Sistemi Intelligenti
Corso di Laurea in Informatica, A.A. 2017-2018
Università degli Studi di Milano

Discrete planning (an introduction)

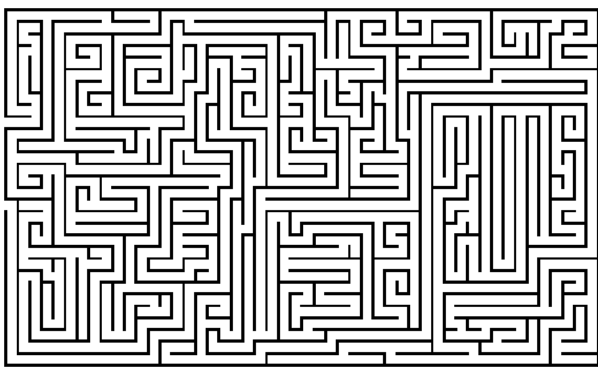
Nicola Basilico
Dipartimento di Informatica
Via Comelico 39/41 - 20135 Milano (MI)
Ufficio S242
nicola.basilico@unimi.it
+39 02.503.16294

Sito per queste lezioni

Flavors in discrete planning

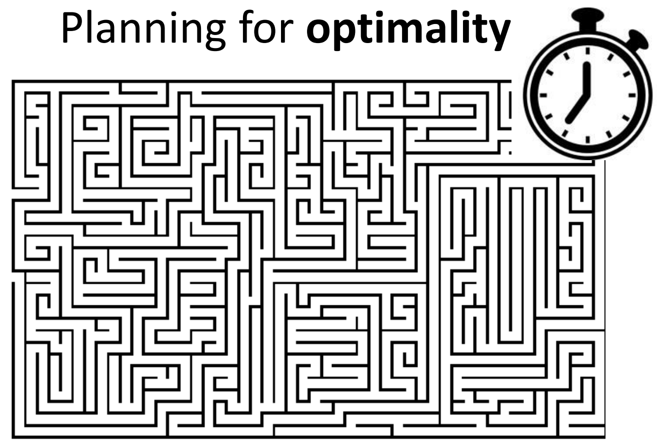
Simplest class of planning problems: no uncertainties, finite or countable state space

Planning for **feasibility**



is there an exit?
(problem solving)

Planning for **optimality**



what’s the shortest
route to exit?

Formulation

- Single agent, multiple agents (doesn’t matter at this stage)

Formulation

- Single agent, multiple agents (doesn’t matter at this stage)
- X is the set of states, x is a generic state, abstraction

Formulation

- Single agent, multiple agents (doesn’t matter at this stage)
- X is the set of states, x is a generic state, abstraction
- U(x) is the set of actions that can be undertaken in state x, u is a generic action

Formulation

- Single agent, multiple agents (doesn't matter at this stage)
- X is the set of states, x is a generic state, abstraction
- U(x) is the set of actions that can be undertaken in state x, u is a generic action
- If action u is taken in state x then state x'=f(x,u) is reached, f is the transition **function** (deterministic)

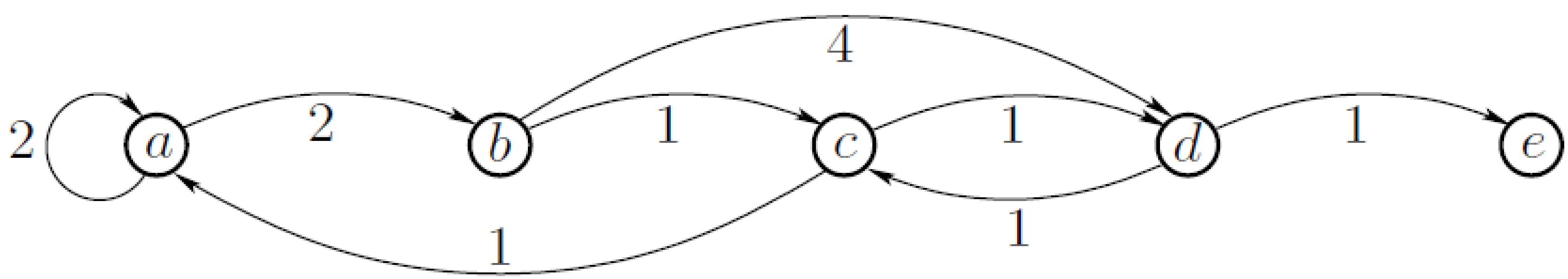
Formulation

- Single agent, multiple agents (doesn’t matter at this stage)
- X is the set of states, x is a generic state, abstraction
- $U(x)$ is the set of actions that can be undertaken in state x, u is a generic action
- If action u is taken in state x then state $x'=f(x,u)$ is reached, f is the transition **function** (deterministic)
- There is a initial state x_i and a set of goal states X_G

Formulation

- Single agent, multiple agents (doesn’t matter at this stage)
- X is the set of states, x is a generic state, abstraction
- U(x) is the set of actions that can be undertaken in state x, u is a generic action
- If action u is taken in state x then state x’=f(x,u) is reached, f is the transition **function** (deterministic)
- There is a initial state x_i and a set of goal states X_G

- Equivalent representation: state transition graph G=(V,E)



