Recitation 1
Search 01

Uninformed Search

Administrations – schedule

	Sunday	Monday	Tuesday	Wednesday	Thursday
8-9					
9-10					
10-11			Lecture Chemistry 7		
11-12					
12-13					
13-14				Recitation (Maya) Shprinzak 116	
14-15					
15-16				Recitation (Yoni) Shprinzak 215	
16-17			Recitation (Yoni) Chemistry 7		
17-18				Recitation (Maya) Shprinzak 217	
18-19					
19-20					

Administrations – Communication

The course Moodle page: www.cs.huji.ac.il/~ai

News Forum, on the Moodle: News forum

Discussion Forum: for general questions Students Forum

Personal Forum. for appeals, personal requests (Miluim etc.)
On the Moodle: Personal forum

There will also be a forum for each exercise.

Our emails (please use them wisely):

Jeff – jeff@cs.huji.ac.il
Yoni – yoni.sher@mail.huji.ac.il
Maya – maya.cohen6@mail.huji.ac.il
Efraim (Tzar) – efraim.hazony@mail.huji.ac.il

Administrations – grading

theoretical assignments	best 4 out of 5 12%	One week each
programing assignments	20%	Two weeks each
quizzes	Search quiz – 6% KR/planning quiz – 11% learning/GT -11% Total: 28%	One hour each The quizzes will be computerized. Time extensions are automatic through מנהל תלמידים.
Final project	40%	~Month

Administrations – Appeals

Programing assignments	You can hand in a new version on the appeal link. It will be automatically graded (same as original submission). Penalty by number lines of changed: Let g be your grade after the corrections: If 1-2 lines changed, you get 0.95*g If 3-5 lines changed, you get 0.85*g If 6-8 lines changed, you get 0.75*g	Check your code before submission!
Other assignments	Send us a message via personal forum.	Try to be clear as you can

Administrations – Cheating

Louis CK on Sharks

Don’t do it!

It’s embarrassing
And relatively easy to spot
Also you will be kicked out of the course

Course map

> Search
>
> > Uninformed
>
> Informed

- Knowledge Representation
- Voting
- Planning
- Learning
- Game Theory

Uninformed Search

Today’s class:

- What’s search
- How we formalize it
  - Definition
  - Generalized search
  - Search properties and costs
- Search Strategies
  - DFS
  - BFS
  - UCS
  - Depth limited \ ID-DFS

Search

- Many real-life applications
  - Navigation
  - games
- As a building block for more advanced techniques
  - Planning
  - CSP
  - SAT
  - Learning

Search - definitions:

- Assumptions:
  - Single agent
  - Static
  - Deterministic
  - Fully observable
  - Finite state space

Search - definitions:

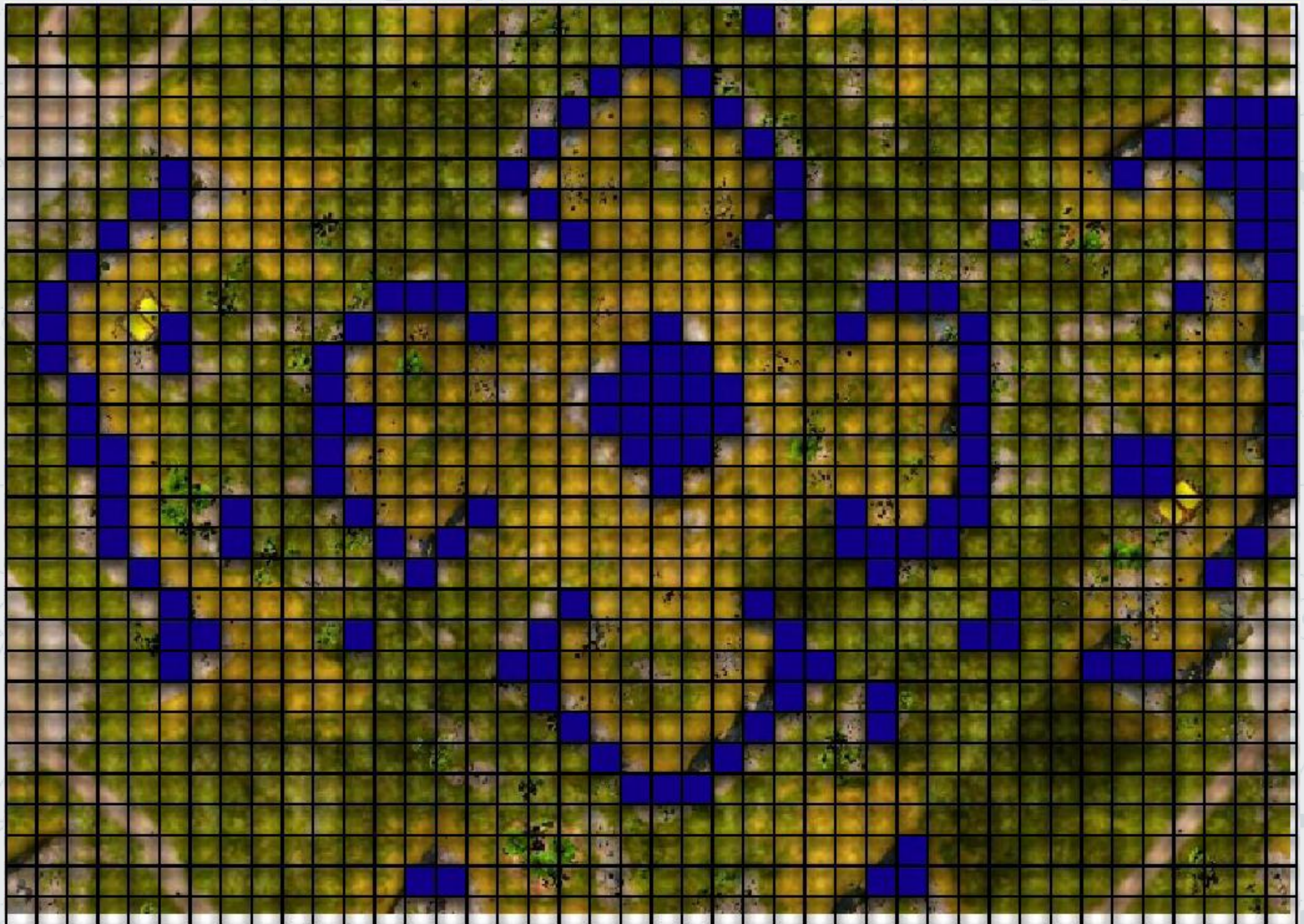
- States
  - Configurations the world can be in
- Actions
  - Taken by agents to move between states
- Initial state
- Goal state
- Solutions
  - A valid path (list of states) between an initial state and a goal state, optionally including the actions taken.

