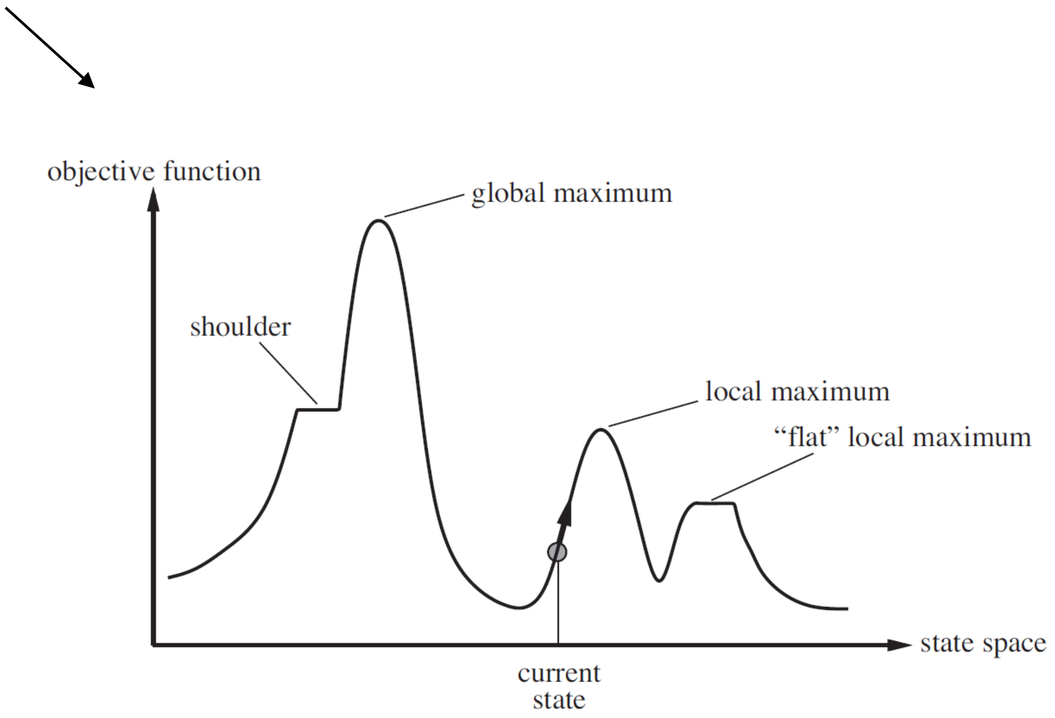
(some of) these assumptions are not admissible for several problems



Discrete vs Continuous State Space



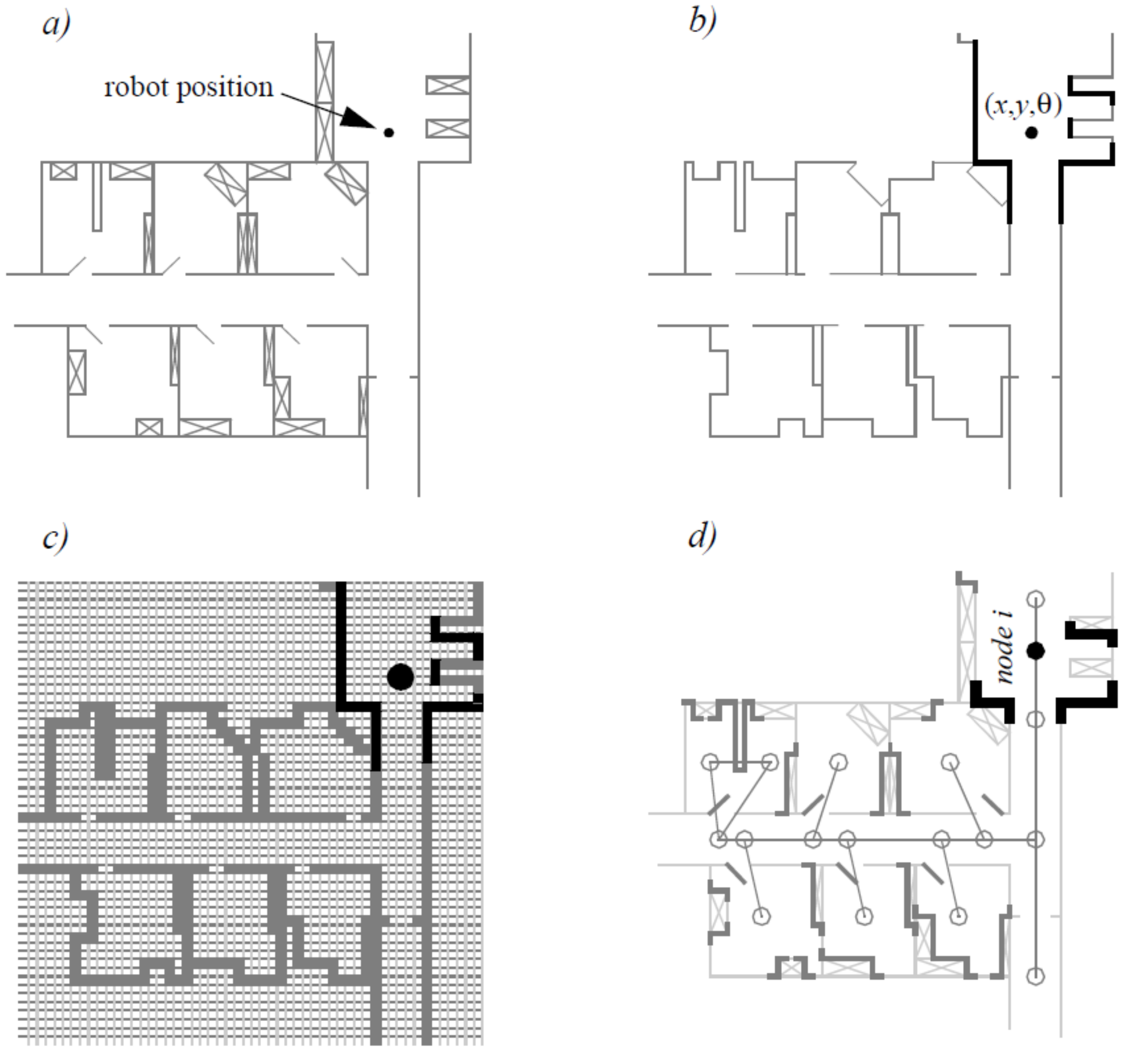
Discretization can be used to reduce a continuous problem to a discrete one (e.g. path planning in grid maps for robots)



From Russel, Norvig, AI A Modern Approach 3ed, Pearson, 2010

If this is not possible, continuous problems are usually solved by different approaches

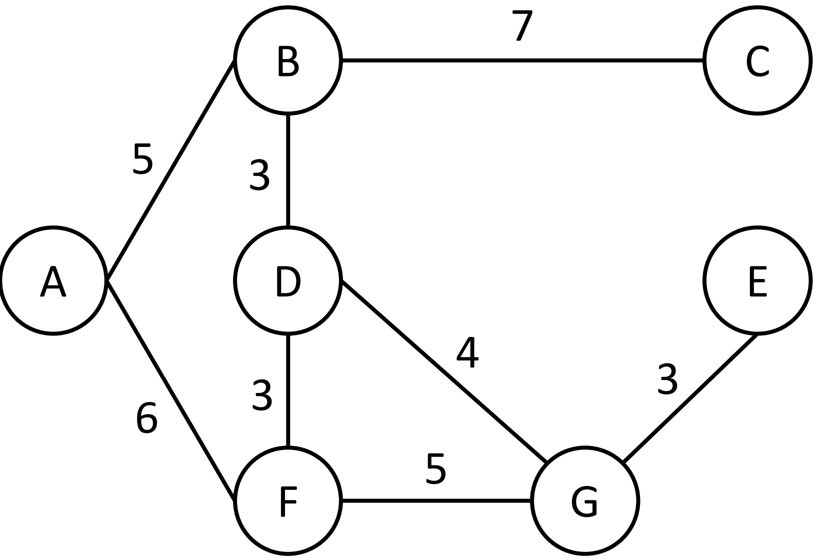
Discrete vs Continuous State Space



From Siegwart,Introduction to Autonomous Mobile Robots, MIT Press 2004

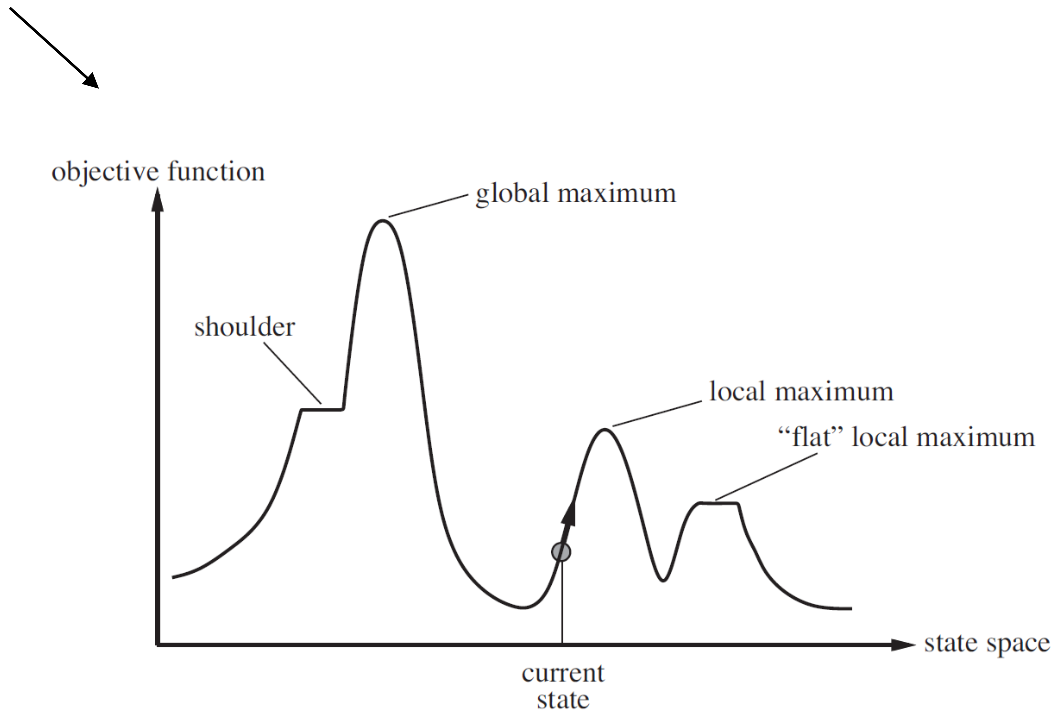
Discretization can be used to reduce a continuous problem to a discrete one (e.g. path planning in grid maps for robots)

Discrete vs Continuous State Space



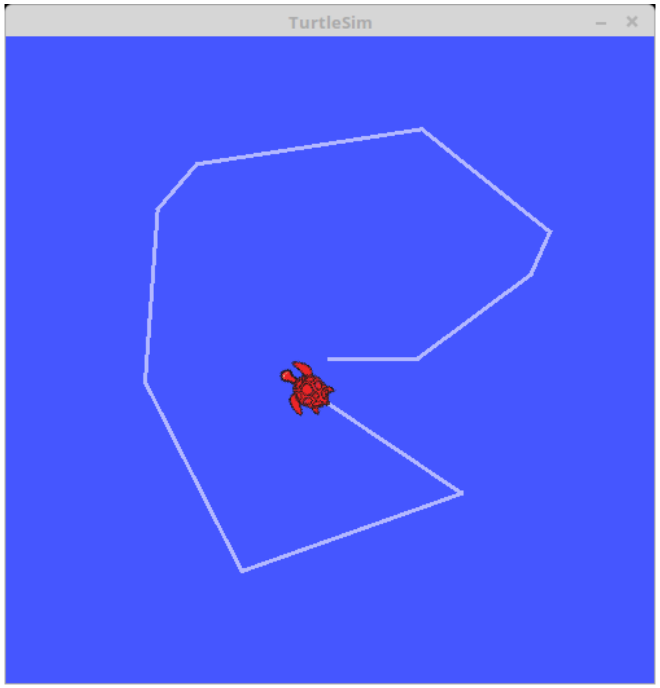
Discretization can be used to reduce a continuous problem to a discrete one (e.g. path planning in grid maps for robots)

If this is not possible, continuous problems are usually solved by different approaches

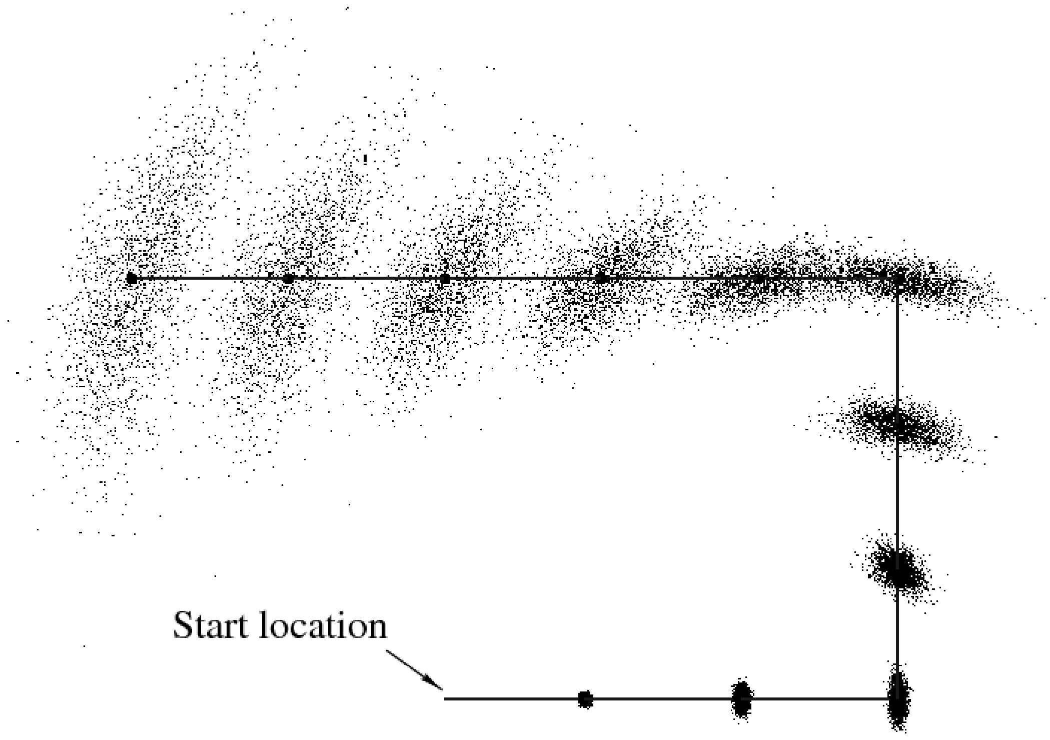


From Russel, Norvig, AI A Modern Approach 3ed, Pearson, 2010

Deterministic vs Stochastic Transitions



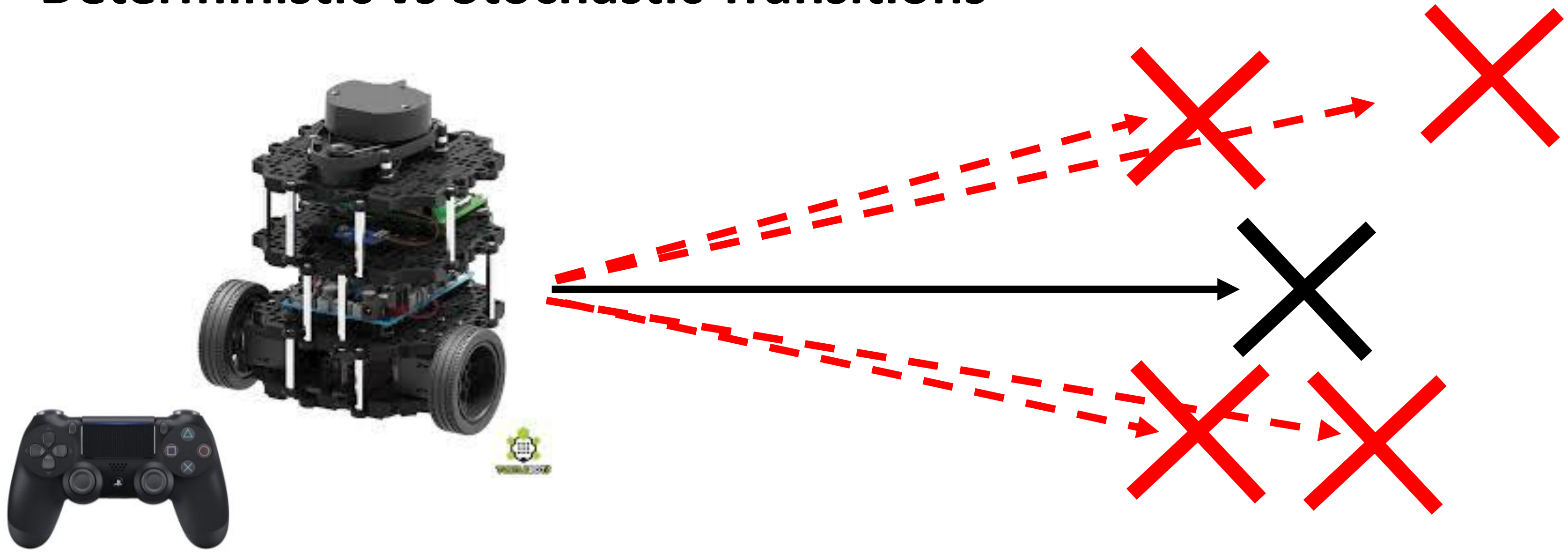
In general, we want to know the outcome of an action



From Thrun Burgard Fox, Probabilistic Robotics, MIT Press 2006

In practice, this is not often possible

Deterministic vs Stochastic Transitions

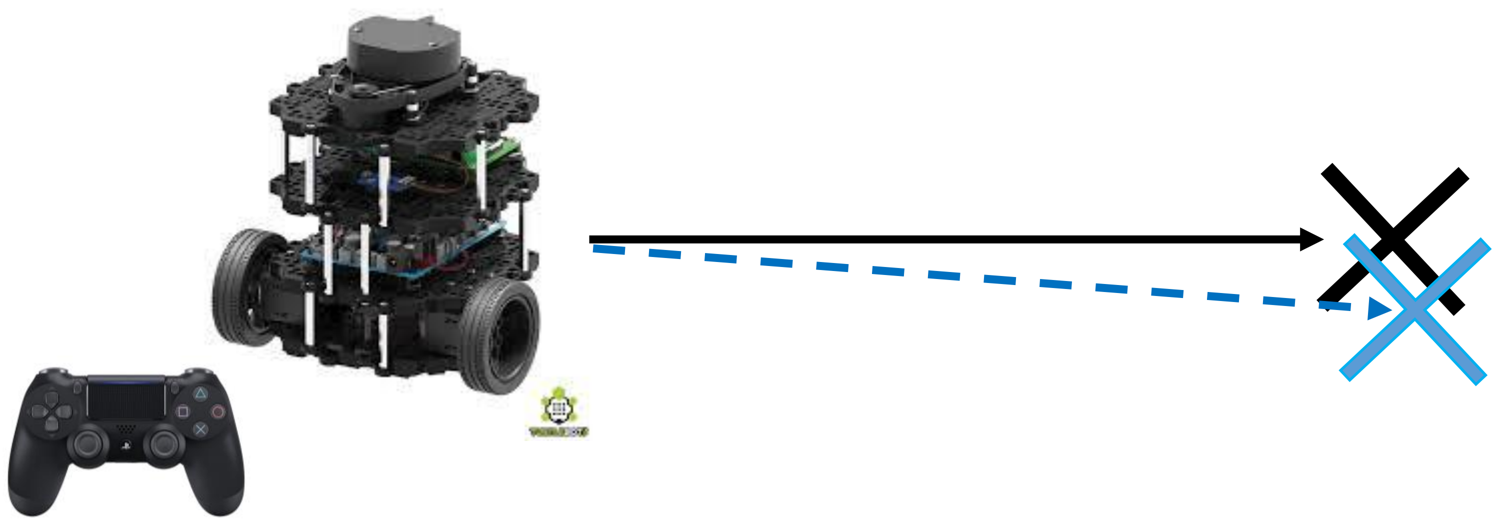


Assuming that an action has a deterministic outcome is like moving around blindfolded.