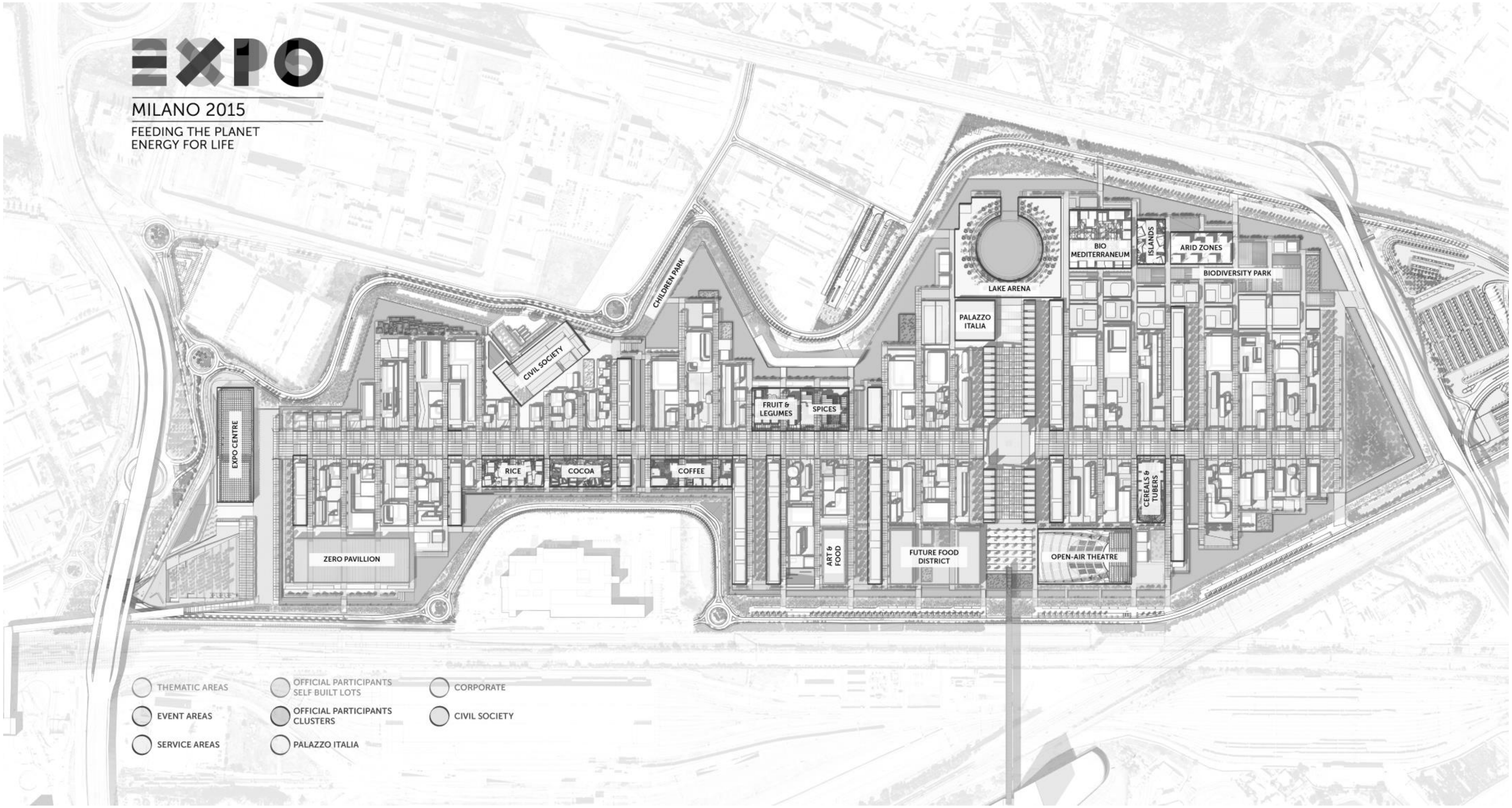
we will use “state” and “node” as interchangeable terms

Example

Consider a mobile robot moving on a graph-represented environment:

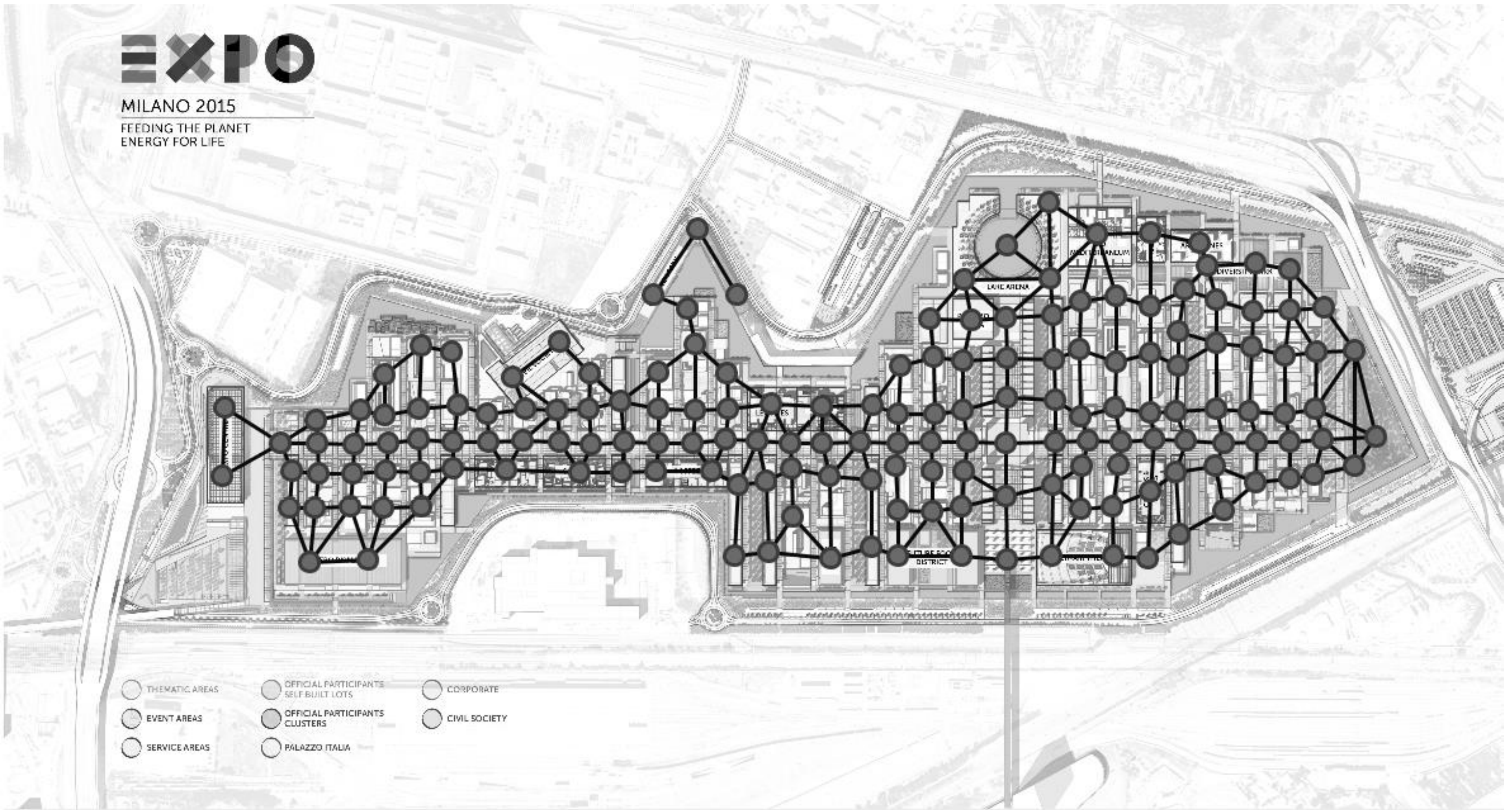
- **States**: nodes of the graph, they represent physical locations
- **Edges**: represent connections between nearby locations or, equivalently, movement actions
- **Initial state**: some starting location for the robot
- **Goal state(s)**: some location(s) to reach, e.g., recharging station, parking depot…