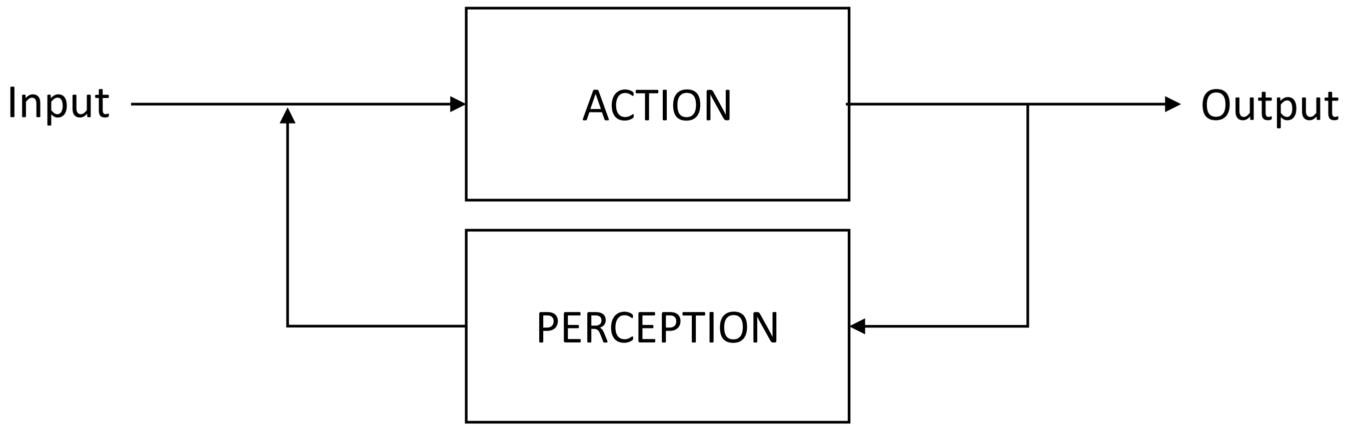
$$\text{Input} \longrightarrow \boxed{\text{ACTION}} \longrightarrow \text{Output}$$

Open Loop = we trust the outcome of an action

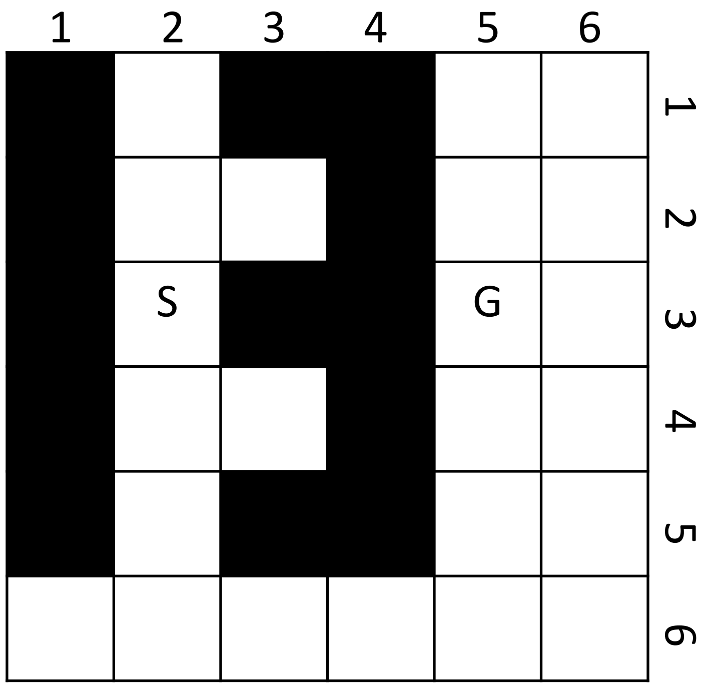
Deterministic vs Stochastic Transitions



What we do: use perception to observe the outcome of an action



Stochastic Transitions



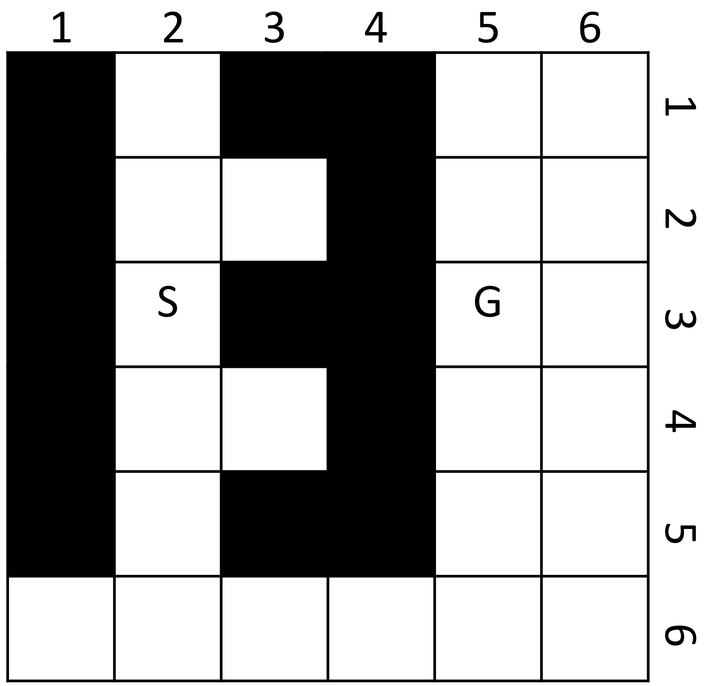
Initial State: S

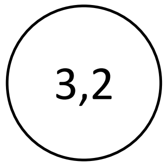
Goal State: G

Actions: ◀ ▶ ▲ ▼, costs = 1

Stochastic Transitions: P = 0.6 desired state, P = 0.4 of another state

Stochastic Transitions





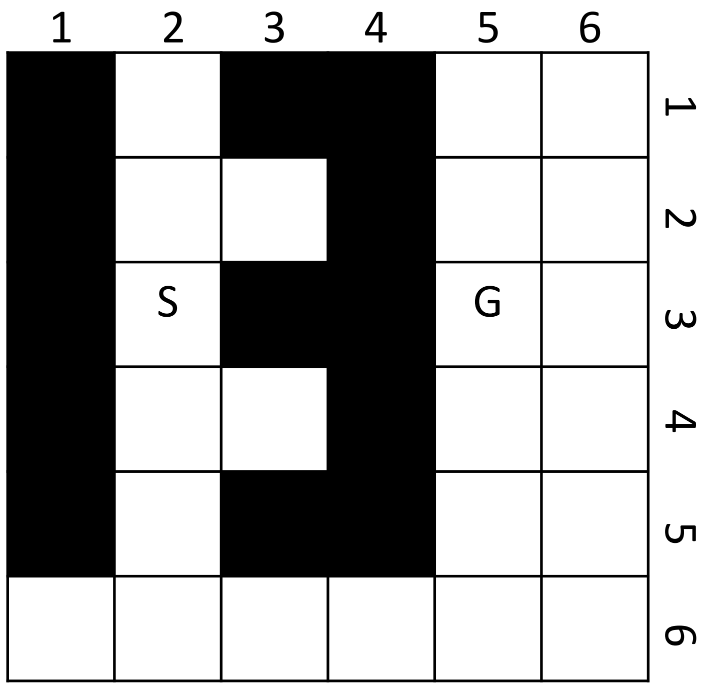
Initial State: S

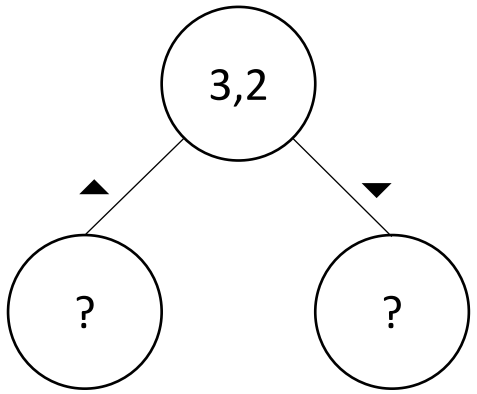
Goal State: G

Actions: ◀ ▶ ▲ ▼, costs = 1

Stochastic Transitions: P = 0.6 desired state, P = 0.4 another state

Stochastic Transitions





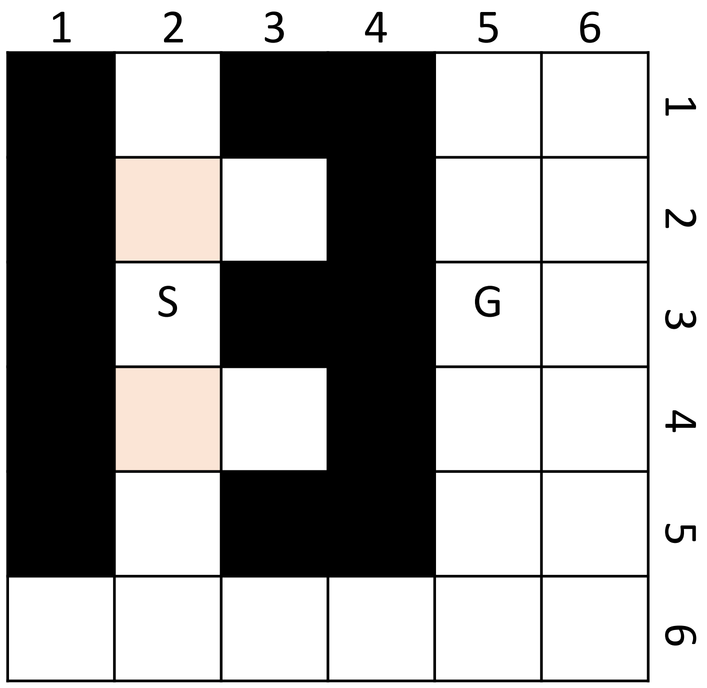
Initial State: S

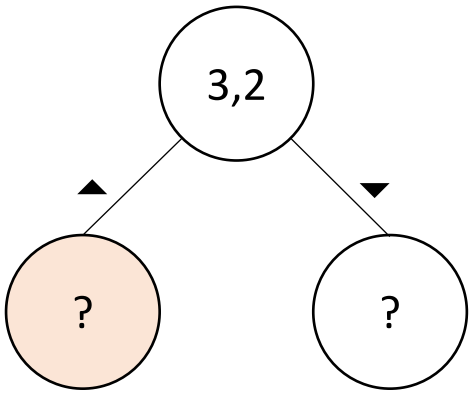
Goal State: G

Actions: ◀ ▶ ▲ ▼, costs = 1

Stochastic Transitions: P = 0.6 desired state, P = 0.4 another state

Stochastic Transitions

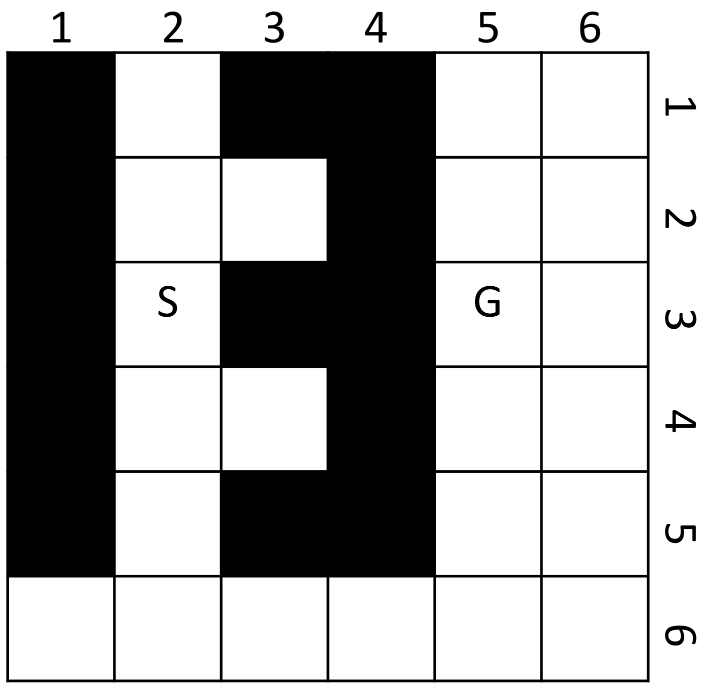


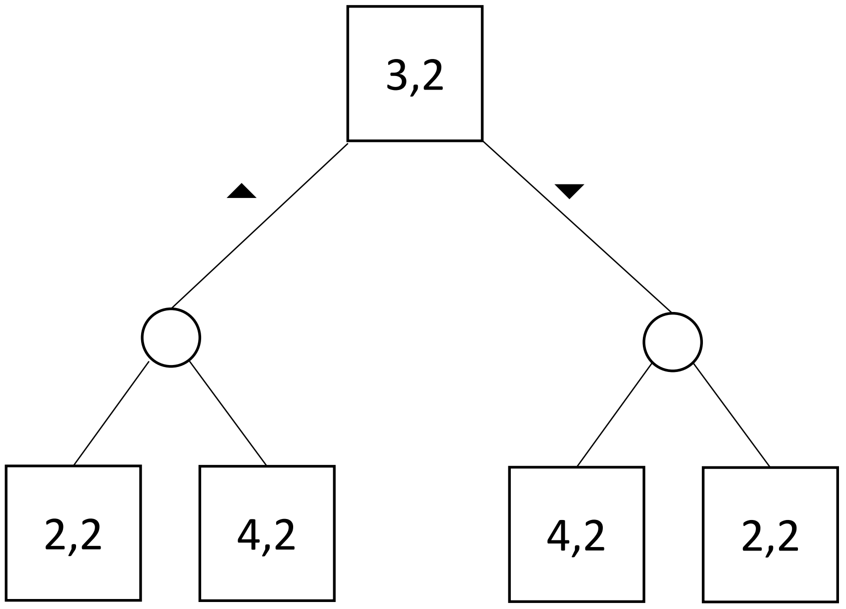


Applying both possible actions, I have a != 0 probability of ending in the same states

How can we model this for applying a search strategy?

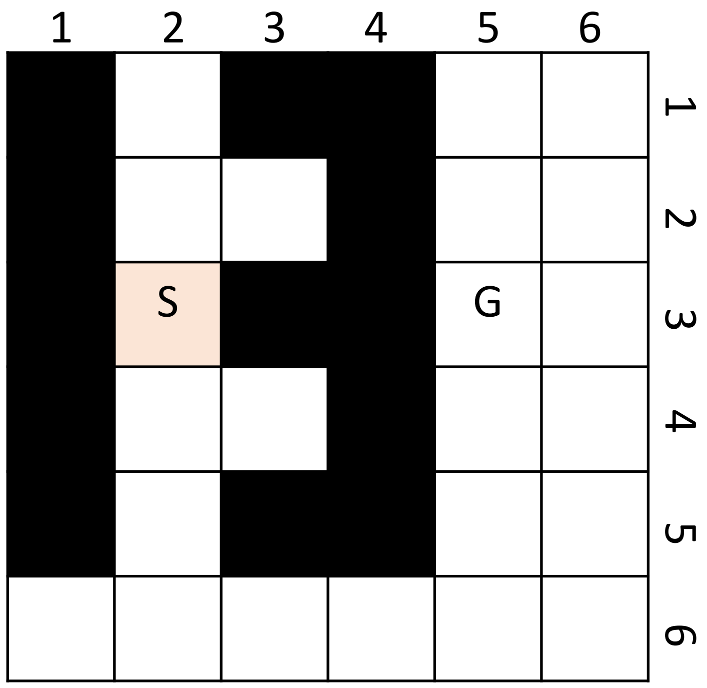
AND-OR Search Trees

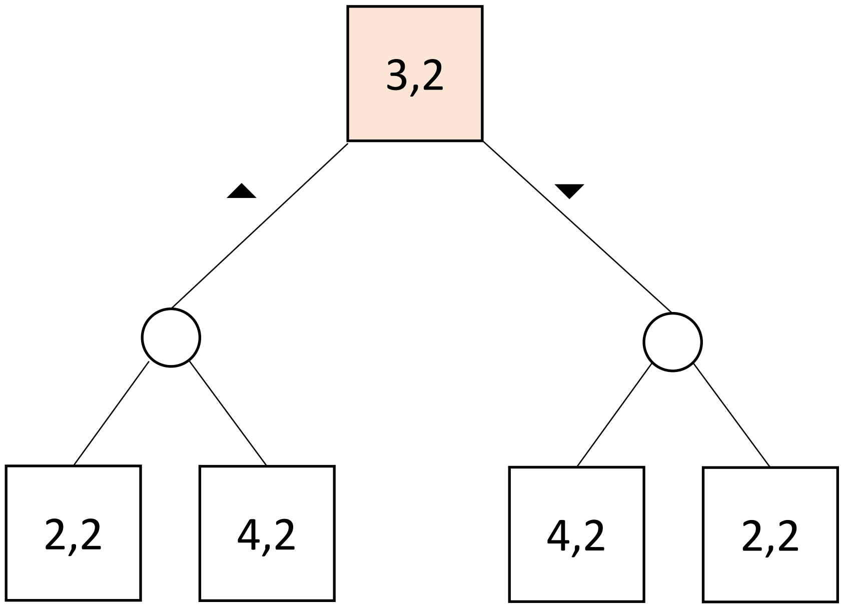




- OR nodes are state nodes (squares), where some action must be chosen
- AND nodes (circles) represents every possible outcome of an action
- A solution is a <u>subtree</u> that
    1. Has a goal node at every leaf
    2. Specifies on action of each OR nodes
    3. Includes every outcome branch at each of its AND nodes

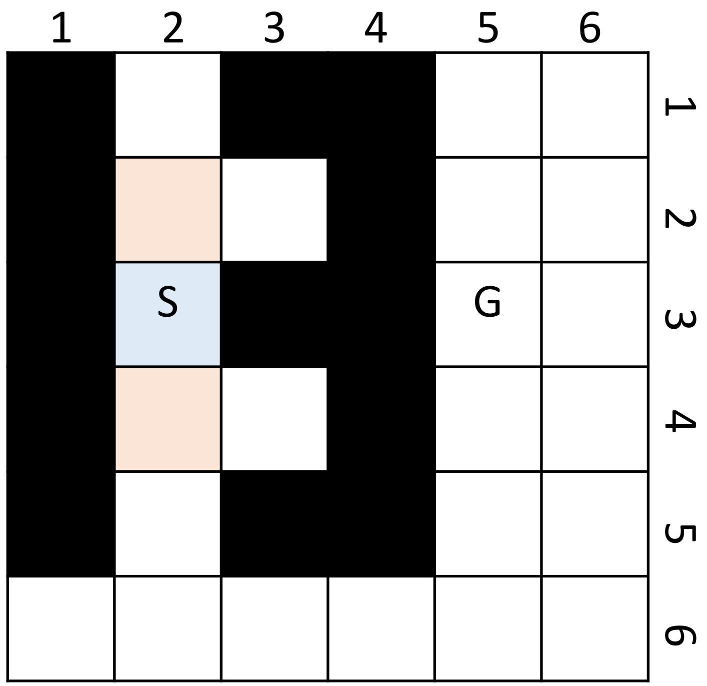
AND-OR Search Trees

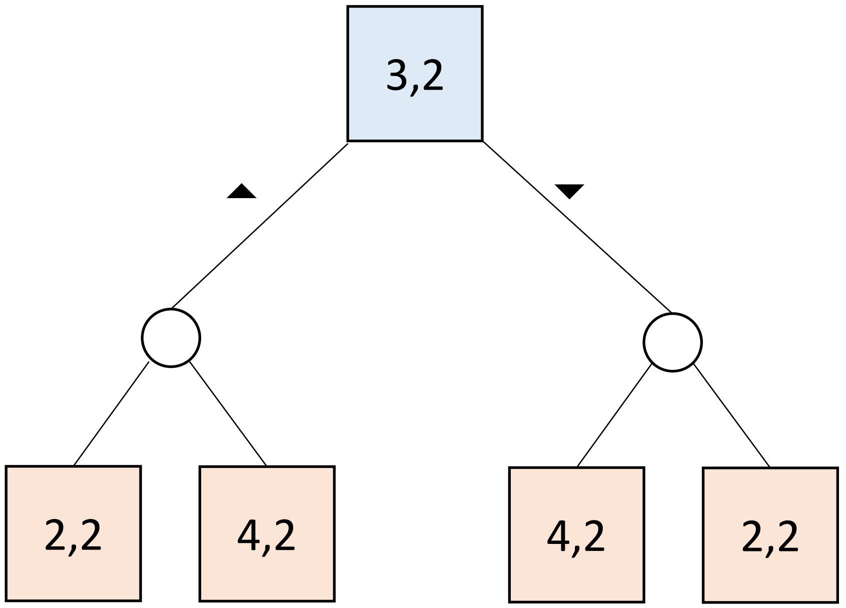




- Stochastic transitions increase the branching factor and the search space
- LOOPS should be detected as are frequent

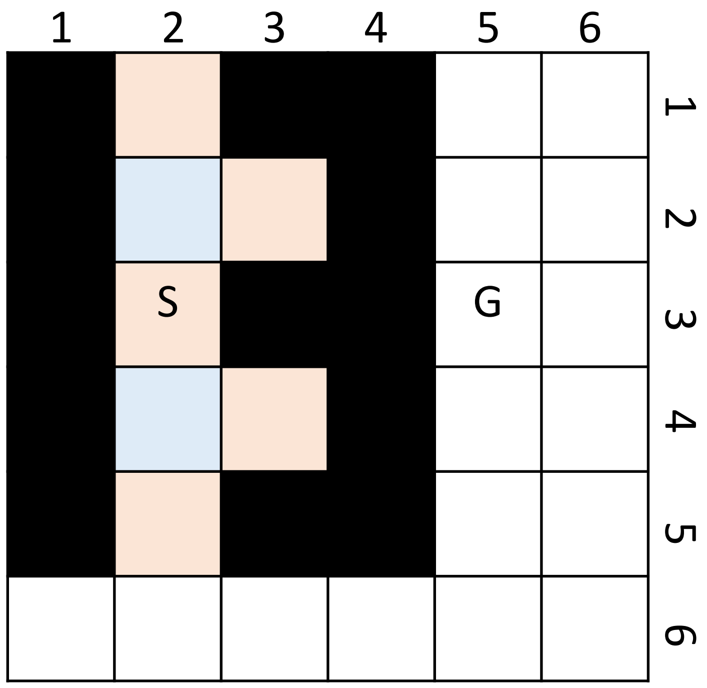
AND-OR Search Trees

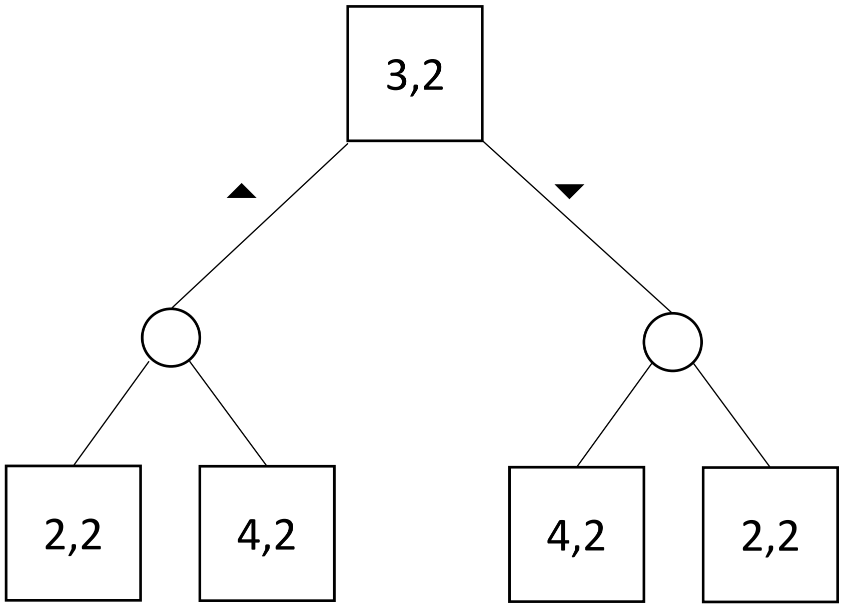




- Stochastic transitions increase the branching factor and the search space
- LOOPS should be detected as are frequent

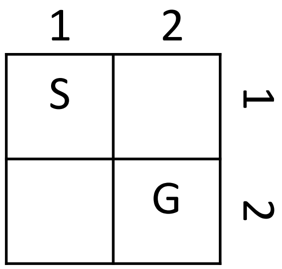
AND-OR Search Trees

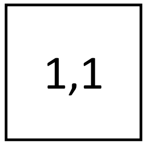




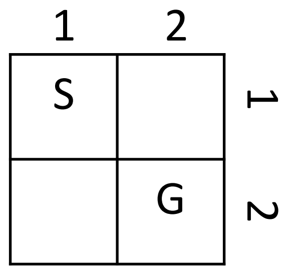
- Stochastic transitions increase the branching factor and the search space
- LOOPS should be detected as are frequent
- A solution is a <u>subtree</u> where every leaf is a goal node

AND-OR Search Trees

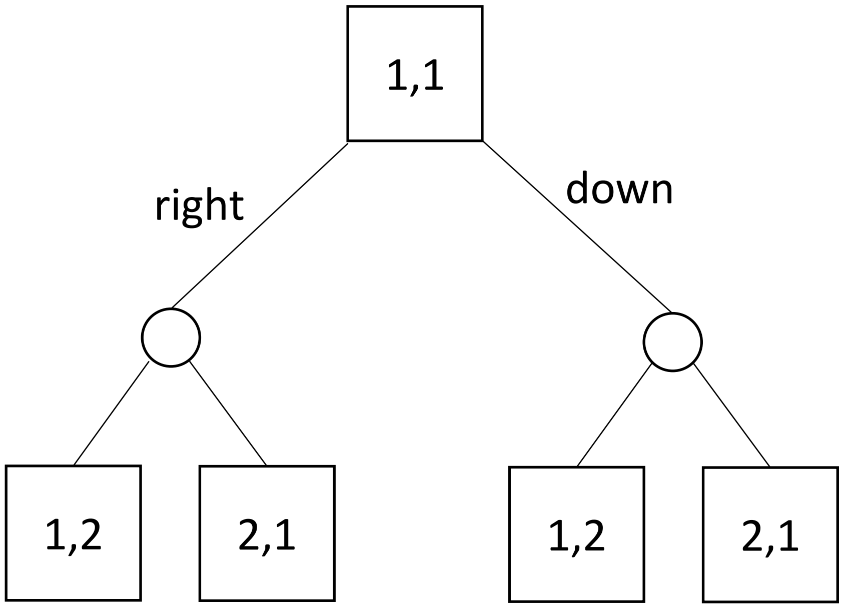




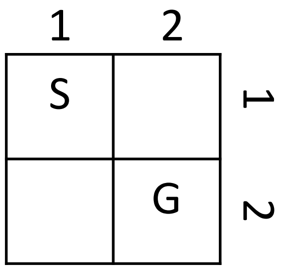
AND-OR Search Trees

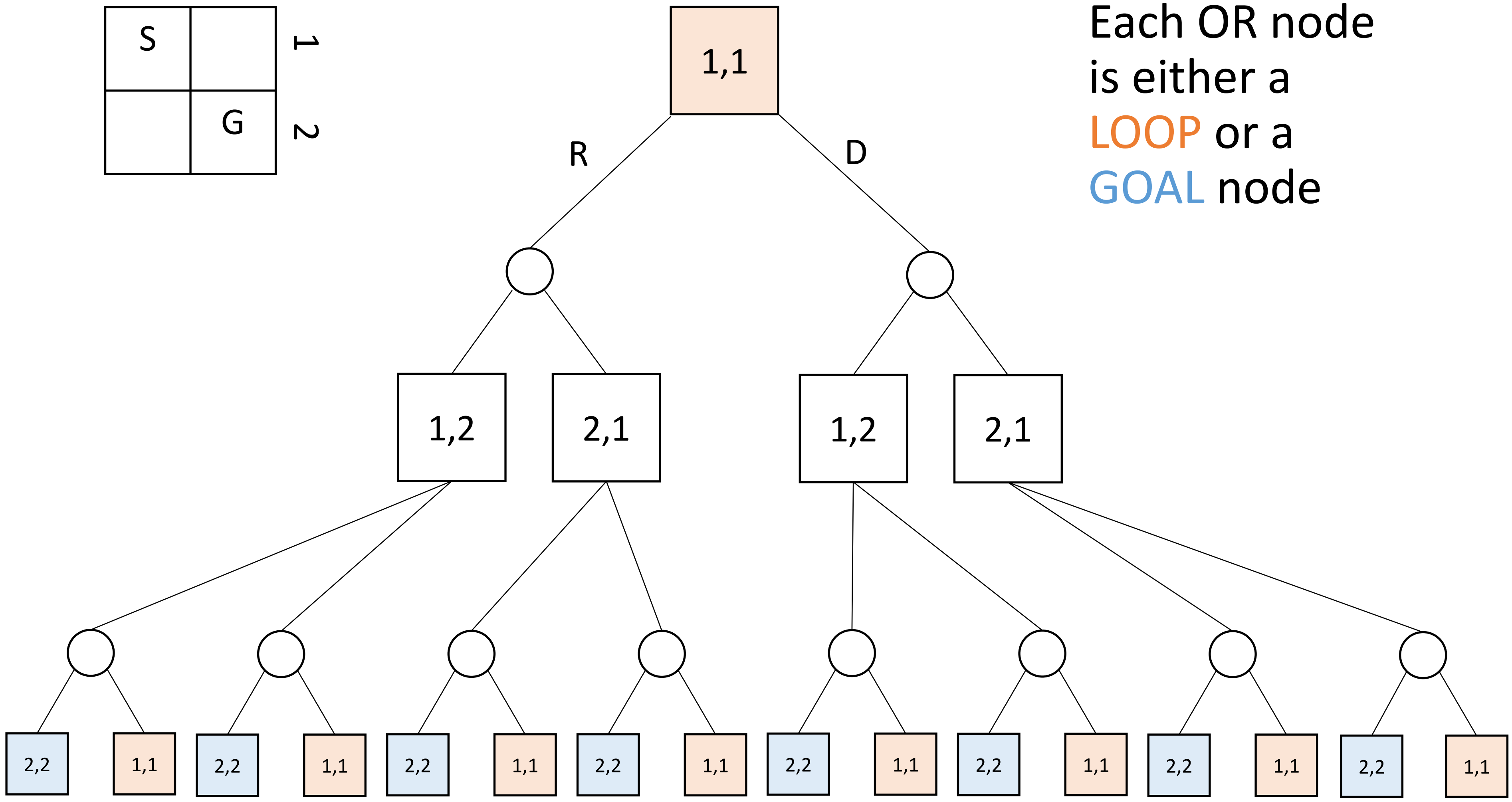


L Left
R Right
D Down
U Up



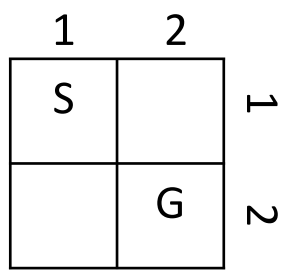
AND-OR Search Trees

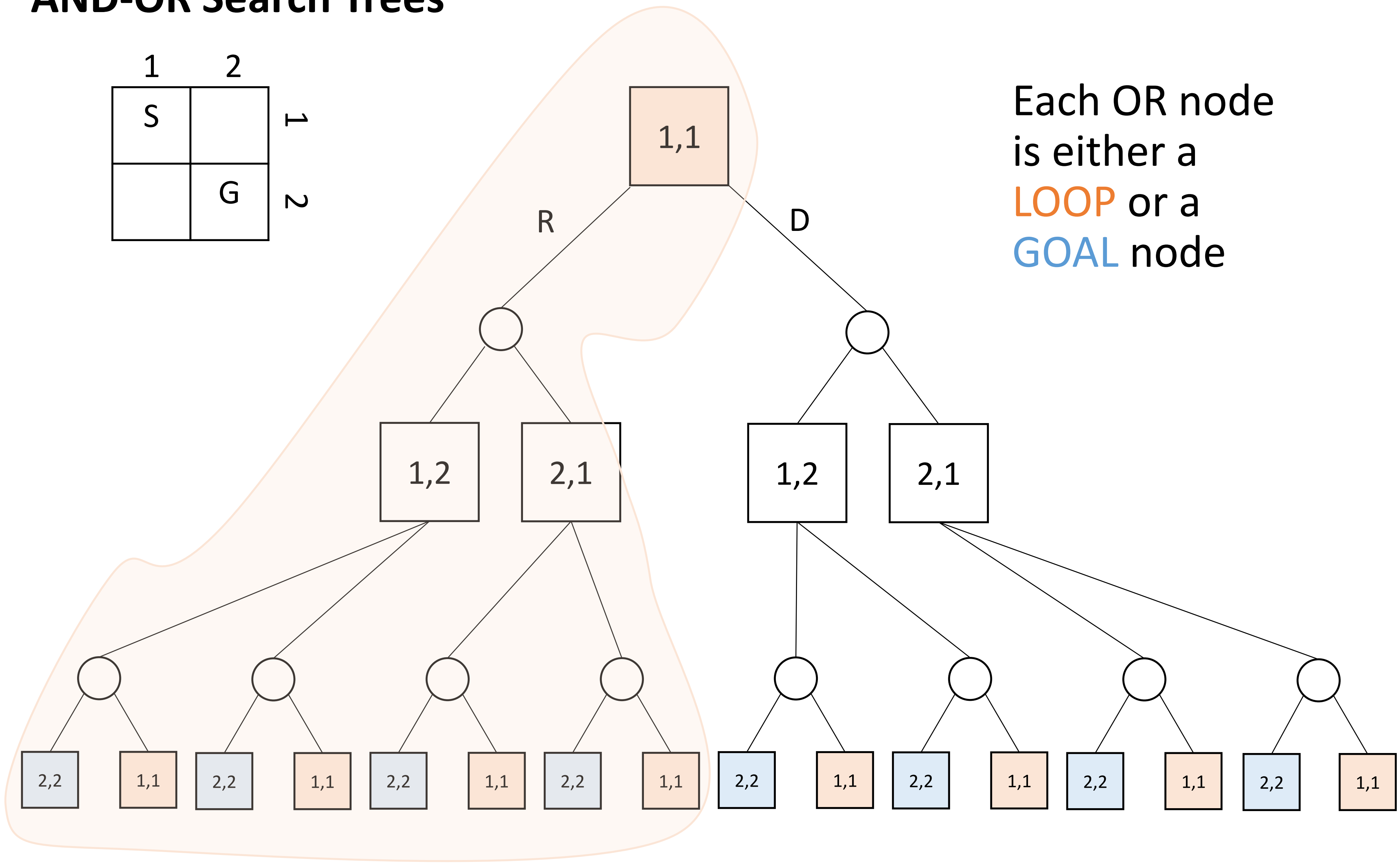




Each OR node is either a LOOP or a GOAL node

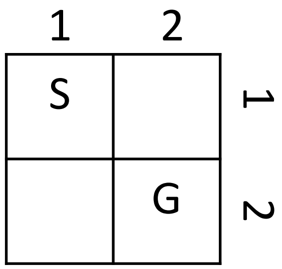
AND-OR Search Trees

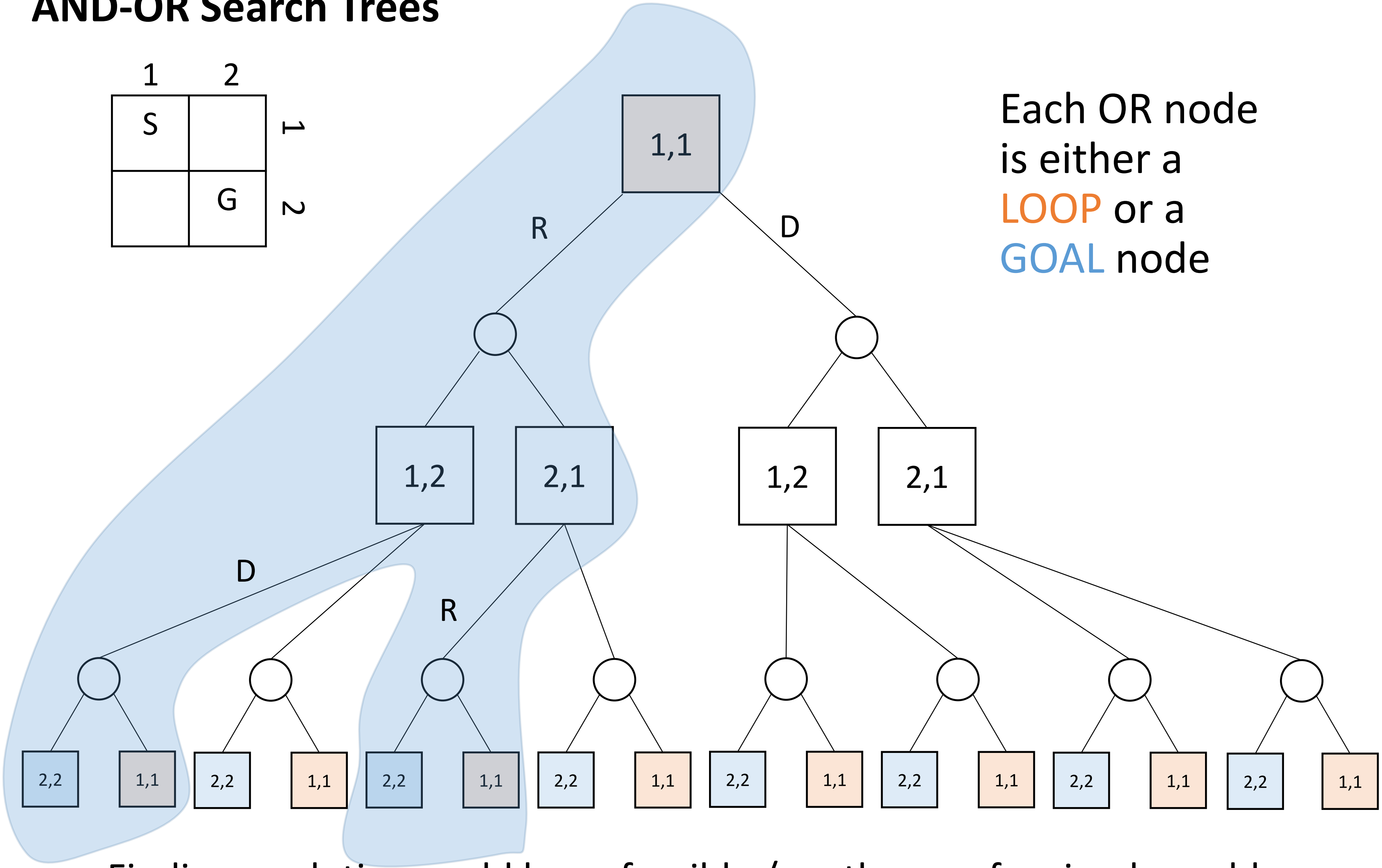




Each OR node is either a LOOP or a GOAL node

AND-OR Search Trees

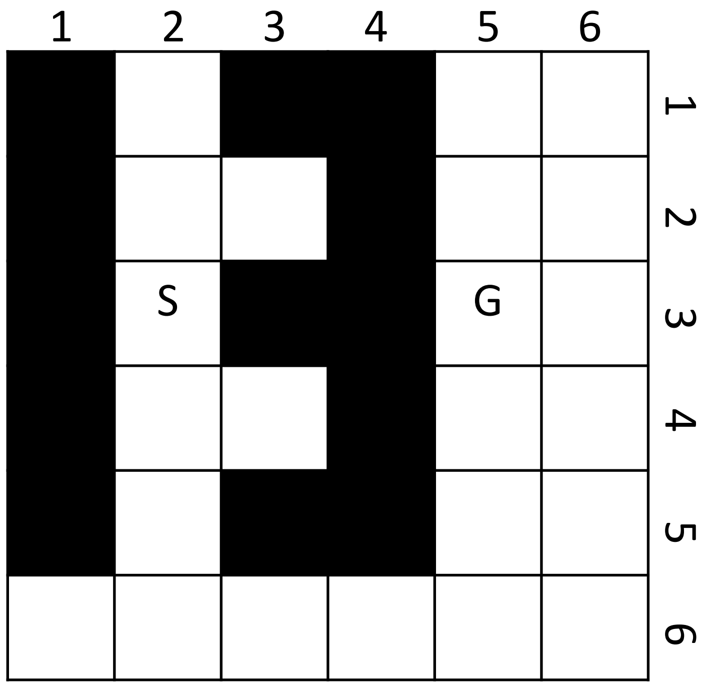




Each OR node is either a LOOP or a GOAL node

Finding a solution could be unfeasible / costly even for simple problems

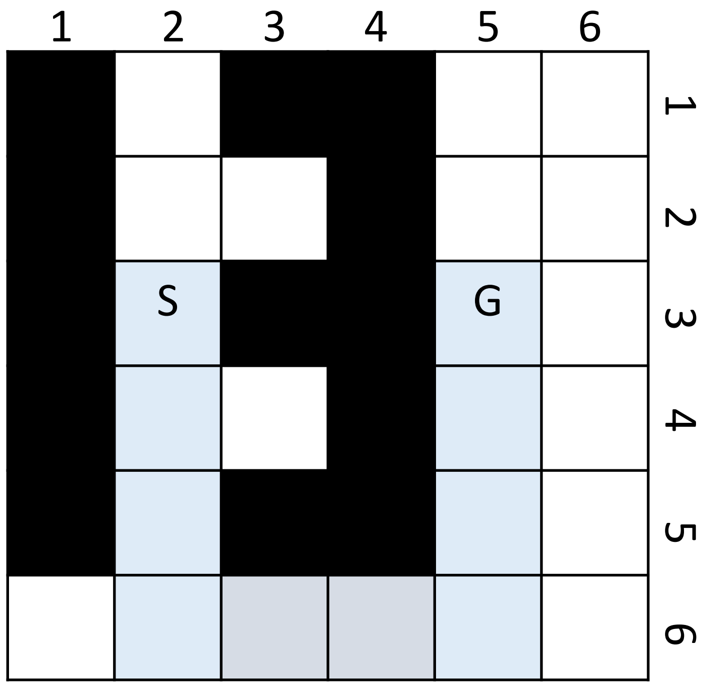
Observable and Partially-Observable States



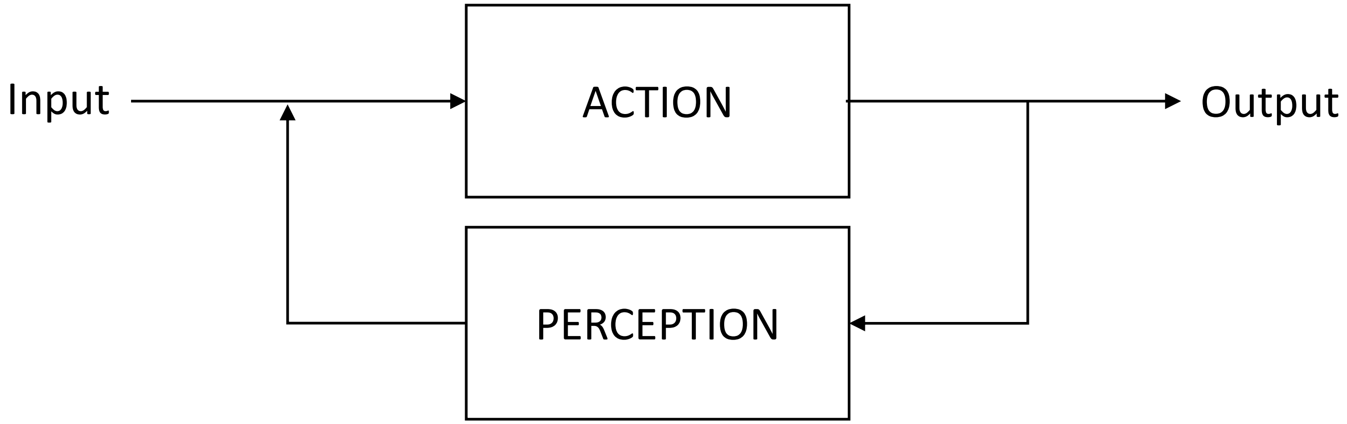
1. Problem formulation (with stochastic transitions)
2. We solve the problem (offline) using a search algorithm
3. We find a solution (e.g., using AND-OR Search methods)

What happens if we want to apply such solution?