Example

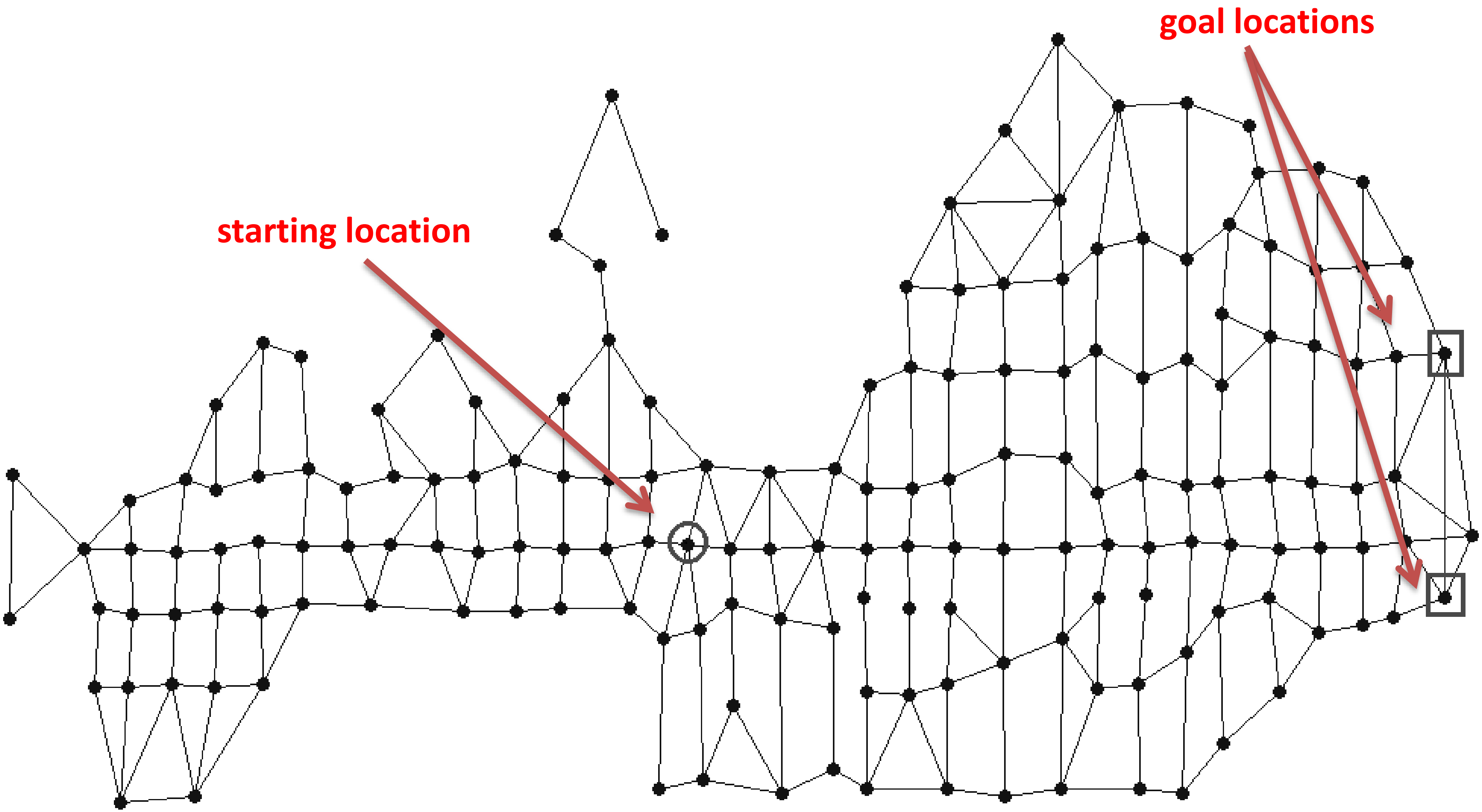


EXPO
MILANO 2015
FEEDING THE PLANET
ENERGY FOR LIFE
CHILDREN PARK
CIVIL SOCIETY
LAKE ARENA
BIO MEDITERRANEUM
ISLANDS
ARID ZONES
BIODIVERSITY PARK
PALAZZO ITALIA
FRUIT & LEGUMES
SPICES
EXPO CENTRE
RICE
COCOA
COFFEE
CEREALS & TUBERS
ZERO PAVILLION
ART & FOOD
FUTURE FOOD DISTRICT
OPEN-AIR THEATRE
THEMATIC AREAS
EVENT AREAS
SERVICE AREAS
OFFICIAL PARTICIPANTS SELF BUILT LOTS
OFFICIAL PARTICIPANTS CLUSTERS
PALAZZO ITALIA
CORPORATE
CIVIL SOCIETY

Example


EXPO
MILANO 2015
FEEDING THE PLANET
ENERGY FOR LIFE
LAKE ARENA
DISTRICT
THEMATIC AREAS
EVENT AREAS
SERVICE AREAS
OFFICIAL PARTICIPANTS SELF BUILT LOTS
OFFICIAL PARTICIPANTS CLUSTERS
PALAZZO ITALIA
CORPORATE
CIVIL SOCIETY

Example

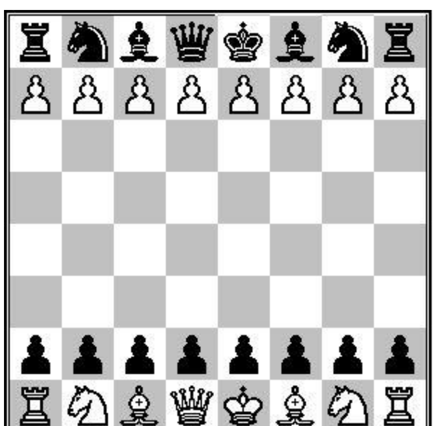


Specification

- How to **specify** a planning problem?
- First approach: provide the full state transition graph G (that’s what we did in the previous example)
- Most of the times is not an affordable option due to the combinatorial nature of the state space:

Specification

- How to **specify** a planning problem?
- First approach: provide the full state transition graph G (that's what we did in the previous example)
- Most of the times is not an affordable option due to the combinatorial nature of the state space:



- **Chess board**: approx. 10^{47} states
- We can specify the initial state and the transition function in some compact form (e.g., set of rules to generate next states)
- The planning problem "unfolds" as search progresses

Searching for a plan

- Objective: find a feasible plan, i.e., **any** sequence of actions bringing the world from the initial state to **a** goal state

- Classical approach: searching on the state transition graph

- Search algorithm:
  - iteratively visits nodes until a goal is eventually found
  - generates a search graph which is a sub-graph of G