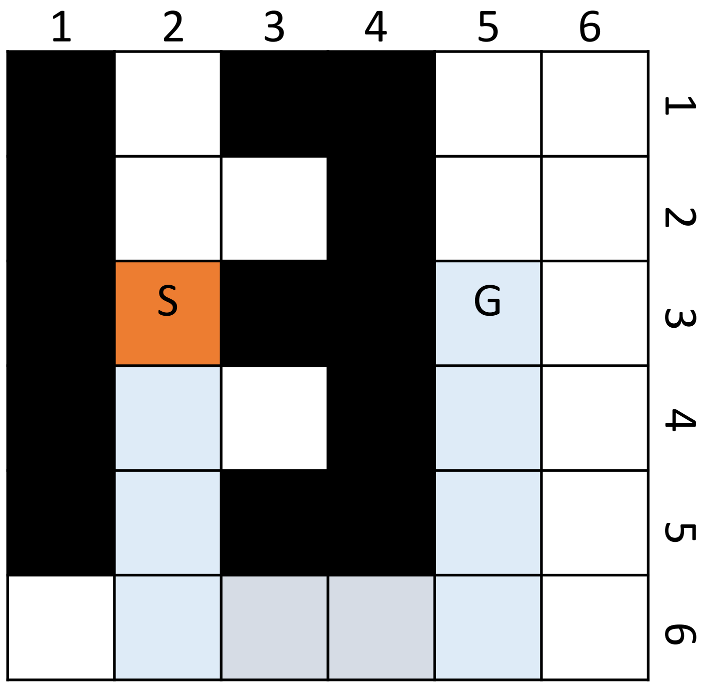
Observable and Partially-Observable States

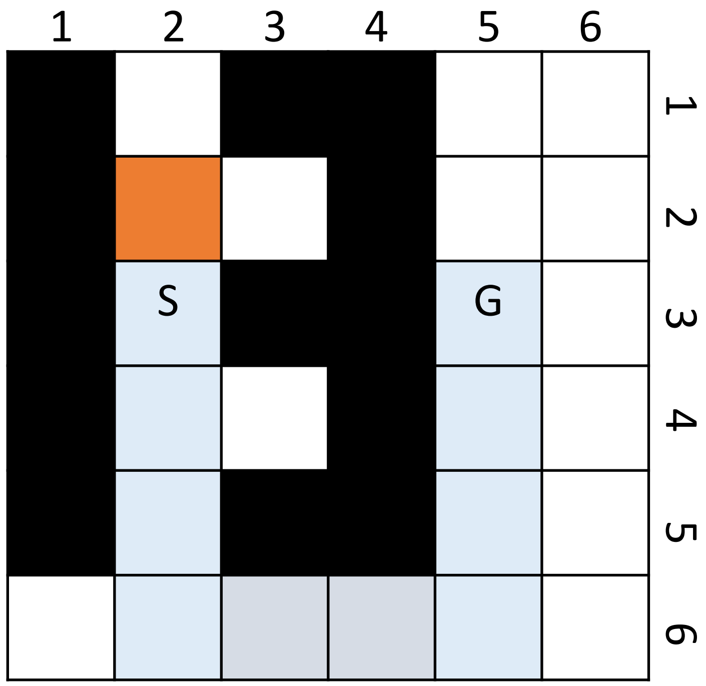


- We have a plan, but
- We do not know the outcome of action, how we know our state?
- We use perception to observe the state we ended with



Observable and Partially-Observable States

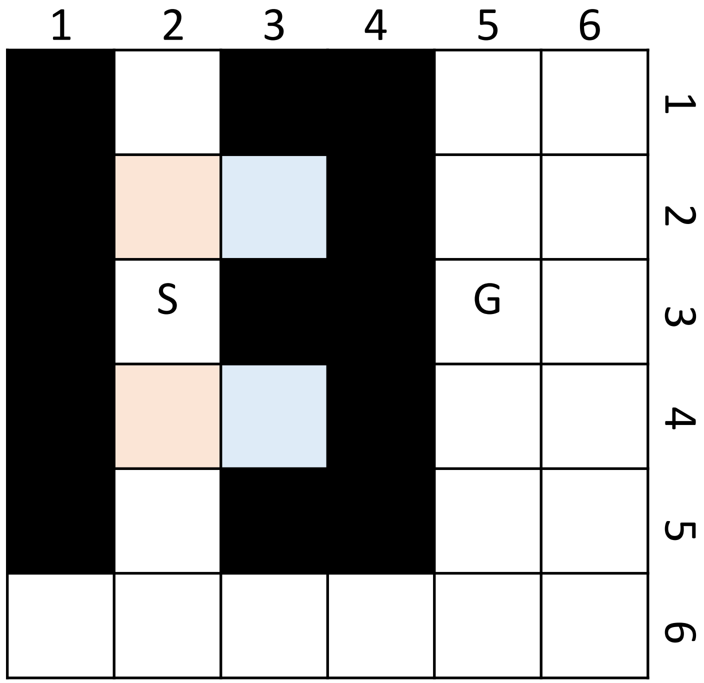




We start in S, and we try to go down, however, we end in (2,2):

- Fully-observable state = we know that we are in (2,2) with perception

Observable and Partially-Observable States



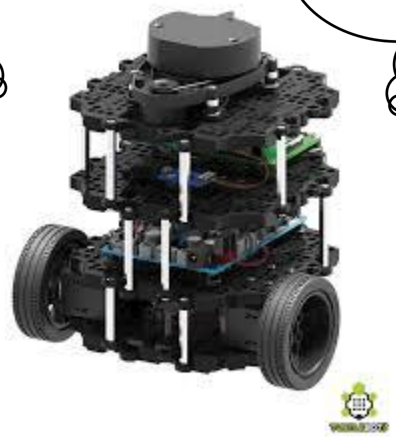
However, in several problems also this assumption does not hold.

We need also to estimate the state we end with after an action as the environment is only **partially-observable**.

Planning under uncertainty

Are the effects of my actions perfectly predictable?

Am I always sure about what's going on?



Deterministic
vs
Stochastic
transitions

Fully-observable
vs
Partially-observable
states

Searching with Partial Observation

The agent does not know its exact state, but it has a

Belief states, an estimate of its current states given the sequence of actions and percepts up to that point

- A set of states
- A set of actions
- A transition model
- A set of costs
- A goal test
- **Belief states** $\longleftarrow$ new!

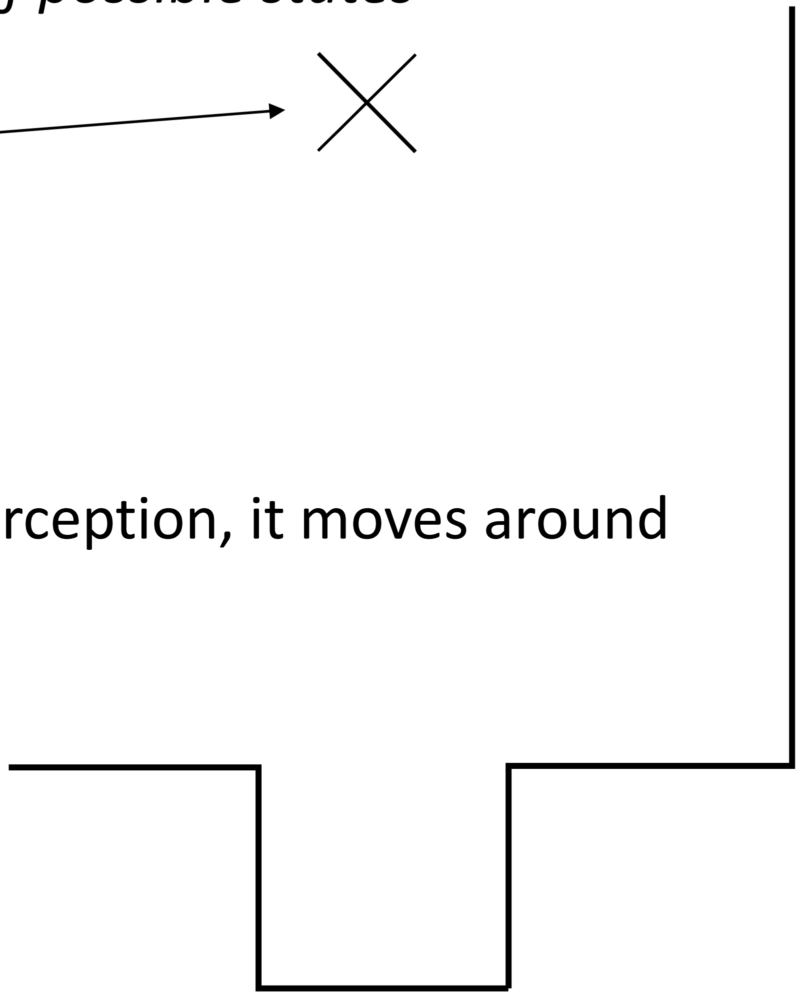
Searching with Partial Observations: an example

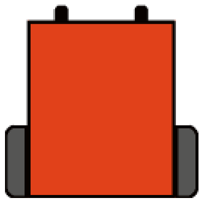
Initial state

I know where am I and I have a model of possible states

a) a discrete and known grid map

b) initial cell is known





We have a robot, with perception, it moves around

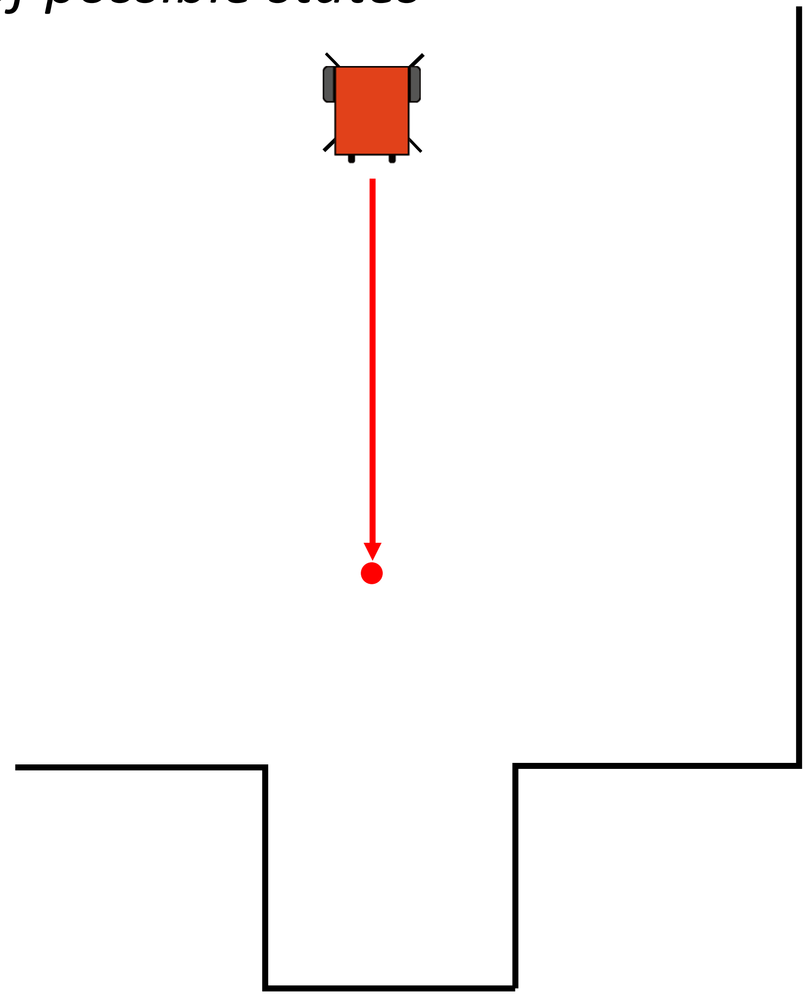
Searching with Partial Observations: an example

Initial state

I know where am I and I have a model of possible states

1. Action:
   "I want to move forward 2m"

The agent tries to perform some action, having a transition model



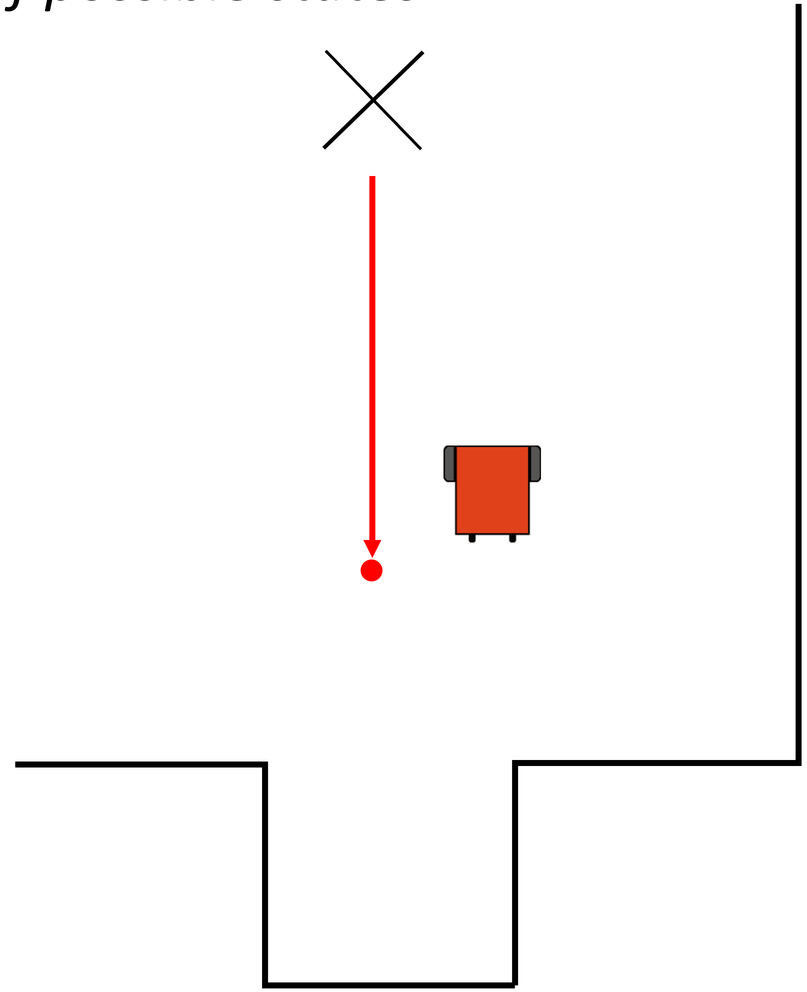
Searching with Partial Observations: an example

Initial state

I know where am I and I have a model of possible states

1. Action:
   "I want to move forward 2m"
2. Prediction
   "I am on the correct position ● "

The transition model computes, from its previous belief, a predicted belief (●)

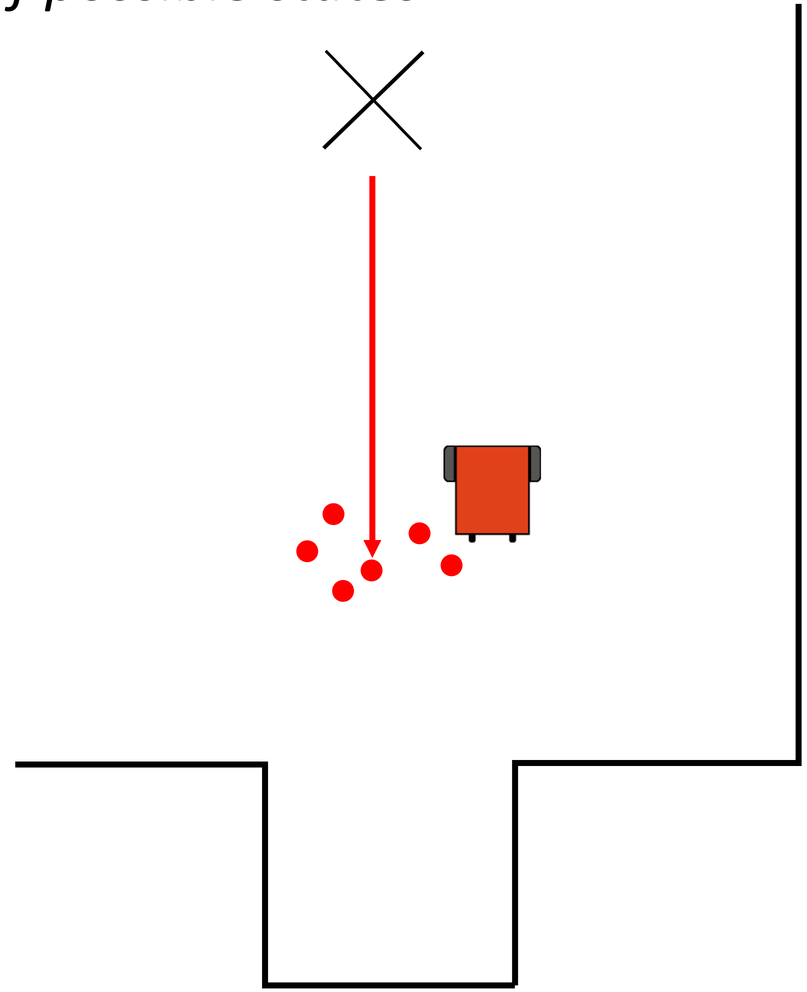


Searching with Partial Observations: an example

Initial state

I know where am I and I have a model of possible states

1. Action:
   "I want to move forward 2m"
2. Prediction
   "I am on the correct position ● "



The predicted belief (●) is a set of states as we do not know the actual outcome of the action.

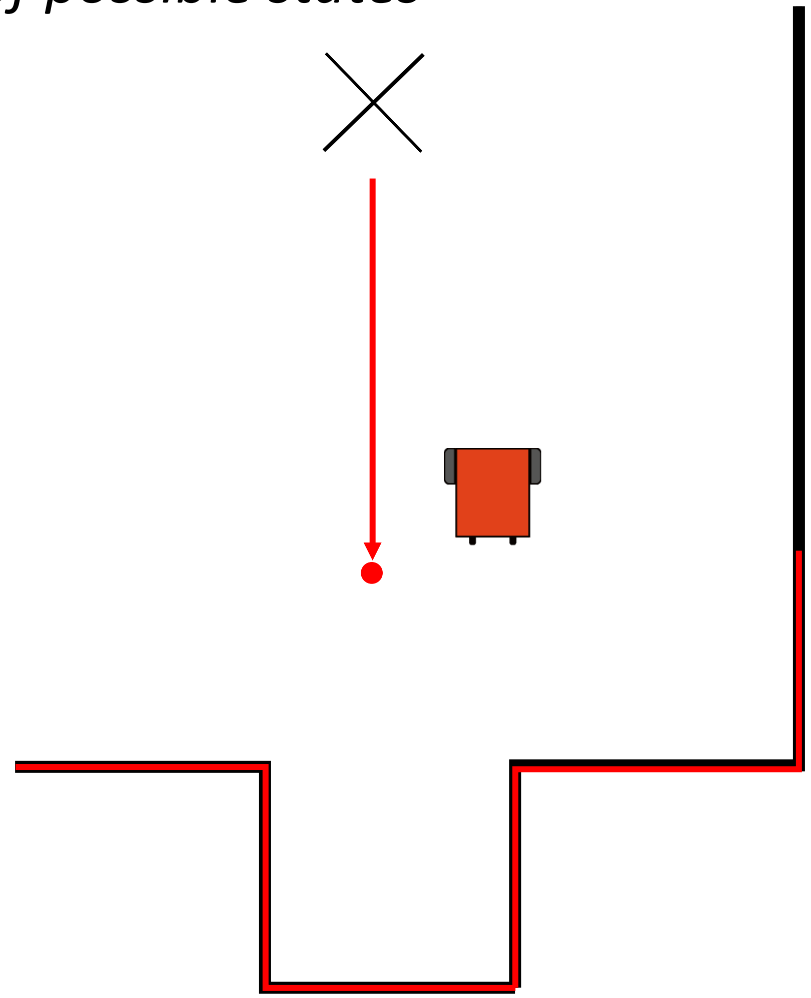
(let's consider one belief state for the sake of simplicity)

Searching with Partial Observations: an example

Initial state
I know where am I and I have a model of possible states

1. Action:
   "I want to move forward 2m"
2. Prediction
   "I am on the correct position ● "
3. Observation prediction
   "I should see walls"

From the predicted belief (●) the robot can estimate the possible percept

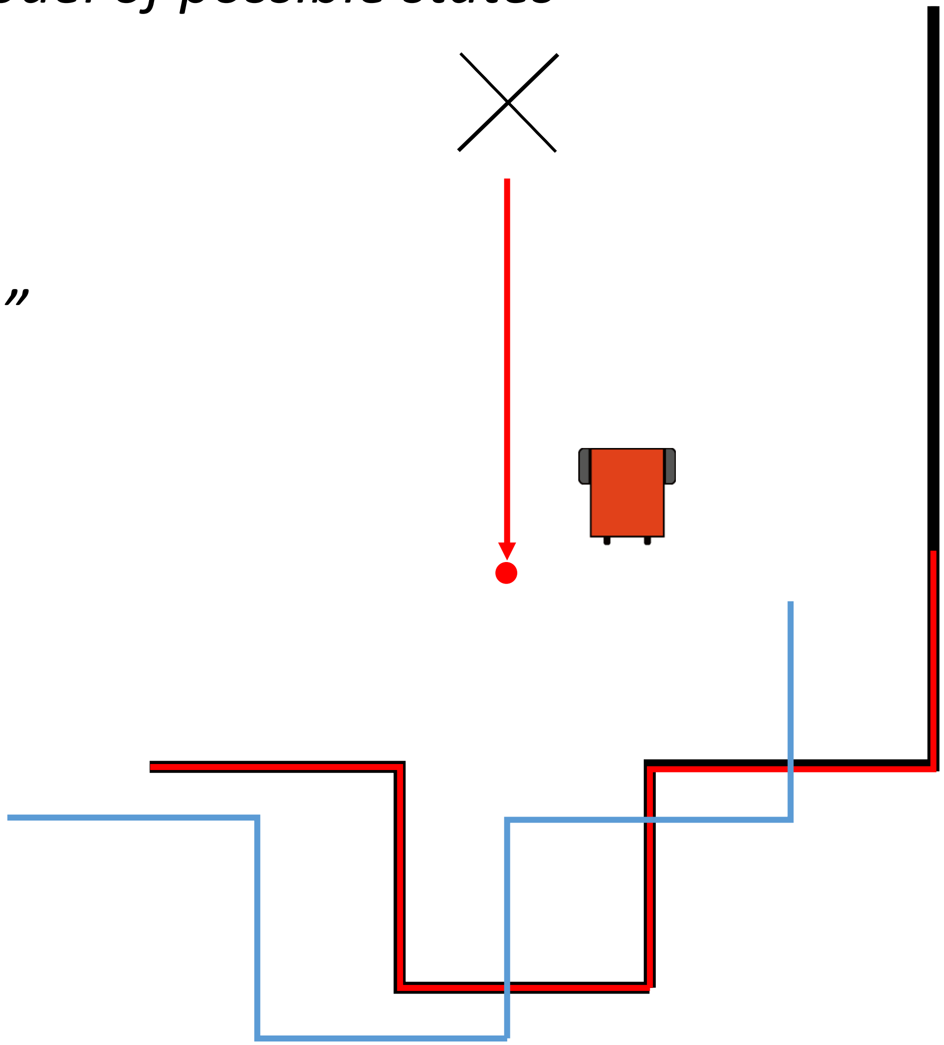


Searching with Partial Observations: an example

Initial state

I know where am I and I have a model of possible states

1. Action:
   “I want to move forward 2m”
2. Prediction
   “I am on the correct position ● ”
3. Observation prediction
   “I should see walls”
4. Perception:
   “I see this”