- A search algorithm can be **systematic** or **non-systematic**

Systematic search

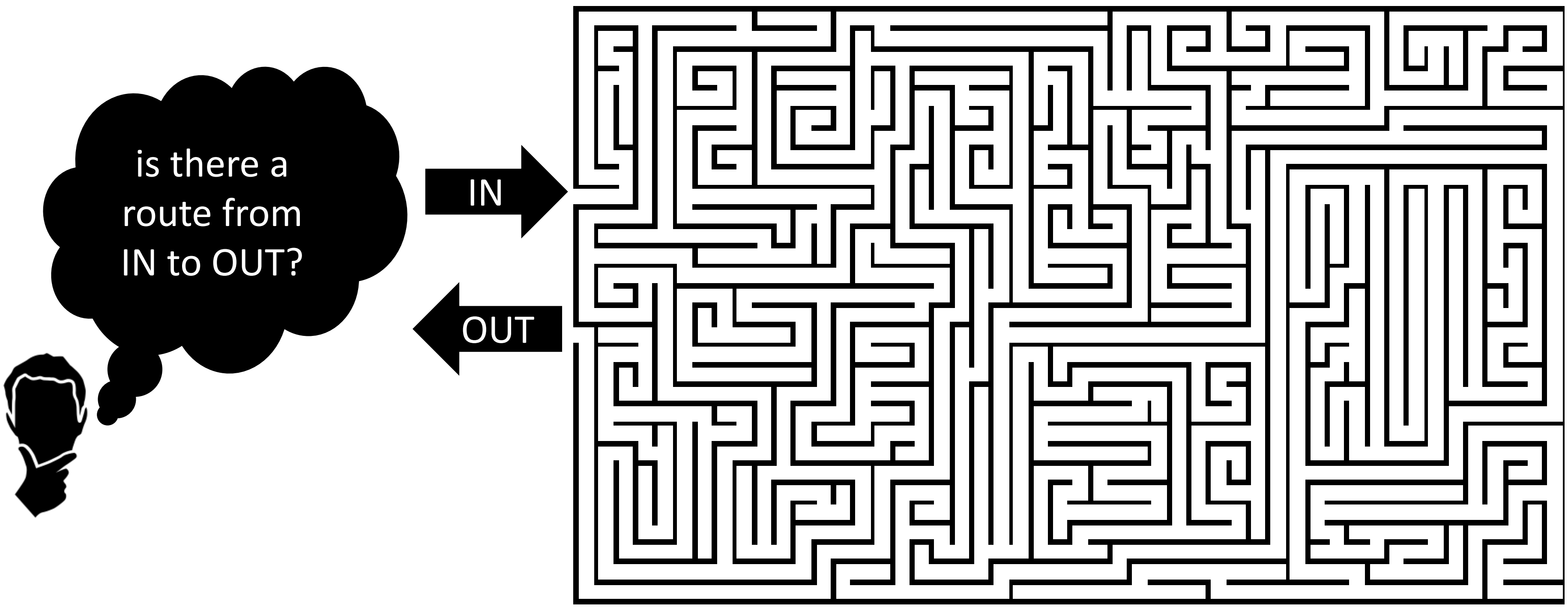
- If the graph is **finite** (finite state space) the algorithm will eventually visit **all reachable states**:
  - preventing redundant exploration suffices to enforce a systematic search
  - always terminates with a “yes” or “no” answer

- If the graph is infinite (infinite state space)?
  - if the answer is “yes” the algorithm must terminate
  - if the answer is “no”, it’s ok if it goes on forever but …
  - … all reachable states must be visited in the limit: as time goes to infinity, all states are visited (this definition is sound under the assumption of countable state space)

Non-systematic search

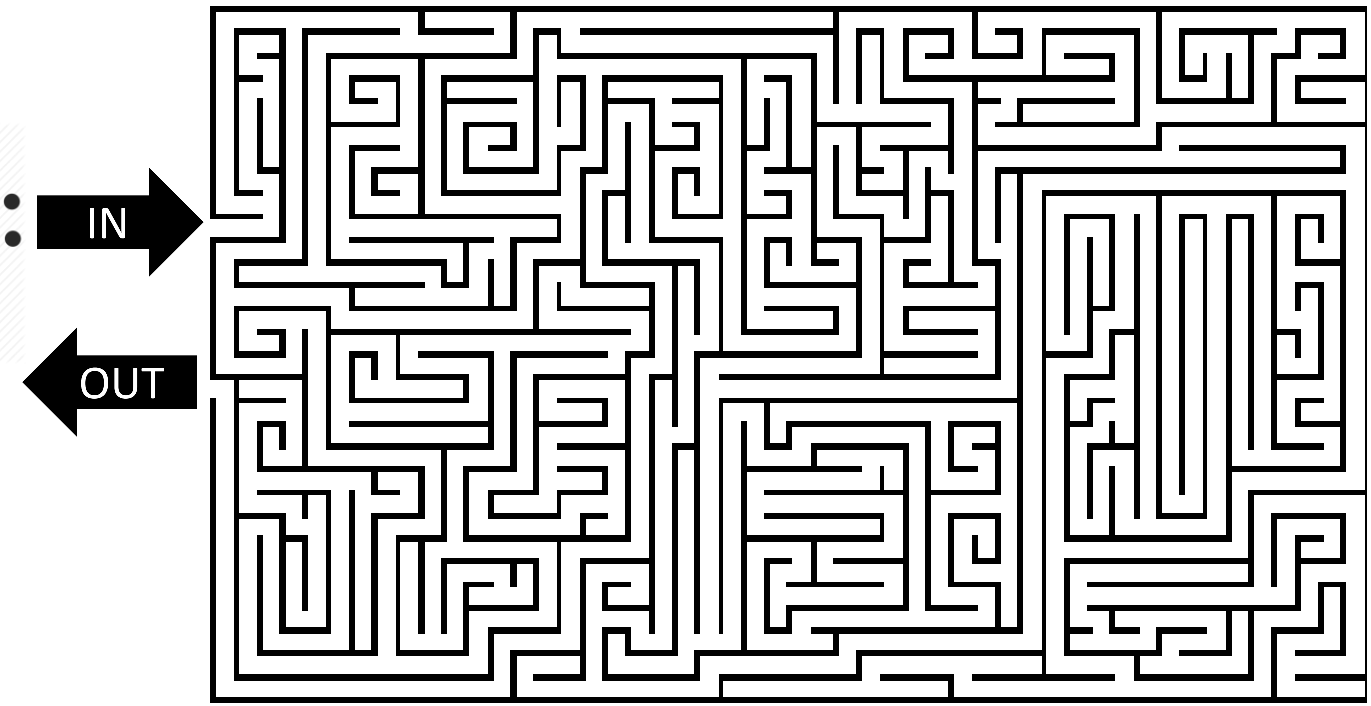
- The algorithm cannot guarantee to fully cover the state space
- It might terminate with a “no” answer (or not terminate) even if the problem admits a solution