We can observe if perception evidence support our belief or if there are discrepancies

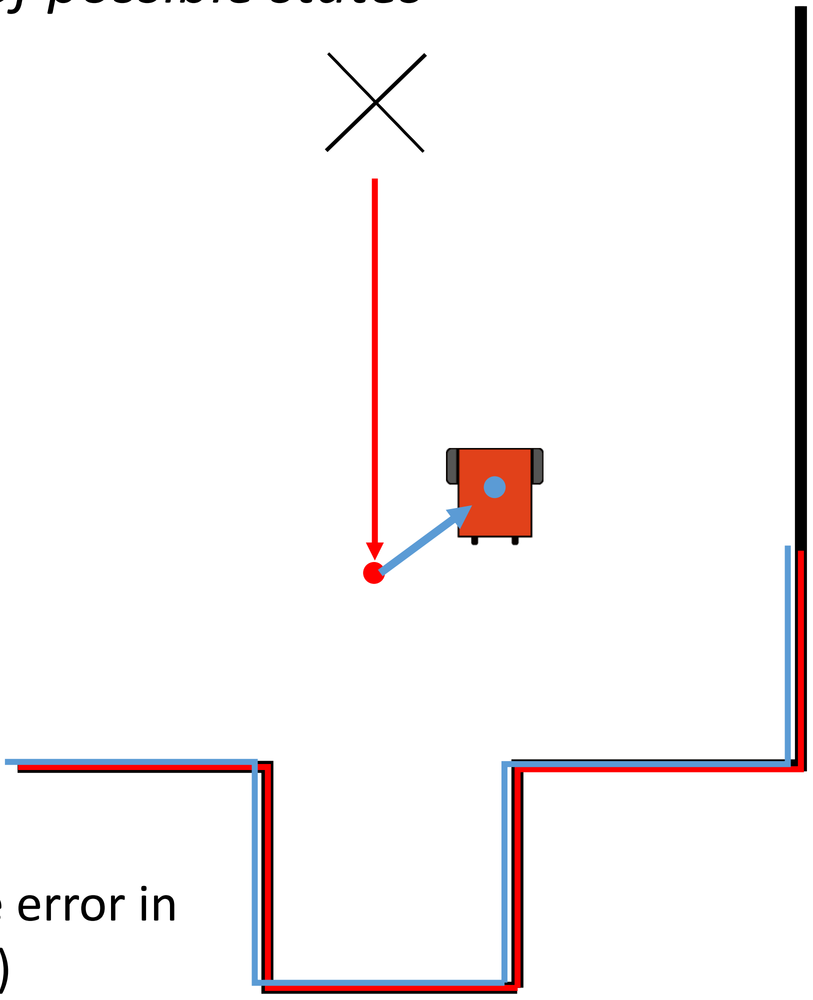
Searching with Partial Observations: an example

Initial state

I know where am I and I have a model of possible states

1. Action:
   “I want to move forward 2m”
2. Prediction
   “I am on the correct position ● ”
3. Observation prediction
   “I should see walls”
4. Perception:
   “I see this”
5. Update
   “So I should be here ● ”

We match the belief and perception to find the error in the prediction and compute the new belief (●)



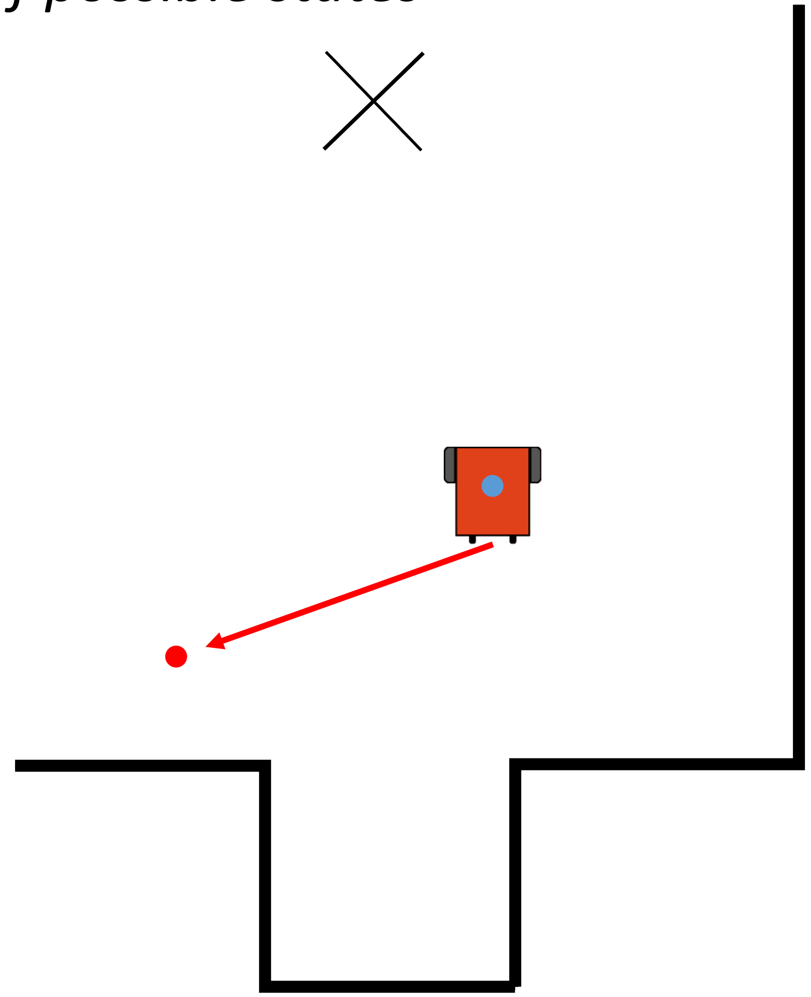
Searching with Partial Observations: an example

Initial state

I know where am I and I have a model of possible states

1. Action:
   "I want to leftwards 1m"
2. Prediction
   "I am on the correct position ● "
3. Observation prediction
   "I should see walls"
4. Perception:
   "I see this"
5. Update
   "So I should be here ● "

We start again from 1., with ● as initial belief



Searching with Partial Observations: an example

Initial state

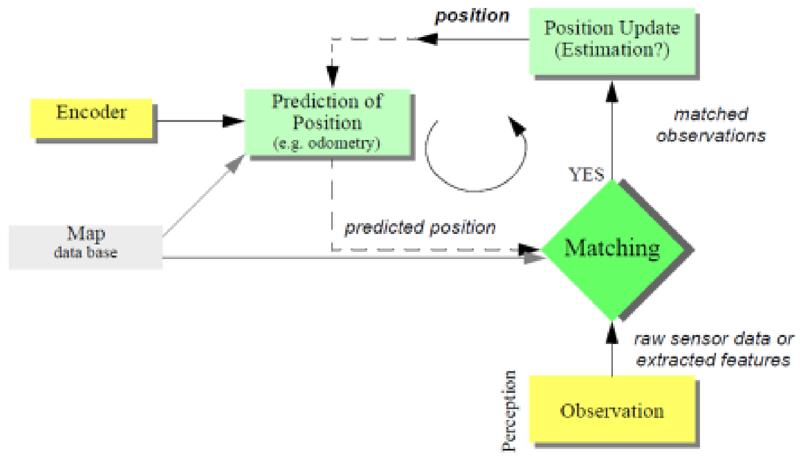
I know where an

1. Action:
   “I want to le
2. Prediction
   “I am on the
3. Observation
   “I should see
4. Perception:
   “I see this”
5. Update
   “So I should be here ● ”

We start again from 1., with ● as initial belief

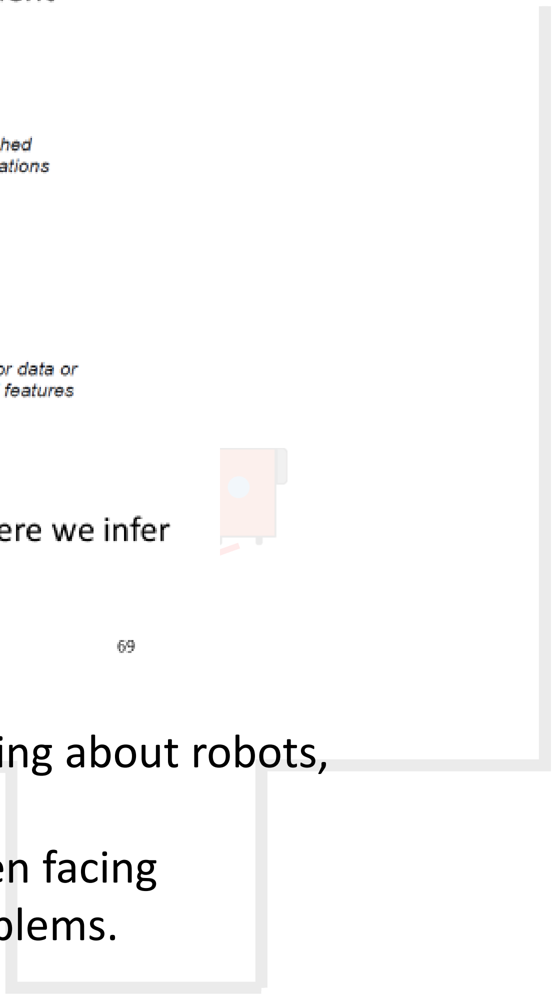
Localization

Identifying the position of the robot in a **known** environment


position
Position Update (Estimation?)
Encoder
Prediction of Position (e.g. odometry)
matched observations
YES
Map data base
predicted position
Matching
raw sensor data or extracted features
Perception
Observation

Localization is usually seen as an estimation problem, where we infer the robot position from available data.

Sistemi Intelligenti Avanzati, 2020/21 69



We have already seen a similar problem when talking about robots, as the localization problem;

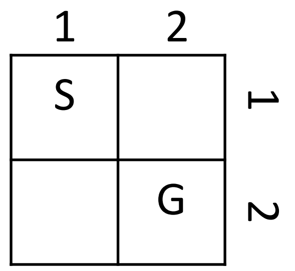
However, this mechanism is common when facing Partially-Observable and Stochastic problems.

PO-problem definition

Let’s start with a “simpler” problem:
Partially Observability but without observations (sensorless)

- A problem *P*, defined by $Actions_P$, a transition model $Result_P$, $GoalTest_P$, and $Cost_P$
- **Belief states**: if *P* has *N* states, we have up to 2^N states

ex:



$$
\begin{aligned}
b \in \{ \\
&\{(1,1)\}, \\
&\{(1,2)\}, \\
&\{(2,1)\}, \\
&\{(2,2)\}, \\
&\{(1,1),(1,2)\}, \\
&\{(1,1),(2,2)\}, \\
&\ldots, \\
&\{(1,1),(1,2),(2,1),(2,2)\}\}
\end{aligned}
$$

PO-problem definition

Let’s start with a “simpler” problem:
Partially Observability but without observations (sensorless)

- A problem *P*, defined by $Actions_P$, a transition model $Result_P$, $GoalTest_P$, and $Cost_P$
- **Belief states**: if *P* has *N* states, we have up to 2^N states
- **Initial states**: possibly more than one
- **Actions:** that can be performed from all belief states, ex:
  $b = \{s_1, s_2\},\ Actions_P(s_1) \neq Actions_P\ (s_2),$

$$Actions(b) = \bigcup_{s \in b} Actions_P(s)$$

PO-problem definitio

Let’s start with a “simpler” problem:
Partially Observability but without observations (sensorless)

- A problem *P*, defined by $Actions_P$, a transition model $Result_P$, $GoalTest_P$, and $Cost_P$
- **Belief states**: if *P* has *N* states, we have up to 2^N states
- **Initial states**: possibly more than one
- **Actions:** that can be performed from all belief states,
$$Actions(b) = \bigcup_{s \in b} Actions_P(s)$$
- **Transition model:** each possible state that I can reach from performing all actions allowed in all states in the belief *b*
$$b' = Result(b, a) = \{s' : s' \in Results_P(s, a), s \in b\}$$
$$= \bigcup_{s \in b} Results_P(s, a).$$

PO-problem definition

Let’s start with a “simpler” problem:
Partially Observability but without observations (sensorless)

- A problem *P*, defined by $Actions_P$, a transition model $Result_P$, $GoalTest_P$, and $Cost_P$
- **Belief states**: if *P* has *N* states, we have up to 2^N states
- **Initial states**: possibly more than one
- **Actions:** that can be performed from all belief states,
- **Transition model:** each possible state that I can reach from performing all actions allowed in all states in the belief *b*
- **Goal-test:** a plan that reach a belief where all states are goals; agents may reach a goal state before but they can’t know that
- **Path-Cost**

PO-problem definition – without observations

Let’s start with a “simpler” problem:
Partially Observability but without observations (sensorless)

- After we have defined a problem like this, we can apply a Search Strategy as in Observable problem
- However, we have increased (a lot!) the set of state-space and the branching factor
- The problem (likely) is no longer feasible

We have transformed a **partially-observable** problem in the *state space* into a (much bigger) **fully-observable** problem into the *belief-state space*

PO-problem definition

Partially Observability

A problem *P*, defined by $Actions_P$, a transition model $Result_P$, $GoalTest_P$, and $Cost_P$

- **Belief states**: if *P* has *N* states, we have up to 2^N states
- **Initial states**: possibly more than one
- **Actions:** that can be performed from all belief states,
- **Transition model:** each possible state that I can reach from performing all actions allowed in all states in the belief *b*
- **Goal-test:** a plan that reach a belief where all states are goals; agents may reach a goal state before but they can't know that
- **Path-Cost**

PO-problem definition

Partially Observability

A problem *P*, defined by $Actions_P,$ a transition model $Result_P,$ $GoalTest_P,$ and $Cost_P$

- **Prediction stage**: the states that the agent expects to reach doing an action *a* from its actual belief *b*

$$\hat{b} = Predict(b, a)$$

PO-problem definition

Partially Observability

A problem *P*, defined by $Actions_P,$ a transition model $Result_P,$ $GoalTest_P,$ and $Cost_P$

- **Prediction stage**: the states that the agent expects to reach doing an action *a* from its actual belief *b*

$$\hat{b} = Predict(b, a)$$

- **Observed prediction**: the set of percepts that could be observed from the predicted belief states

$$PossiblePercepts\left(\hat{b}\right) = \{o : o = Percept(s), s \in \hat{b}\}$$

PO-problem definition

Partially Observability

A problem *P*, defined by $Actions_P$, a transition model $Result_P$, $GoalTest_P$, and $Cost_P$

- **Prediction stage**: the states that the agent expects to reach doing an action *a* from its actual belief *b*

$$\hat{b} = Predict(b, a)$$

- **Observed prediction**: the set of percepts that could be observed from the predicted belief states

$$PossiblePercepts(\hat{b}) = \{o : o = Percept(s), s \in \hat{b}\}$$

- **Update**: the set of states in the predicted belief $\hat{b}$ that can generate the actual percept

$$b_o = Update(\hat{b}, o) = \{s: o = Percept(s), s \in \hat{b}\}$$

PO-problem

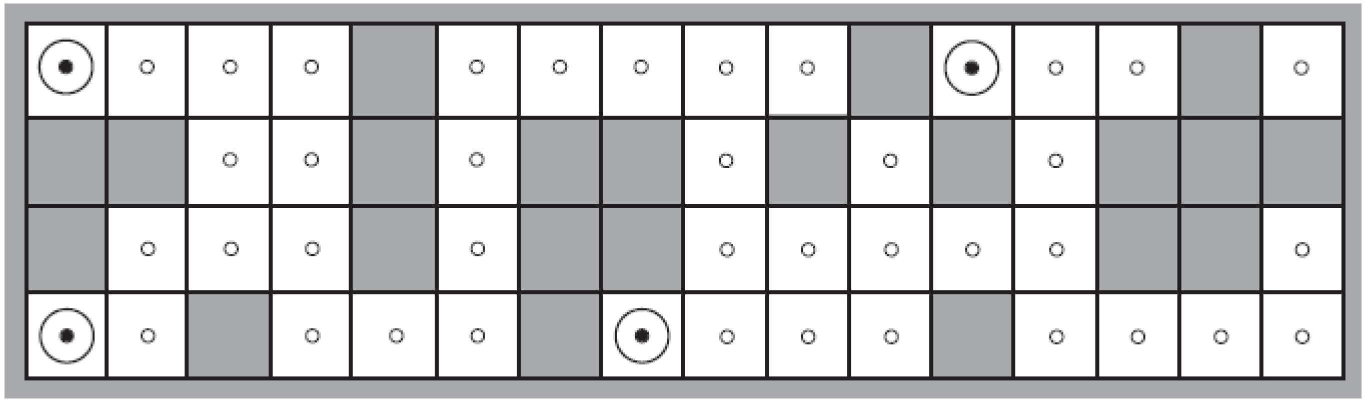
- **Update**: the set of states in the predicted belief $\hat{b}$ that can generate the actual percept
$$b_o = Update(\hat{b}, o) = \{s: o = Percept(s), s \in \hat{b}\}$$

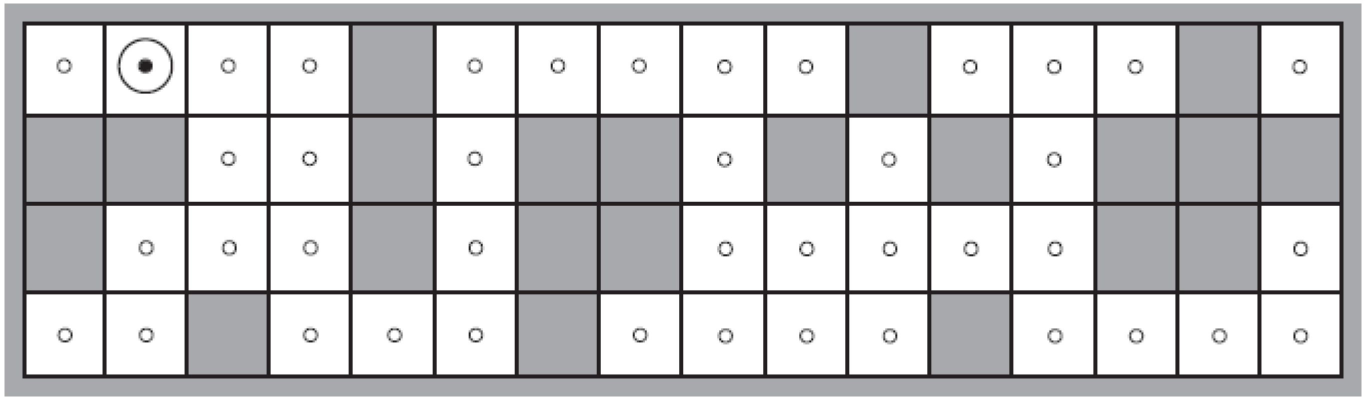
Note that: the update belief b_o **cannot be** larger than the predicted belief $\hat{b}$.

We keep multiple states in the belief b, which are used for the prediction step; the update step **reduces** the uncertainty by selecting only those state which can match the observations.

- **New belief:** putting all together, we move to the next step
$$b' = Update(Predict(b, a), o)$$

PO-problem definition: example


(a) Possible locations of robot after $E_1 = NSW$


(b) Possible locations of robot After $E_1 = NSW, E_2 = NS$

From Russel, Norvig, AI A Modern Approach 3ed, Pearson, 2010

PO-problems

- **New belief:**

$$b' = Update(Predict(b, a), o)$$

This formulation that iteratively integrates:

1. Prediction
2. Observation
3. Updates

Is used to solve different problems as **monitoring**, **filtering**, **state estimation** (where we integrate all past actions and percepts to estimate something).

We have already seen some examples as SLAM and Localization in robotics defined as Recursive-State Estimation problem and solved using probabilistic filtering techniques (EKF).

PO-problems and Stochastic Transition

We shown how we can reduce PO-problems and/or stochastic transitions to fully-observable problems.

However, search algorithms are not usually directly applicable to such problems as the number of states and the branching factor easily makes such problems unfeasible to solve.

Note that:
in fully observable problems

$$b = \{s\}$$

Sistemi Intelligenti Avanzati
Corso di Laurea in Informatica, A.A. 2020-2021
Università degli Studi di Milano

An introduction to MDPs

Matteo Luperto
Dipartimento di Informatica
matteo.luperto@unimi.it

Markov Decision Processes (MDP)

- We assume full observability of states, but non-deterministic actions
- We cannot specify a transition function like before, instead we give a set of transition probabilities

$P(s'|s,a)$ Probability of reaching state s’, given that current state is s and action a is taken

- State transitions satisfy the **Markov property**: they depend only on the current state and not on states visited before

- The Markov property can be stated more generally: the state encodes all the information we need to pick an optimal action

Example (Markovian, deterministic)

MDPs

- Can we formulate the problem asking for a plan?
  Stochastic transitions, so it would be too complex – large state space and branching factor.
- Plans are unfit for this situation: we cannot tell how to reach some goal by giving a mere sequence of actions

MDPs

- Can we formulate the problem asking for a plan?
  Stochastic transitions, so no – it would be too complex.
- Plans are unfit for this situation: we cannot tell how to reach some goal by giving a mere sequence of actions

But we have an observable state – we know the agent state.

- The Markov property allows the agent to consider only the current (known state) and not how the agent ended there

MDPs

- **Reward**:
  An agent receive, for each state, a reward $R(s)$
- **Policy:** a function that maps, for each state, the action that the agent should do

$$\pi : S \rightarrow a$$

  Given the current state, it returns what action to play (deterministic)
  An optimal policy π^* is the one that yields the highest expected utility

- **Policy Execution**:
  1. Observe current state *s*
  2. Execute action and reach next state
  3. Repeat from 1

Note that the policy is executed **online** as we cannot plan in advance

MDPs

- We previously spoken about action costs, in MDPs we speak about immediate rewards

$R_a(s, s')$ The payoff that an agent gets performing an action *a* from state *s* to state *s'* with an action *a*

- Rewards can be thought as a generalization of what before we described by means of goal states specification

- Solving the MDP means finding a policy that maximizes the expected reward over some finite or infinite time horizon; such policy is called the optimal policy (π^*)

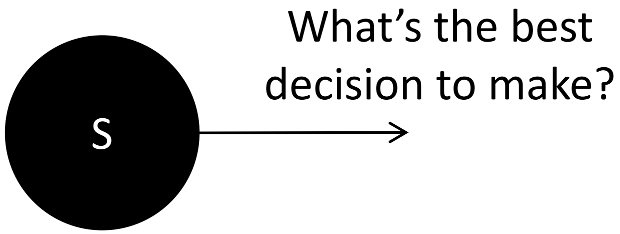
Some features of policies

- **Deterministic**: given a state it does not randomize on which action to take
- **Stationary** (or memoryless): it does not change over time

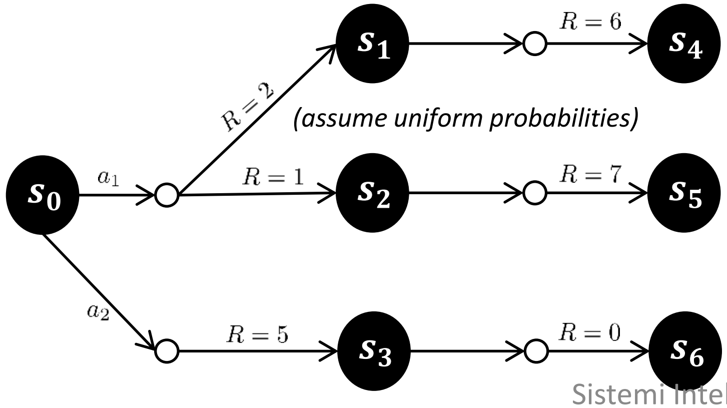
In MDPs, up to some reasonable properties or common operative choices, these assumptions are not restrictive: there always exists an optimal policy that is deterministic and stationary

Stationary optimal policies

- In infinite horizon MDPs, the optimal policy is stationary (or memoryless):



- The Markov property says that all the information (transitions, rewards, ...) we need to decide is encoded in the state. Thus, the answer to the above question only depends on the state, not on the time of visit. *Does this hold in finite horizon settings?*
- *H = Horizon*, how many action the agent can perform
- Is this true in finite horizon MDPs?



$\pi(s_0)$	$H=1$	$H=2$
a_1	1.5	π^*8
a_2	π^*5	5

The optimal policy depends on the time of visit!

MDP Value iteration

- Let's introduce the concept of **value function**

- How does it work?

$$V_\pi : X \to \mathbb{R}$$

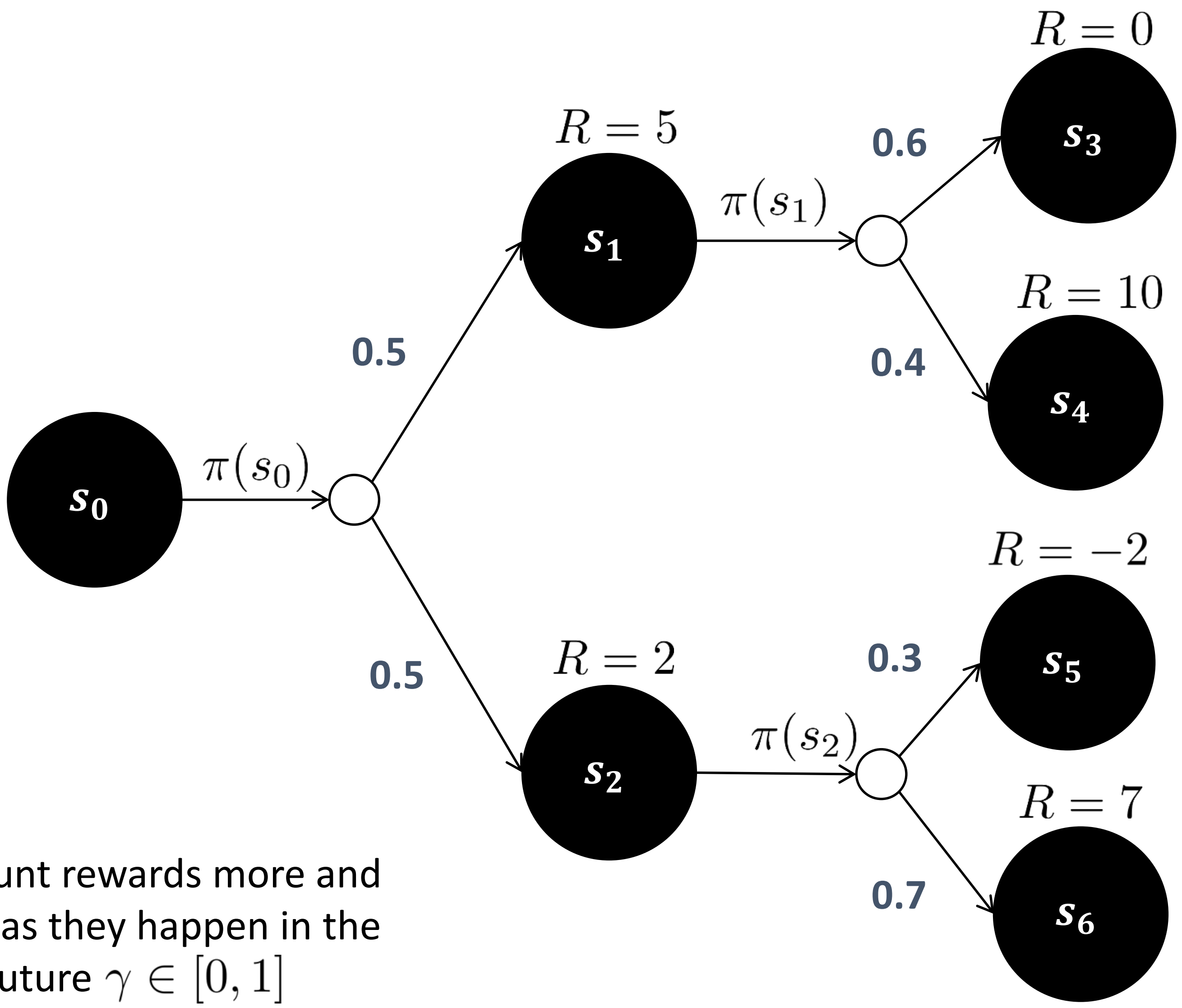
An agent executing policy π is in state s: how happy is the agent?

$$\Downarrow$$

$$V_\pi(s)$$

- This quantity is defined as **the expected cumulative reward that can be obtained by executing π from s**

Example with H=2



Discount rewards more and more as they happen in the long future $\gamma \in [0, 1]$

$$V_\pi(s_0) = 0.5(\gamma^0 5 + (0.4(\gamma^1 10))) + 0.5(\gamma^0 2 + (0.3\gamma^1(-2) + 0.7\gamma^1(7)))$$

MDP Value iteration (intuition)

$V^*_{\pi,k}(s)$ Expected value of the optimal policy from state s when H=k

$V^*_{\pi,0}(s) = 0 \quad \forall s \in X$ The horizon is zero, no action no reward

The horizon is 1, there's room for just one action. The best thing to do is selecting the action that maximizes the **immediate expected reward**

$$V^*_{\pi,1}(s) = \max_a \left\{ \sum_{s' \in X} P(s'|s,a) R_a(s,s') \right\} \forall s \in X$$

Now the horizon is 2. The optimal policy would select the action that maximizes the immediate expected reward plus the **expected discounted reward of acting optimally from the arrival state**

$$V^*_{\pi,2}(s) = \max_a \left\{ \sum_{s' \in X} P(s'|s,a) \big(R_a(s,s') + \gamma V^*_{\pi,1}(s') \big) \right\} \forall s \in X$$

$$V^*_{\pi,3}(s) = \max_a \left\{ \sum_{s' \in X} P(s'|s,a) \big(R_a(s,s') + \gamma V^*_{\pi,2}(s') \big) \right\} \forall s \in X$$

$$\vdots$$

MDP Value iteration

The utility of a state is the immediate reward for that state plus the expected discounted utility of the next state, assuming that the agent chooses the optimal action.

- We obtain a recursive definition:

$$V^*_{\pi,H}(s) = \max_a \left\{ \sum_{s' \in X} P(s'|s,a)\big(R_a(s,s') + \gamma V^*_{\pi,H-1}(s')\big) \right\}$$

$$\pi^*(s) = \arg\max_a \left\{ \sum_{s' \in X} P(s'|s,a)\big(R_a(s,s') + \gamma V^*_{\pi,H-1}(s')\big) \right\}$$

- In infinite horizon settings the above definition becomes the *Bellman Equation*

$$V^*_{\pi}(s) = \max_a \left\{ \sum_{s' \in X} P(s'|s,a)\big(R_a(s,s') + \gamma V^*_{\pi}(s')\big) \right\}$$

- Solved with iterative methods
- Encodes the Bellman’s principle of optimality

Partially-Observable MPD

- We can extend MDP to Partially Observable Problems
- PO-MDP: Partially-Observable Markov Decision Process
- As we have seen before, we need to define a policy for each **belief state *b***
- The problem is much more difficult to solve, we can adapt Value Iteration or try different techniques to find the policy (e.g., Reinforcement Learning)

Sistemi Intelligenti Avanzati
Corso di Laurea in Informatica, A.A. 2020-2021
Università degli Studi di Milano

Online Search

Matteo Luperto
Dipartimento di Informatica
matteo.luperto@unimi.it

Search Algorithms

However, we have made several assumption

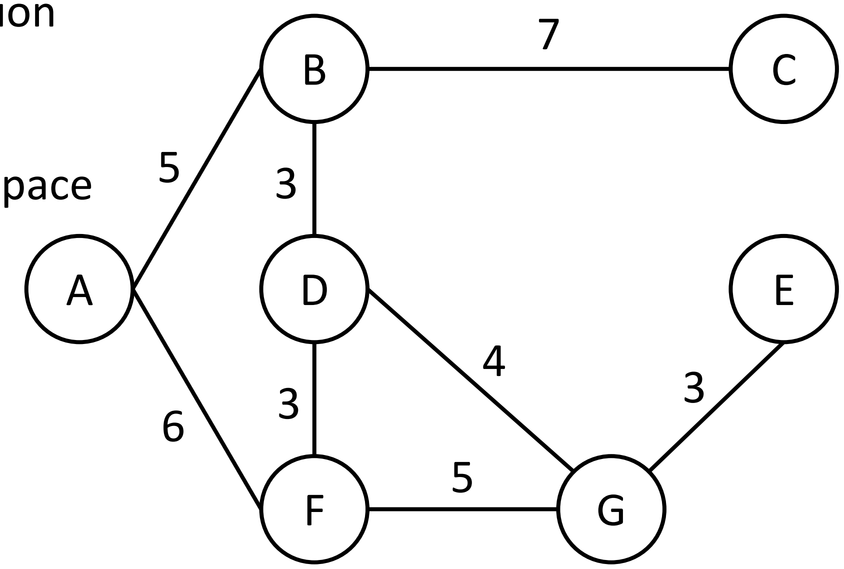
1. Discrete and finite search space
   vs Continuous and/or infinite search space

2. Deterministic transitions
   vs Stochastic transitions

3. Observable States
   vs partially-observable states

4. Known Search Space
   vs <u>unknown search space</u>

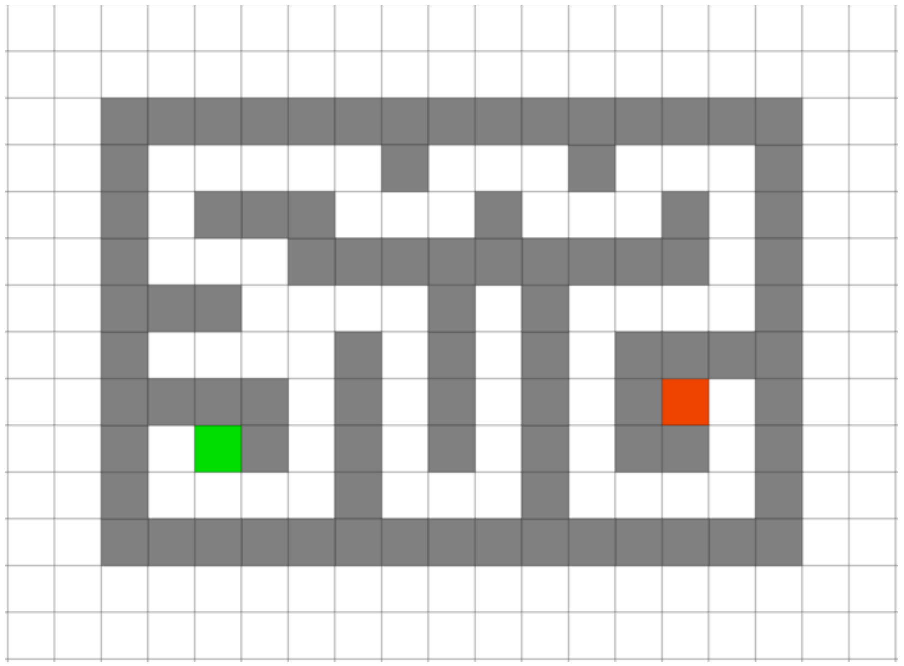
5. Offline search
   vs <u>online search</u>

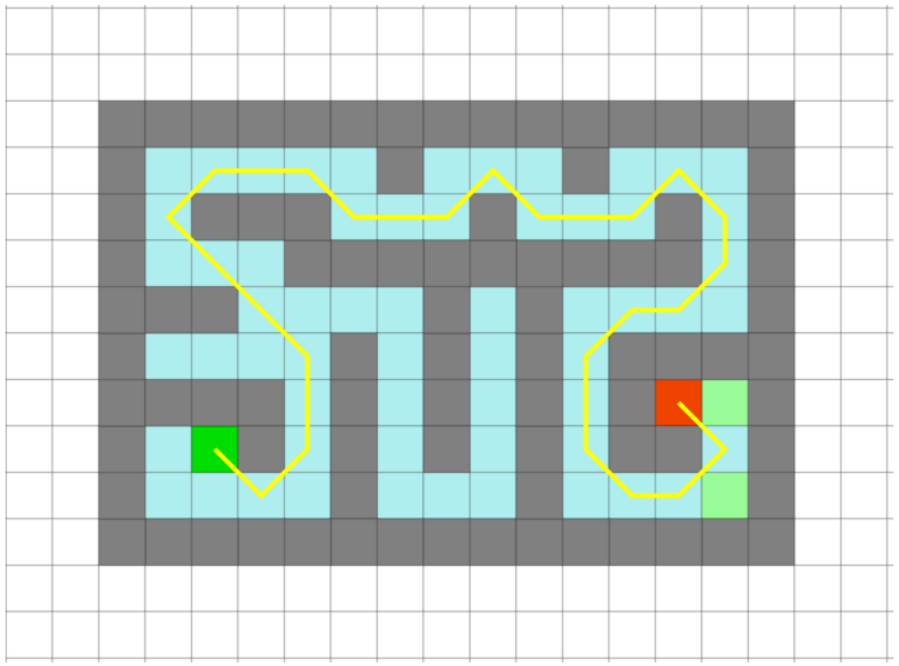


(some of) these assumptions are not admissible for several problems

Online Search Problems

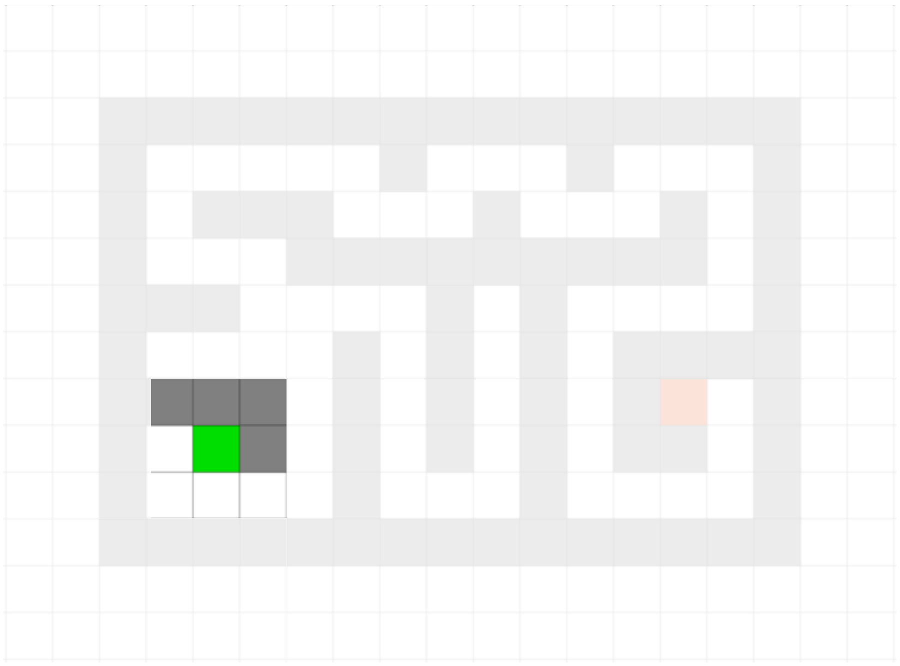
- Almost all problems that we are discussed so far could be solved *offline.*
- State space is known and thus the agent can *simulate* the execution of all actions *before* acting

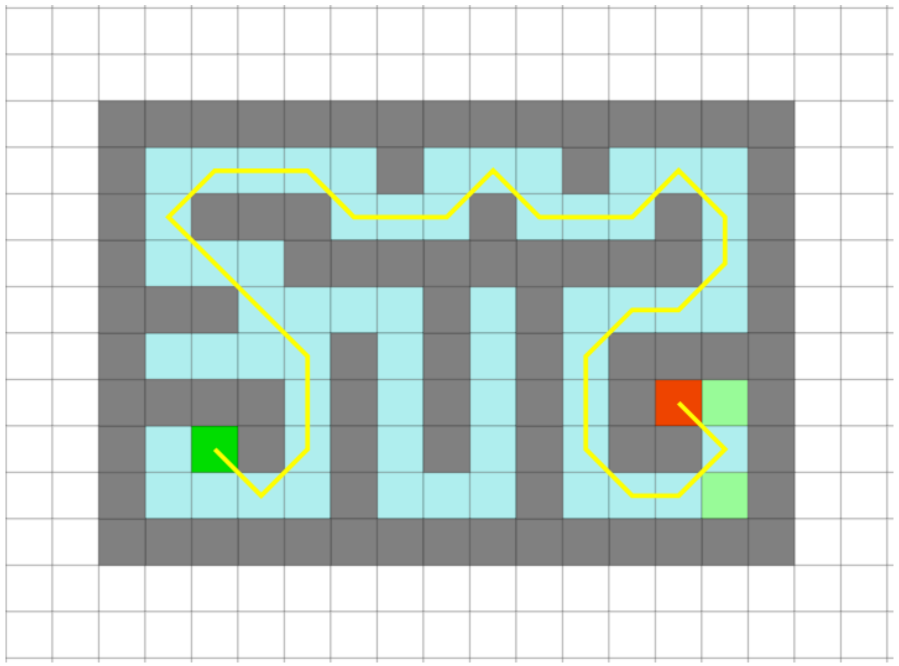




Online Search Problems

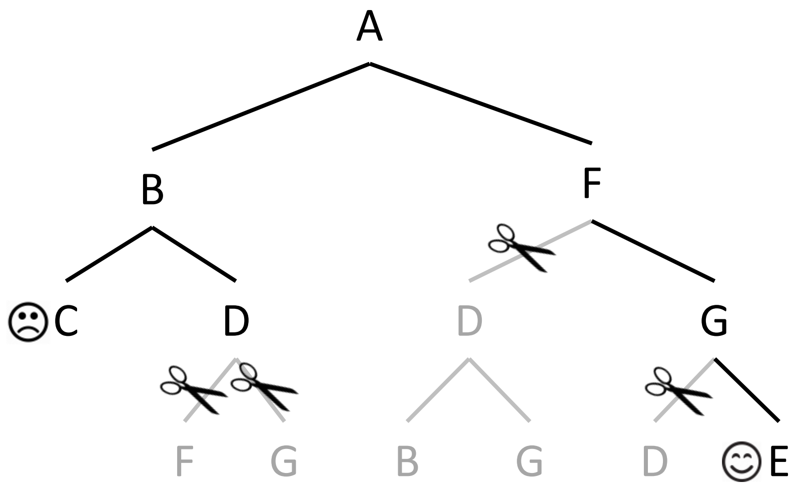
- Online search is *necessary* for unknown environment
- Agent does not know what state exists or what action can do
- **Exploration problem:** agents uses actions as experiments to *learn* enough to make decisions