Example



Example

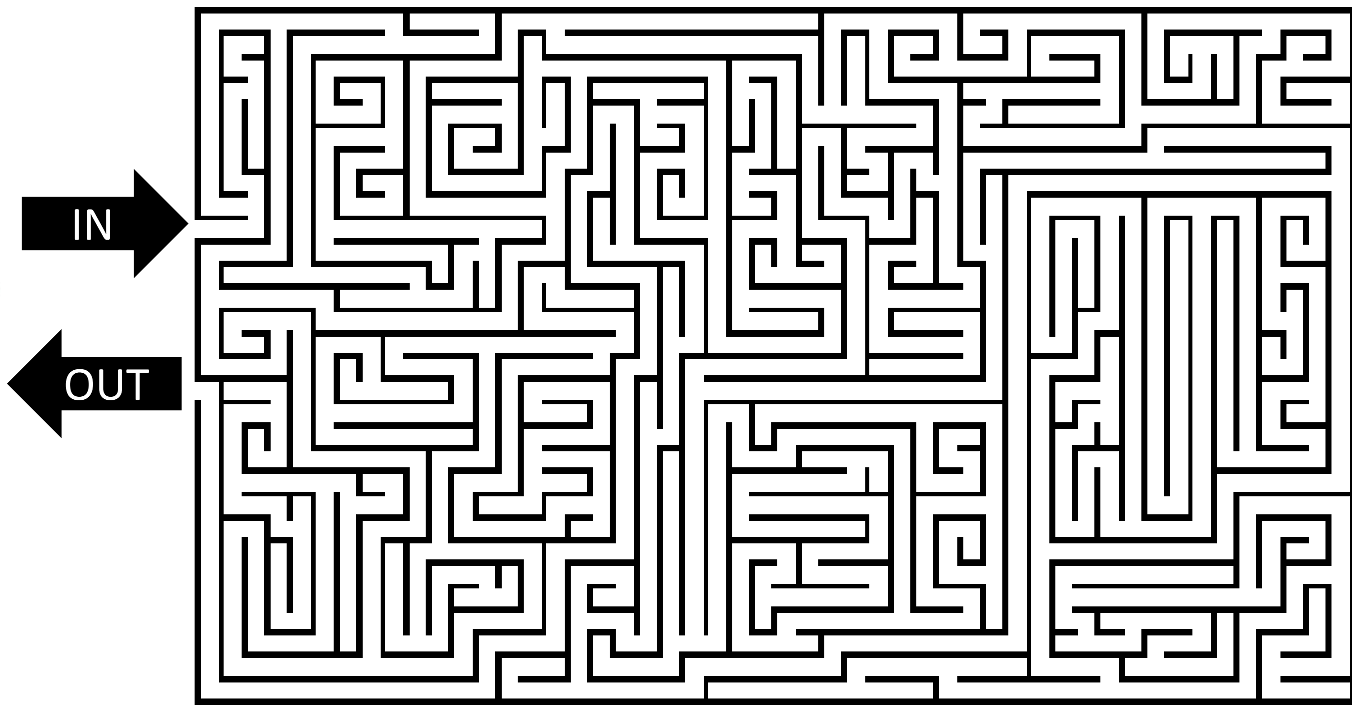
Systematic



- Searching along **multiple** trajectories (either concurrently or not)

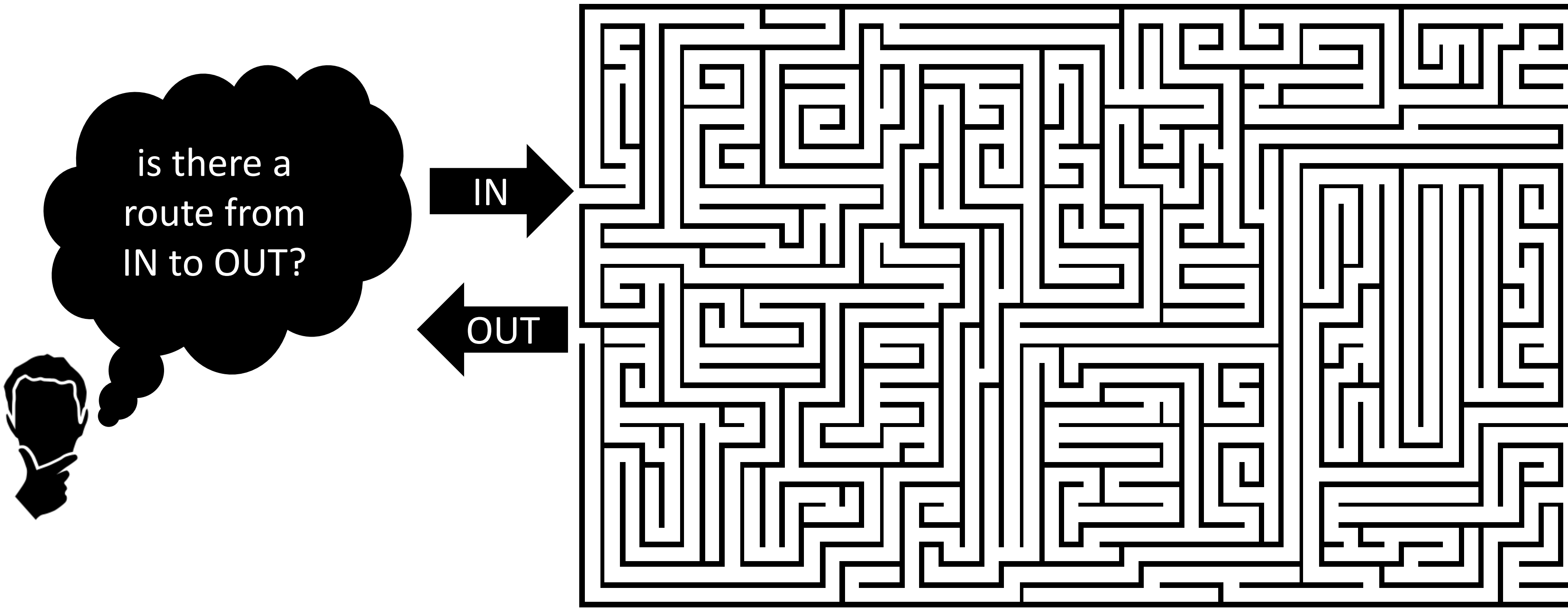
Example

Non-systematic



- Searching along a **single** trajectory

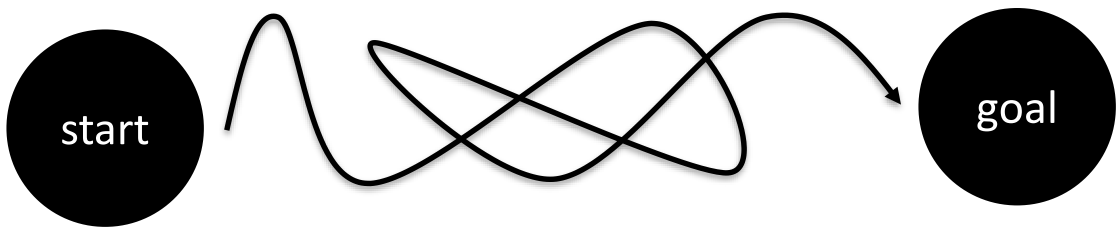
Example



- What would happen with an infinite labyrinth? (We actually need to require that the free area that can be reached from IN is infinite)

Forward search

- Idea: exploring the graph starting from the initial node, trying to find our way to the goal



- At any step during the search, a node can have one of three labels:
  - **unvisited**: still needs to be visited by the algorithm
  - **alive**: visited, but the algorithm still needs to visit nodes directly reachable from it
  - **dead**: visited, and any “next node” has been visited as well

- Alive nodes are store in a priority queue **Q**

Forward search

FORWARD_SEARCH

$Q.Insert(x_I)$ and mark x_I as visited
while Q not empty **do**
    $x \leftarrow Q.GetFirst()$
    if $x \in X_G$
        return SUCCESS
    forall $u \in U(x)$
        $x' \leftarrow f(x, u)$
        if x' not visited
            Mark x' as visited
            $Q.Insert(x')$
        else
            Resolve duplicate x'
return FAILURE

Sorts the queue and pops the first element

Expansion: getting the next states

Typically discards the vertex or updates some cost measure

Forward search

```
FORWARD_SEARCH
Q.Insert(x_I) and mark x_I as visited
while Q not empty do
    x ← Q.GetFirst()
    if x ∈ X_G
        return SUCCESS
    forall u ∈ U(x)
        x' ← f(x, u)
        if x' not visited
            Mark x' as visited
            Q.Insert(x')
        else
            Resolve duplicate x'
return FAILURE
```

Sorts the queue and pops the first element

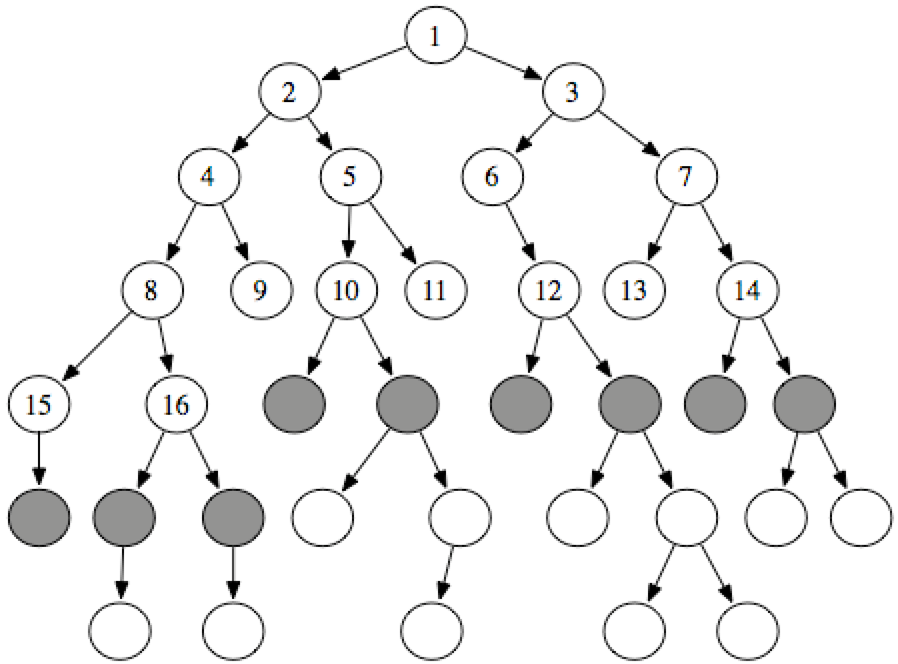
Expansion: getting the next states

Typically discards the vertex or updates some cost measure

- If the answer is “yes”, how to get the plan?
- How to check if a node is visited?
- It’s not an algorithm, it’s a template for an algorithm: to get an actual algorithm we need to specify the sorting criteria for Q