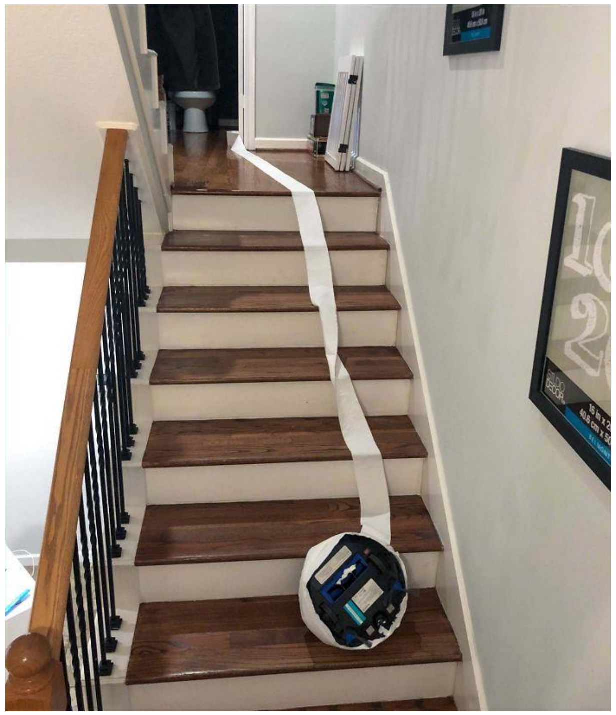




Online Search

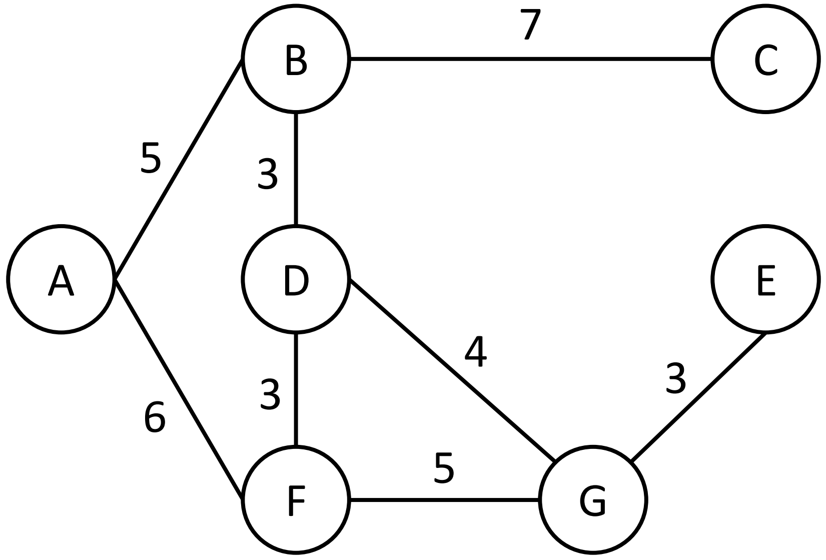
- **Competitive ratio:** ratio between the cost of the path of the agent and the optimal path (computed in the known environment)
- **Dead-ends:** if the agent does not know the states it is possible to end in an irreversible state, that the agent does want to avoid

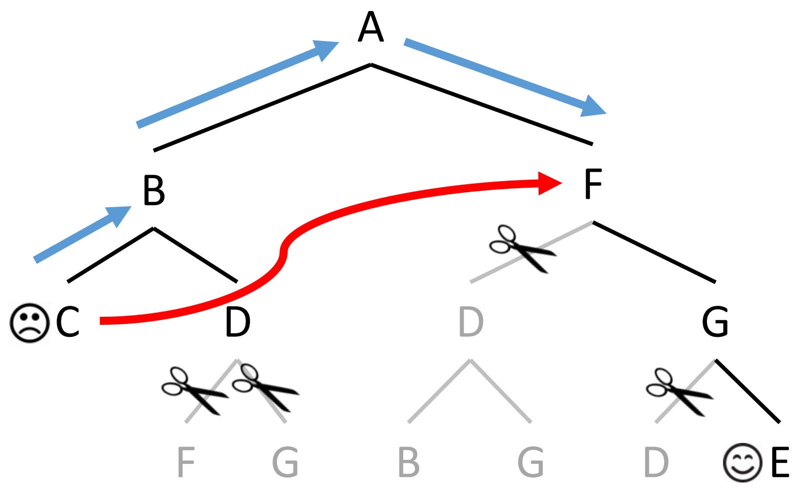




Online vs Offline Search

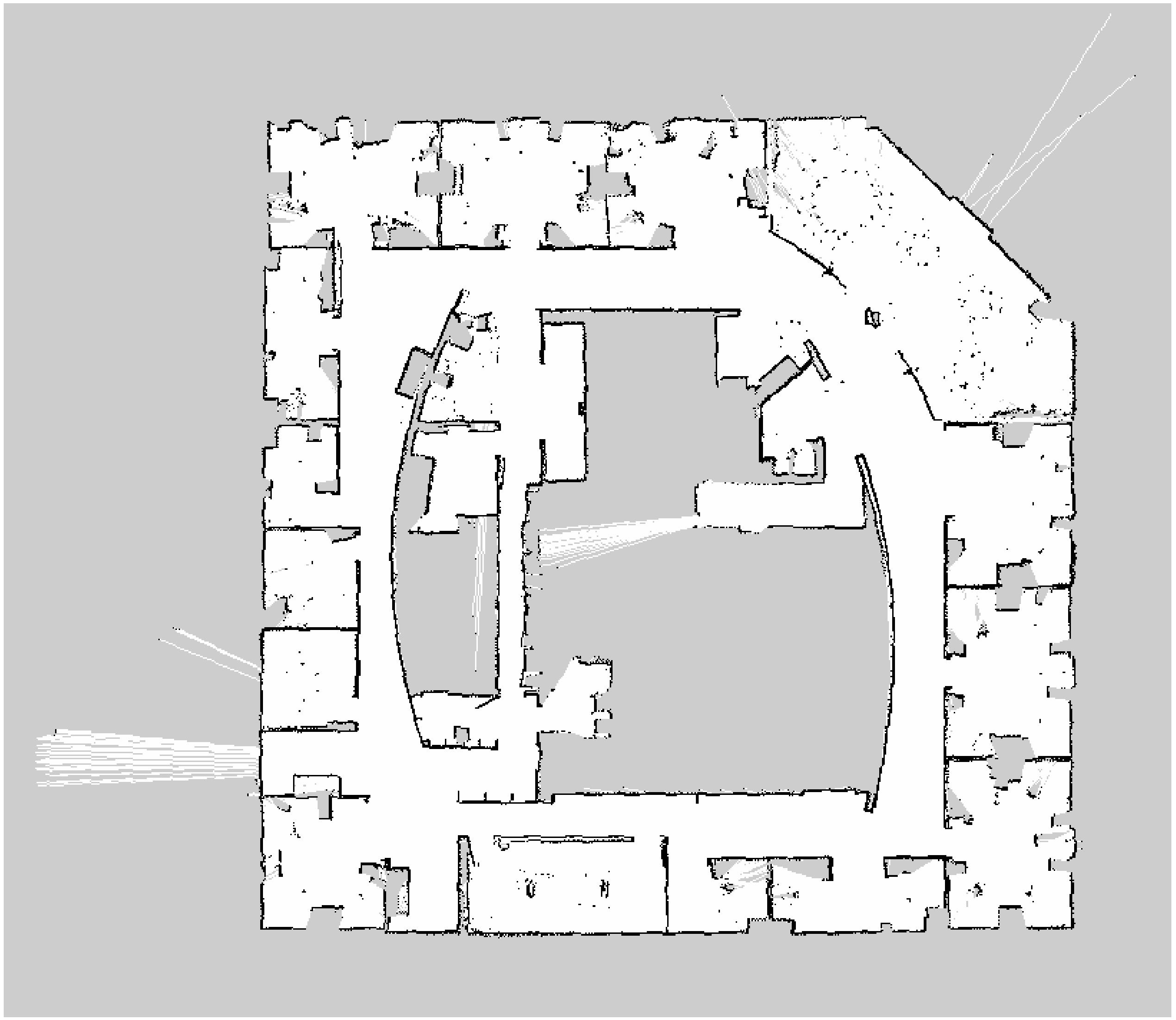
- **Offline Search:** agents simulates the problem in advance, as the state space is known; if the agent ends in a dead-end it can "jump" to a new far-away state
- Online Search: agents perform actions as states are discovery by visiting them; transition to a new state can be costly, so **local search strategies** are preferred (e.g., DFS)



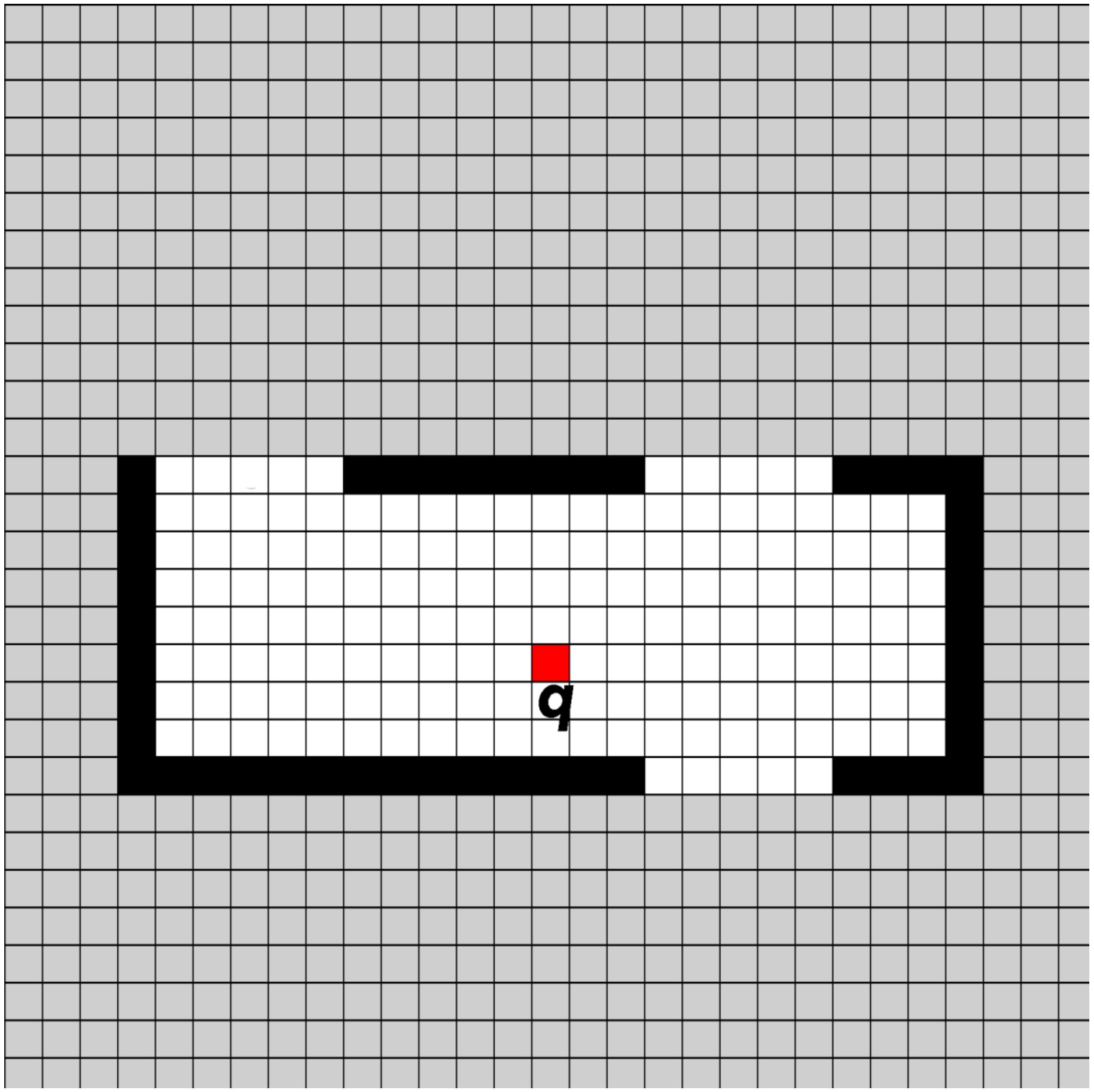


Online Search Example: Robot Exploration

- The robot is placed in a new unknown environment
- GOAL: map the entire environment as fast as possible

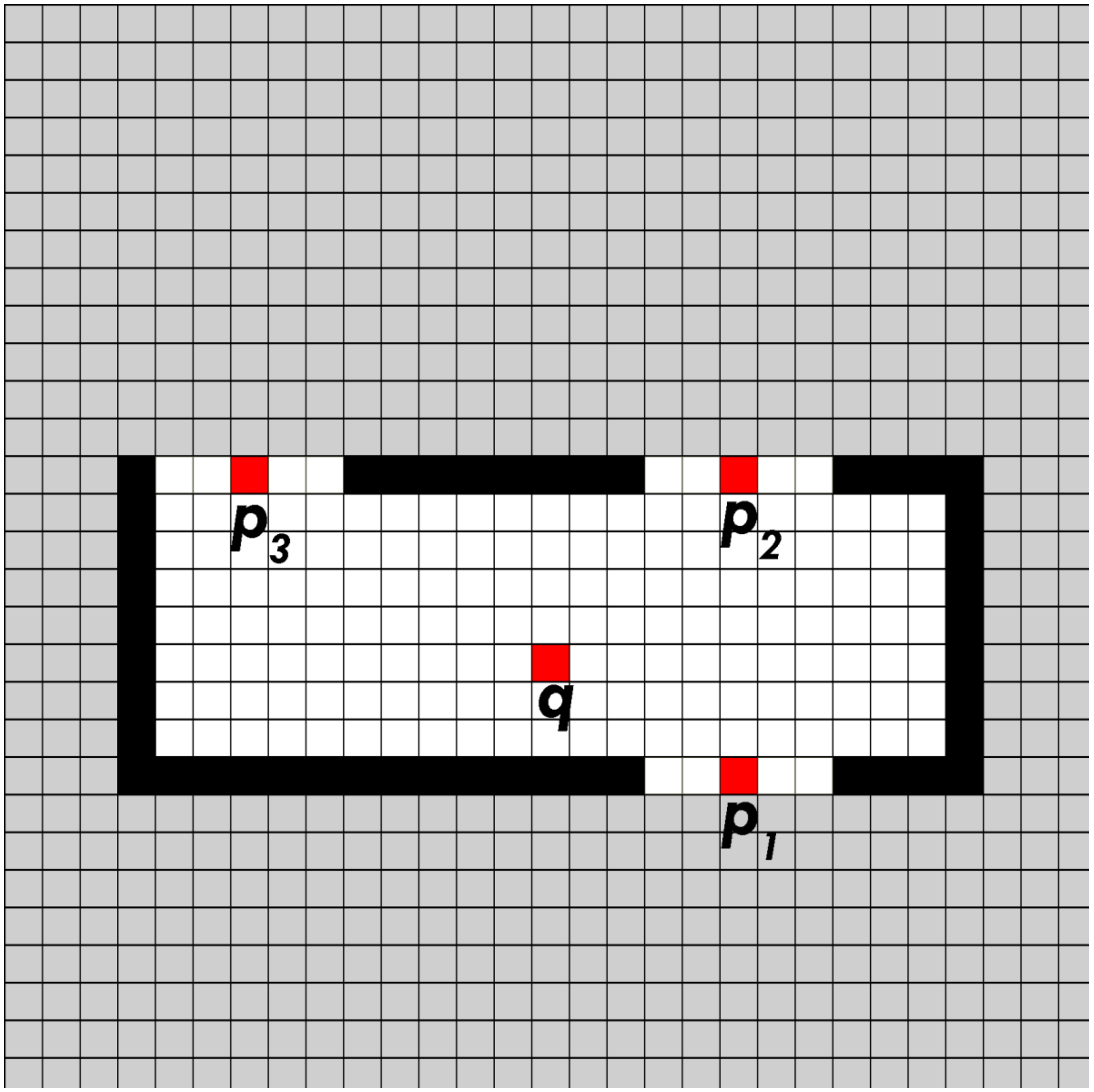


Online Exploration

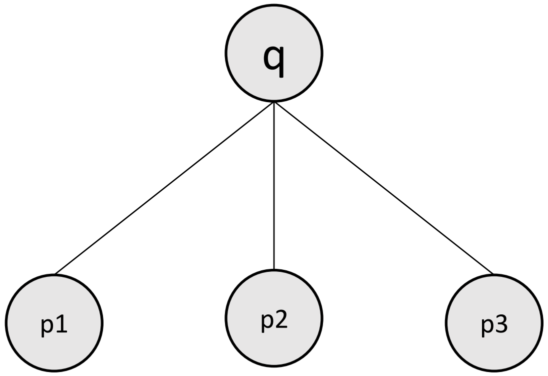


Start from the initial map of the environment (percepts from the starting position)

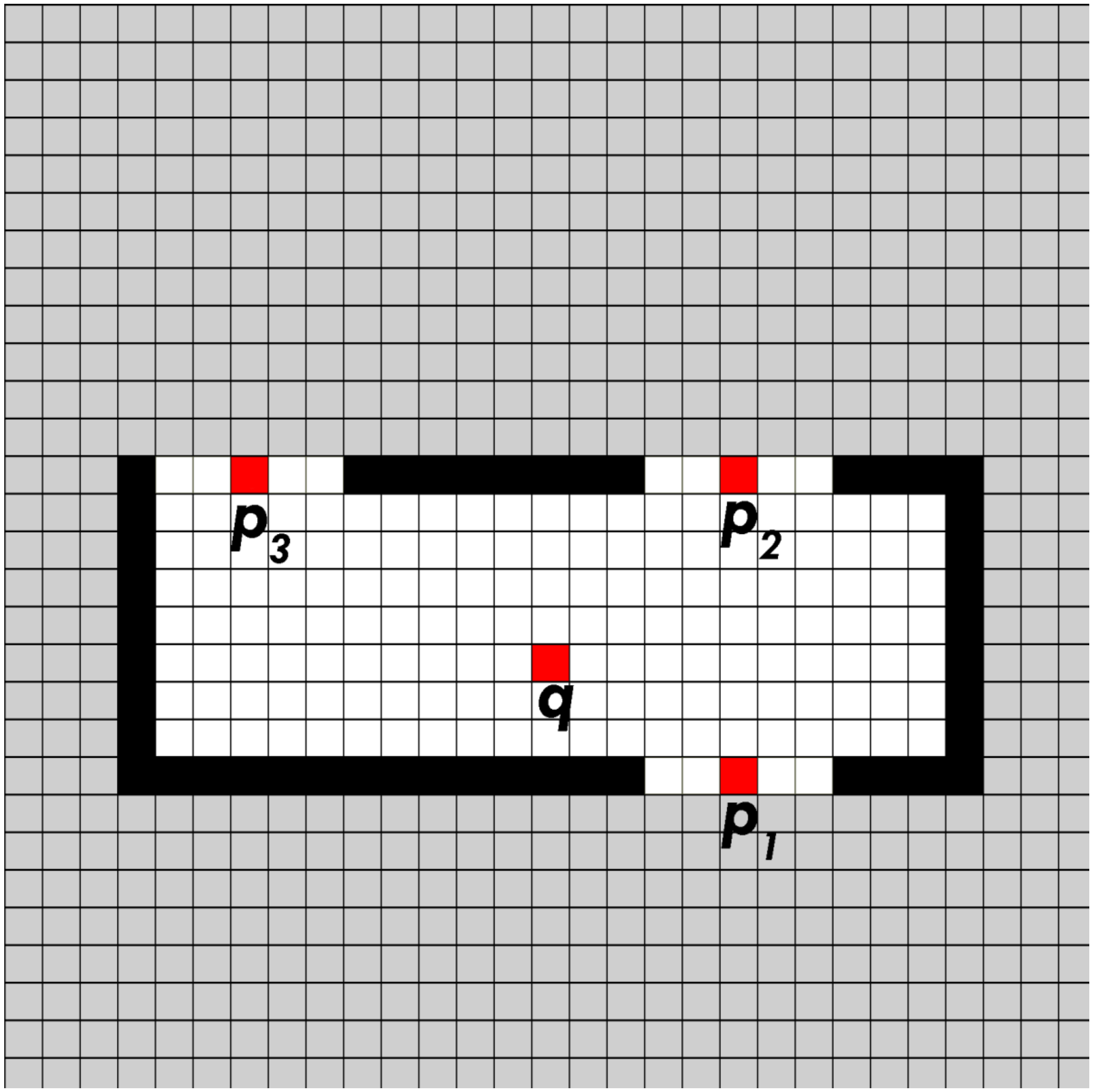
Online Exploration



Identify the set of possible states that the agent can reach from its position *q*