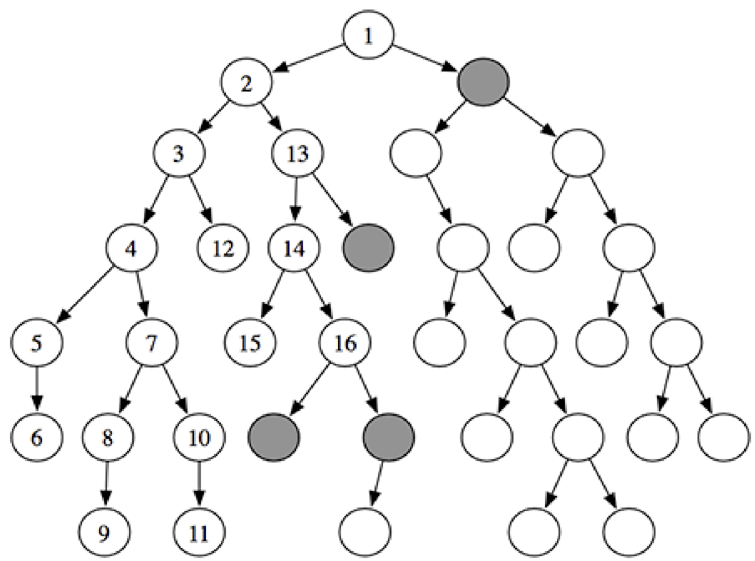
Breadth first search

- Q is a FIFO queue
- Plans with k+1 actions are evaluated after any plan with k actions has been searched
- If found, the plan is guaranteed to have the least number of actions (shortest plan)
- It's systematic, runs in O(|V| + |E|)



Depth first search

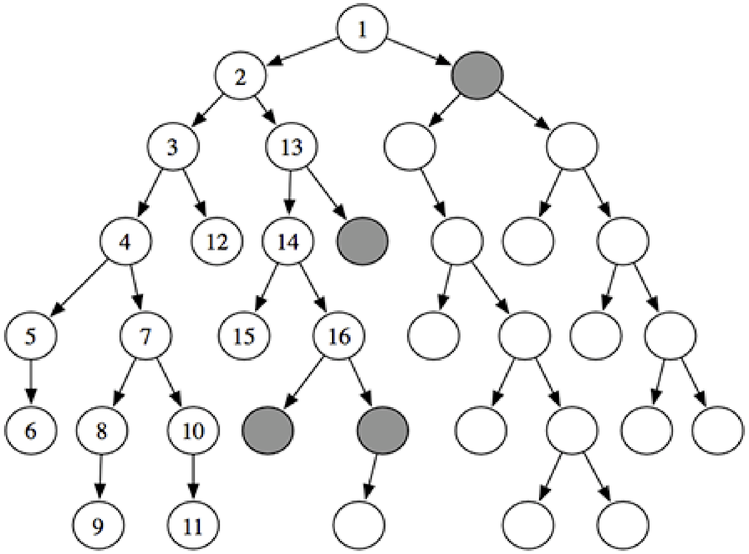
- Q is a stack (LIFO queue)
- More aggressive, searches long plans early
- Still runs in O(|V|+|E|)



- Systematic?

Depth first search

- Q is a stack (LIFO queue)
- More aggressive, searches long plans early
- Still runs in O(|V|+|E|)



- Systematic? Only with finite state spaces!

Dijkstra

- Introducing preferences in the expansion step: *which action do I try first?*
- We need to extend our problem formulation

$$l(x, u)$$

cost for picking action u in state x or…
… for going from state x to state f(x,u)

Dijkstra

- Introducing preferences in the expansion step: *which action do I try first?*
- We need to extend our problem formulation

$$l(x, u)$$

cost for picking action u in state x or…
… for going from state x to state f(x,u)

- Cost of a plan: sum of all the state transition costs for going from the initial state to the goal state
- During the search the algorithm keeps track of the **cost-to-come** for each alive state x

$$C(x)$$

it's the cost paid by the algorithm to
reach x from the initial state

Dijkstra

- Introducing preferences in the expansion step: *which action do I try first?*
- We need to extend our problem formulation

$$l(x, u)$$ cost for picking action u in state x or…
… for going from state x to state f(x,u)

- Cost of a plan: sum of all the state transition costs for going from the initial state to the goal state
- During the search the algorithm keeps track of the **cost-to-come** for each alive state x

$$C(x)$$ it's the cost paid by the algorithm to reach x from the initial state

- It's a forward search where Q is sorted according to C from smallest to highest **cost-to-come**
- Each state has an optimal cost which is initially unknown $C^*(x)$

Dijkstra

- Initially set $C^*(x_I) = 0$
- When expanding x to obtain state x', set

$$C(x') = C^*(x) + l(x, u)$$

- If x' was already visited and the newly discovered path induces a lower cost-to-come, update $C(x')$

- After running the whole forward search we get the minimum cost plan from the initial node to any other one

- Why? When we select x for expansion (getting the first node from Q) we are guaranteed that its current cost-to-come is optimal (that's the reason for the * in the above equation). Proof?