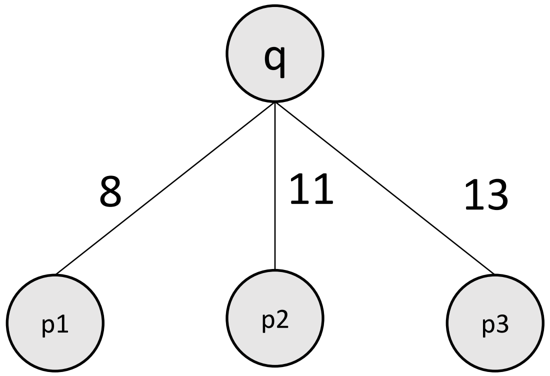


Online Exploration

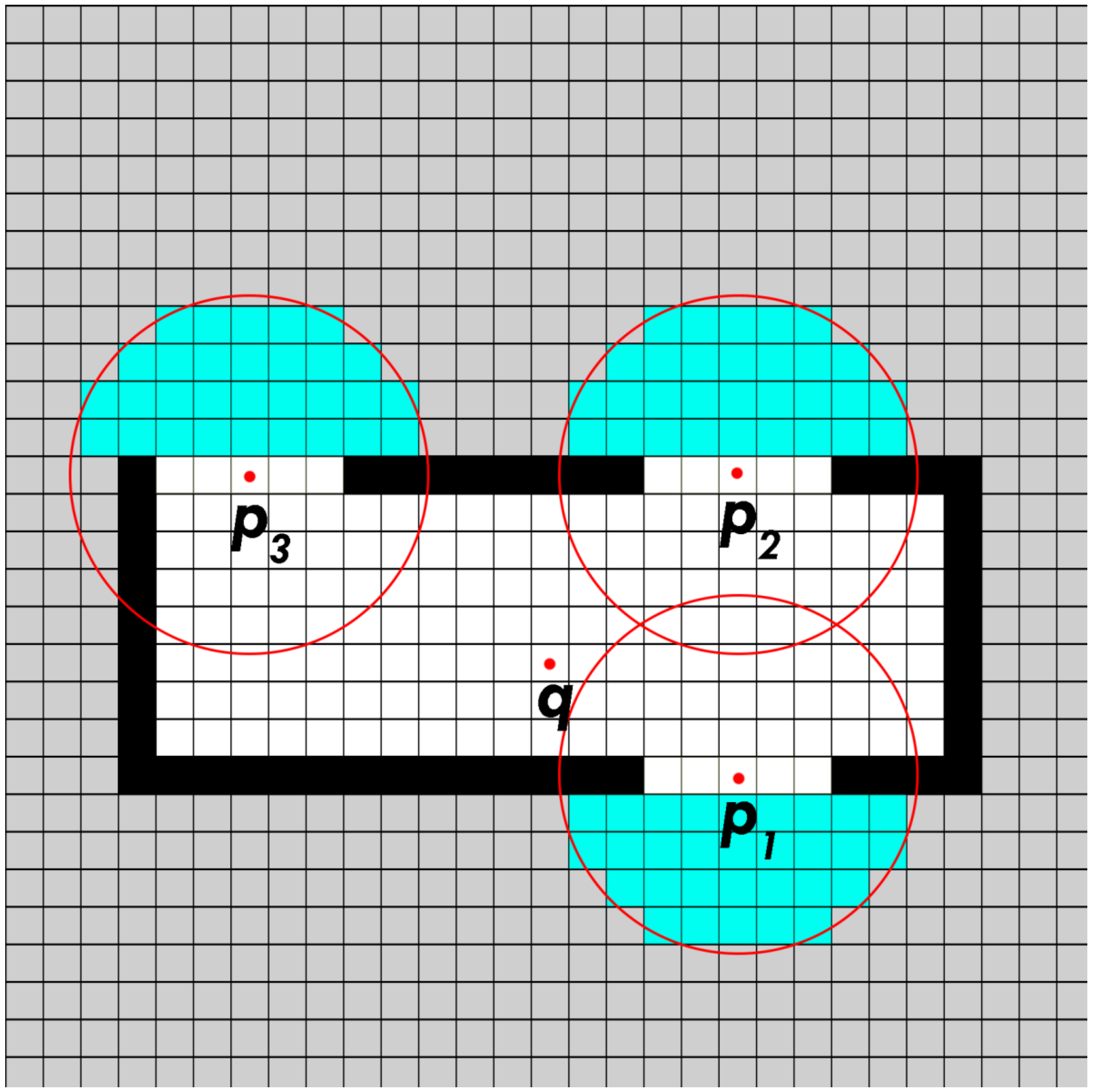


Compute the costs for each action

(e.g. Manhattan distance)



Online Exploration

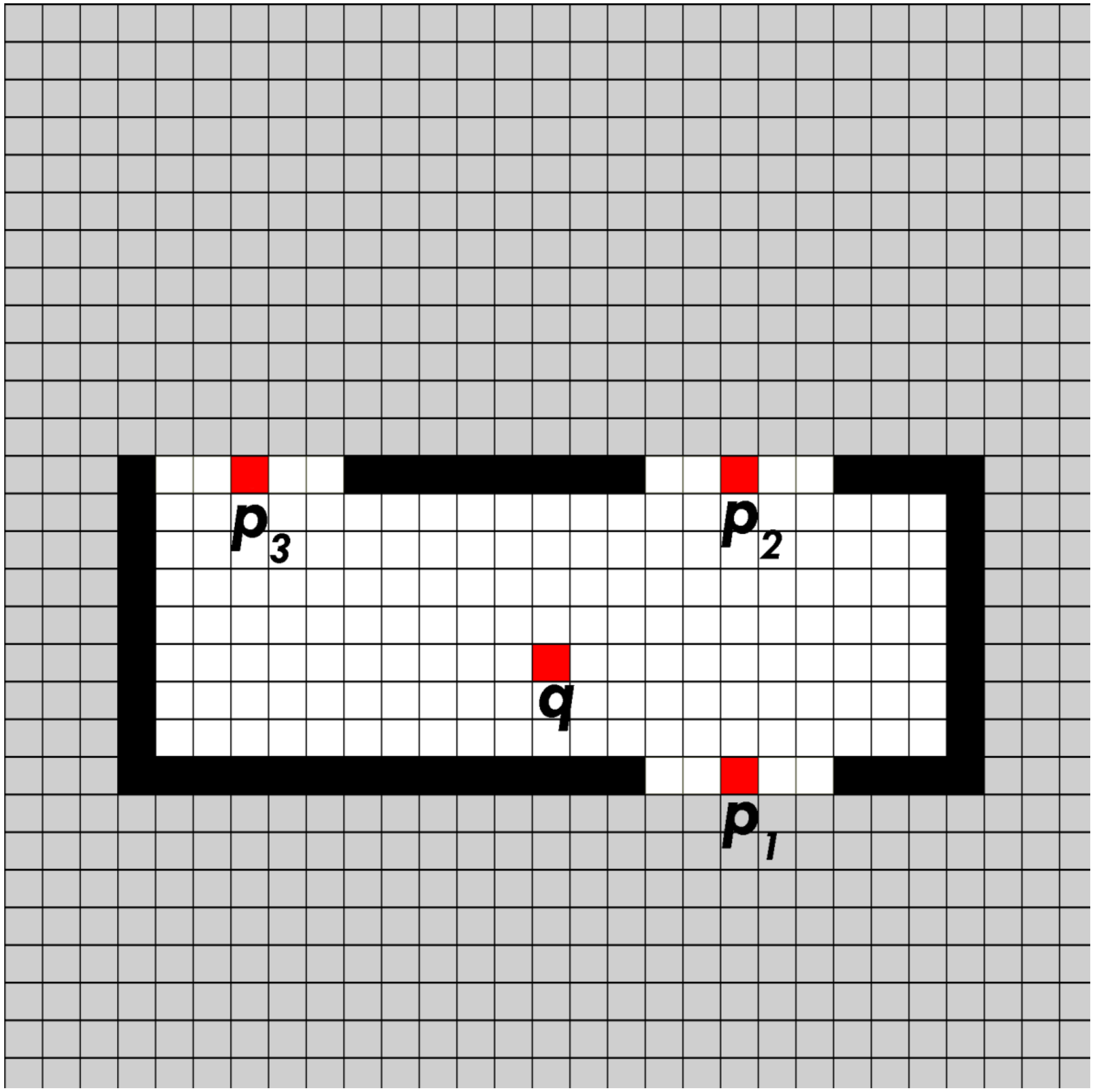


Compute an heuristic function *h*

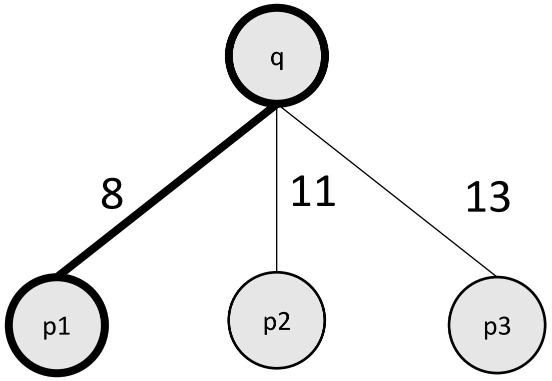
(e.g. estimated 2D lidar beam with full-range)

$$\begin{tikzcd}
 & q \arrow[dl, "8"'] \arrow[d, "11"] \arrow[dr, "13"] & \\
p1 & p2 & p3
\end{tikzcd}$$

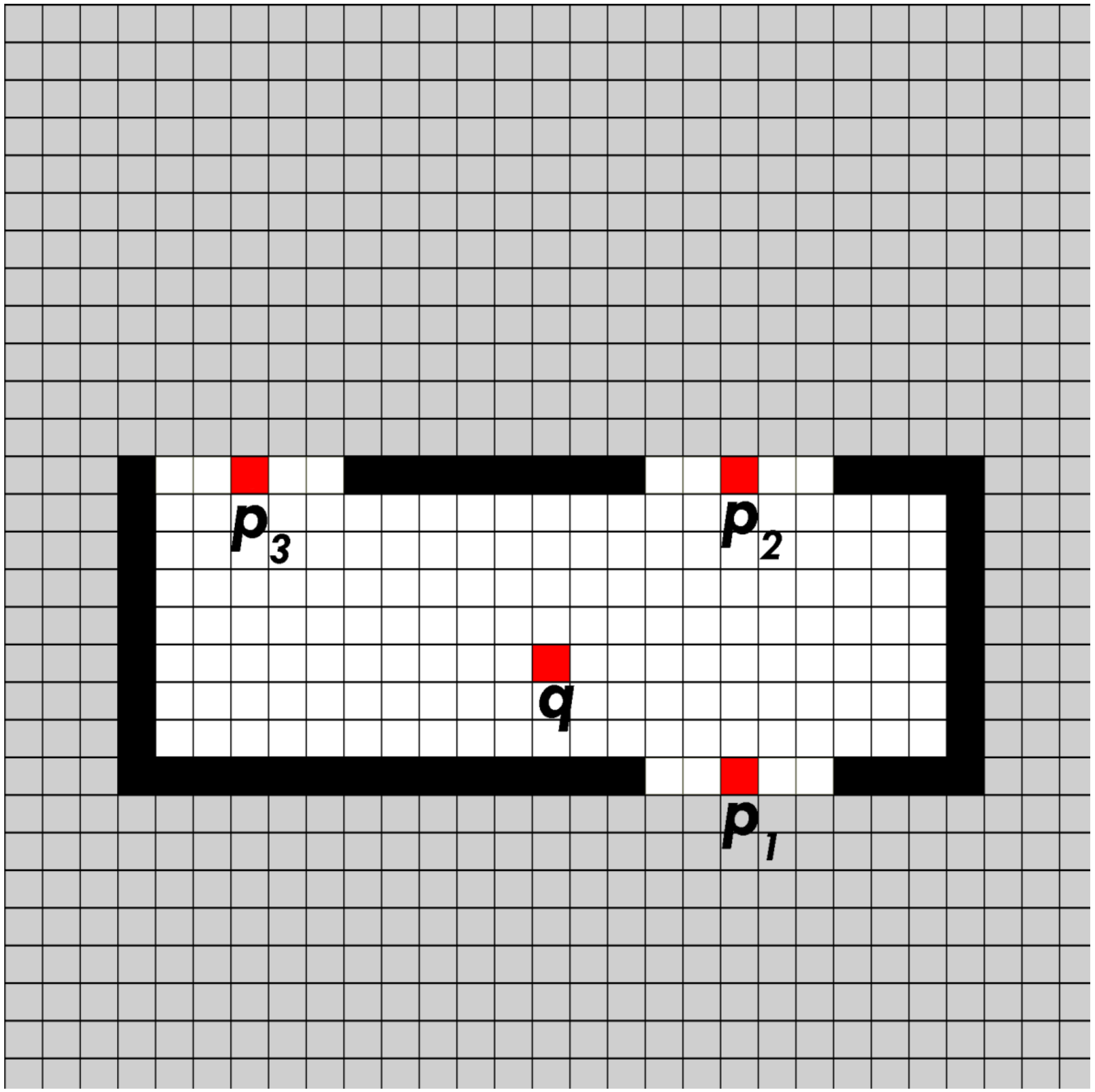
Online Exploration



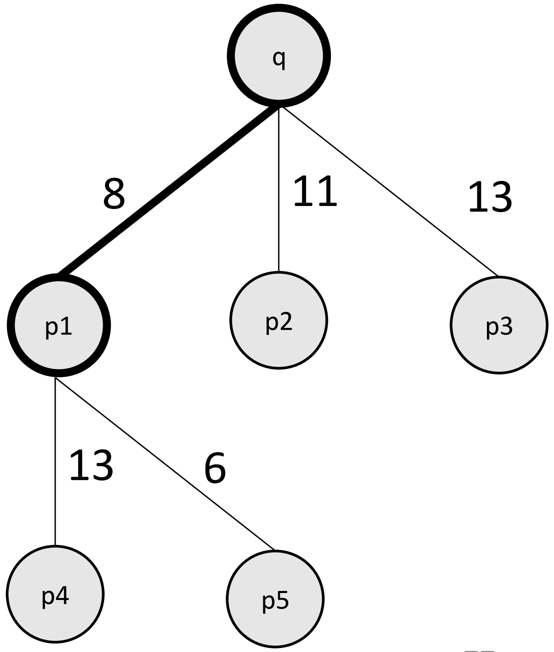
Choose the most promising state according to a Search Strategy (e.g. Depth First Search)



Online Exploration



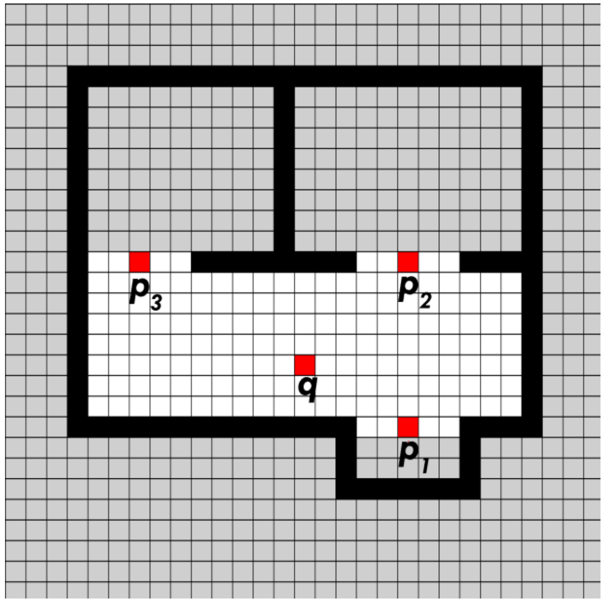
1. reach the location
2. update the search state with new knowledge,
3. repeat



Online Exploration

We solved exploration modeling it as an abstract search problem:

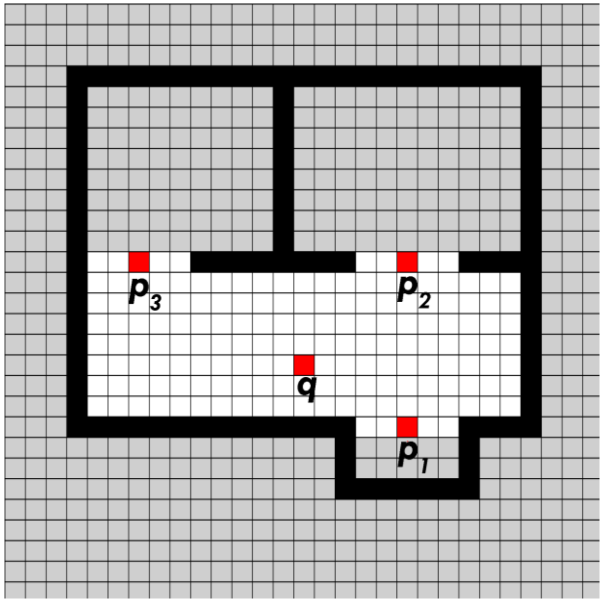
1. Identify all reachable states from current location
    1. Localization
2. Compute Costs and Heuristics
3. Select most promising state according to a Search Strategy
4. Go there
    1. Localization
    2. Path Planning
        1. Global Plan
        2. Local Plan
5. Integrate new knowledge in the map
    1. SLAM
6. Repeat from (1)



Online Exploration

Some sub-problems are solved at the simultaneously

1. Identify all reachable states from current location
    1. Localization
2. Compute Costs and Heuristics
3. Select most promising state according to a Search Strategy
4. Go there
    1. **Localization**
    2. **Path Planning**
        1. Global Plan
        2. Local Plan
5. Integrate new knowledge in the map
    1. **SLAM** – (mapping)
6. Repeat from (1)

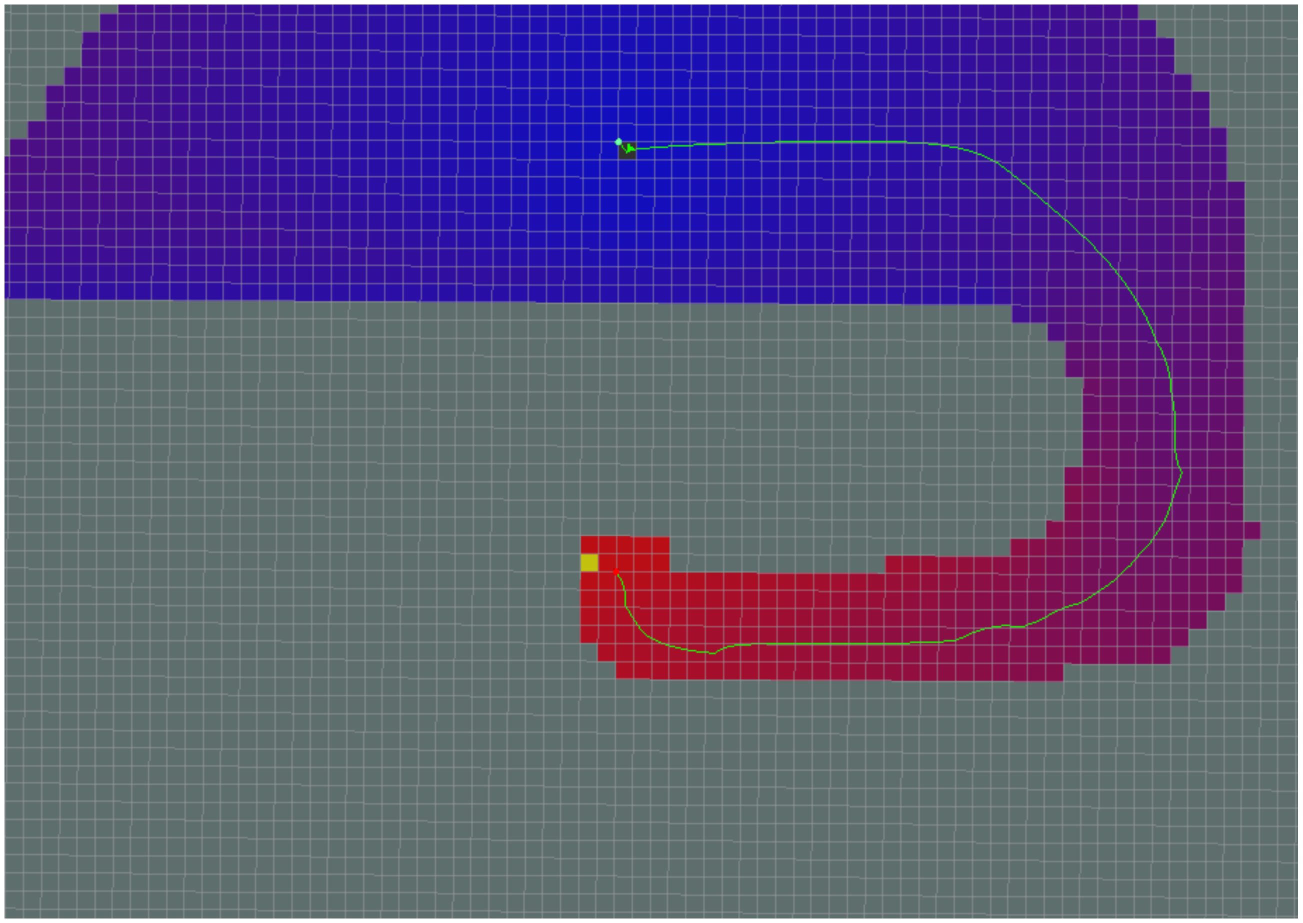


Online Exploration and Path Planning

- Some sub-problems are solved as Search problems as well (and involving other sub-subproblems)
- **Path Planning**
  1. **Global Plan**
  2. Local Plan

Global path planning is modeled as a fully-observable offline deterministic search problem

(here solved with A*)

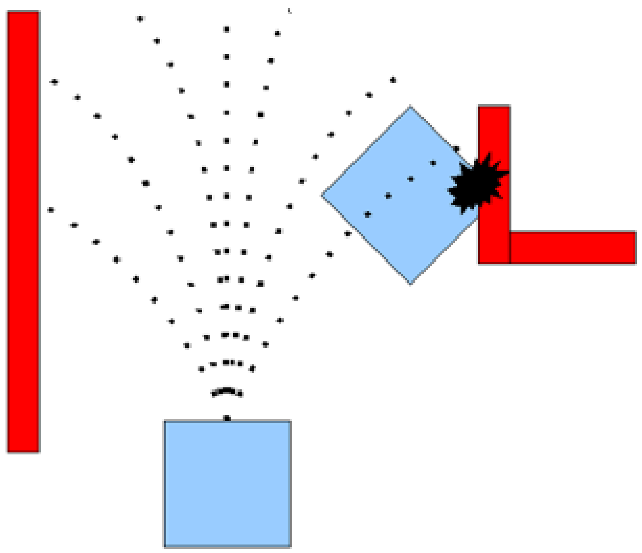


Online Exploration and Path Planning

- Some sub-problems are solved as Search problems as well (and involving other sub-subproblems)
- **Path Planning**
    1. Global Plan
    2. **Local Plan**

DWA Local planner

The local planner package provides a controller that drives a mobile base in the plane and connect the path planner to the robot.
Using a map, the planner creates a kinematic trajectory for the robot to get from a start to a goal location. Along the way, the planner creates, at least locally around the robot, a value function, represented as a grid map. This value function encodes the costs of traversing through the grid cells. The controller's job is to use this value function to determine dx,dy,dtheta velocities to send to the robot.



Example

Autonomous Exploration of an indoor floor
Using a P3AT robot and a 2D Lidar
EXQUISITUS, Center for E-City
School of EEE
Nanyang Technological University

References

- Russel S., Norvig P., Artificial Intelligence, a Modern Approach, III ED