Example

Find a route on a map, from an initial location to a target location, while avoiding all obstacles

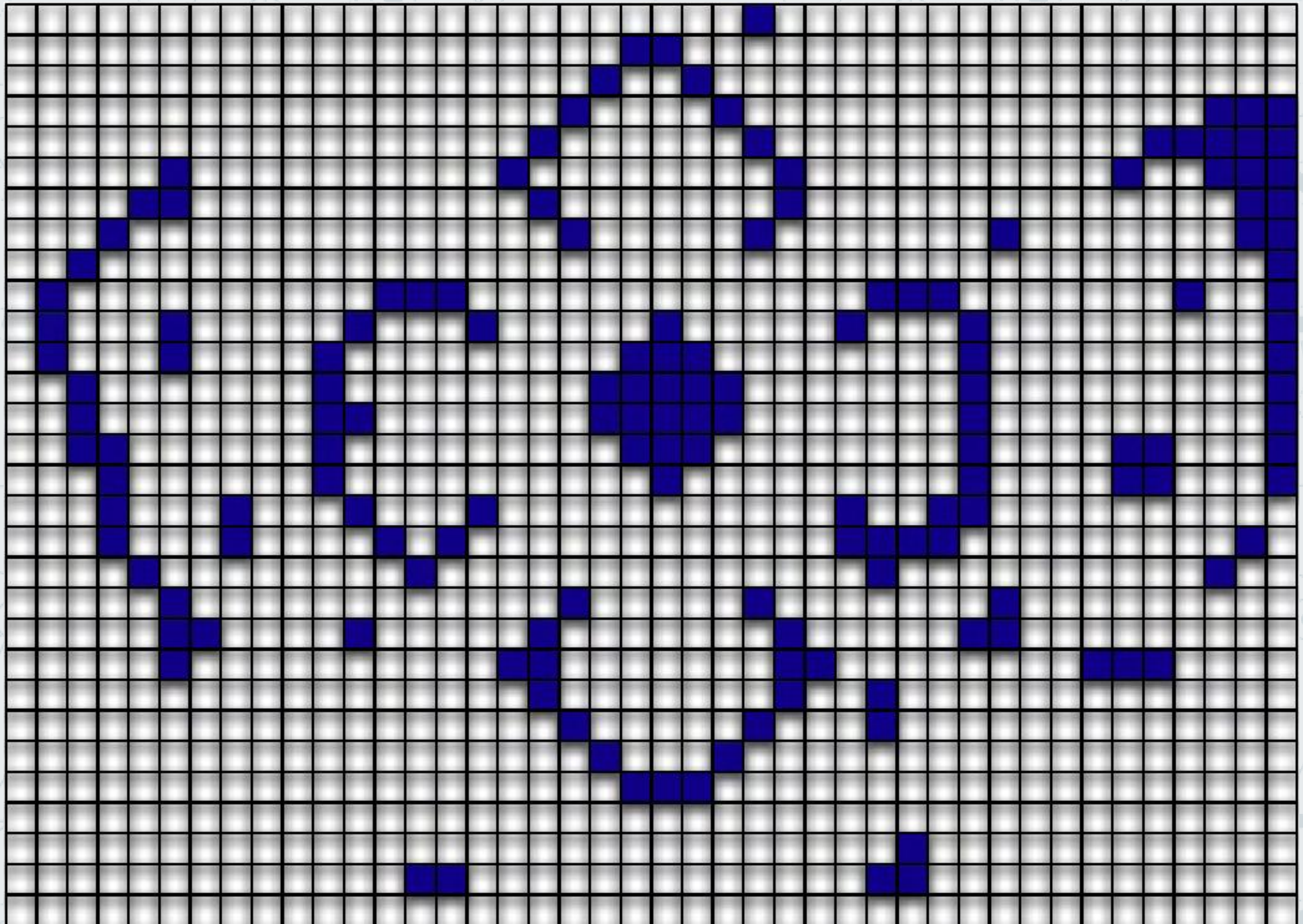
Example- simplified view

Ignore irrelevant data, keep only details that should be taken into account while computing the solution



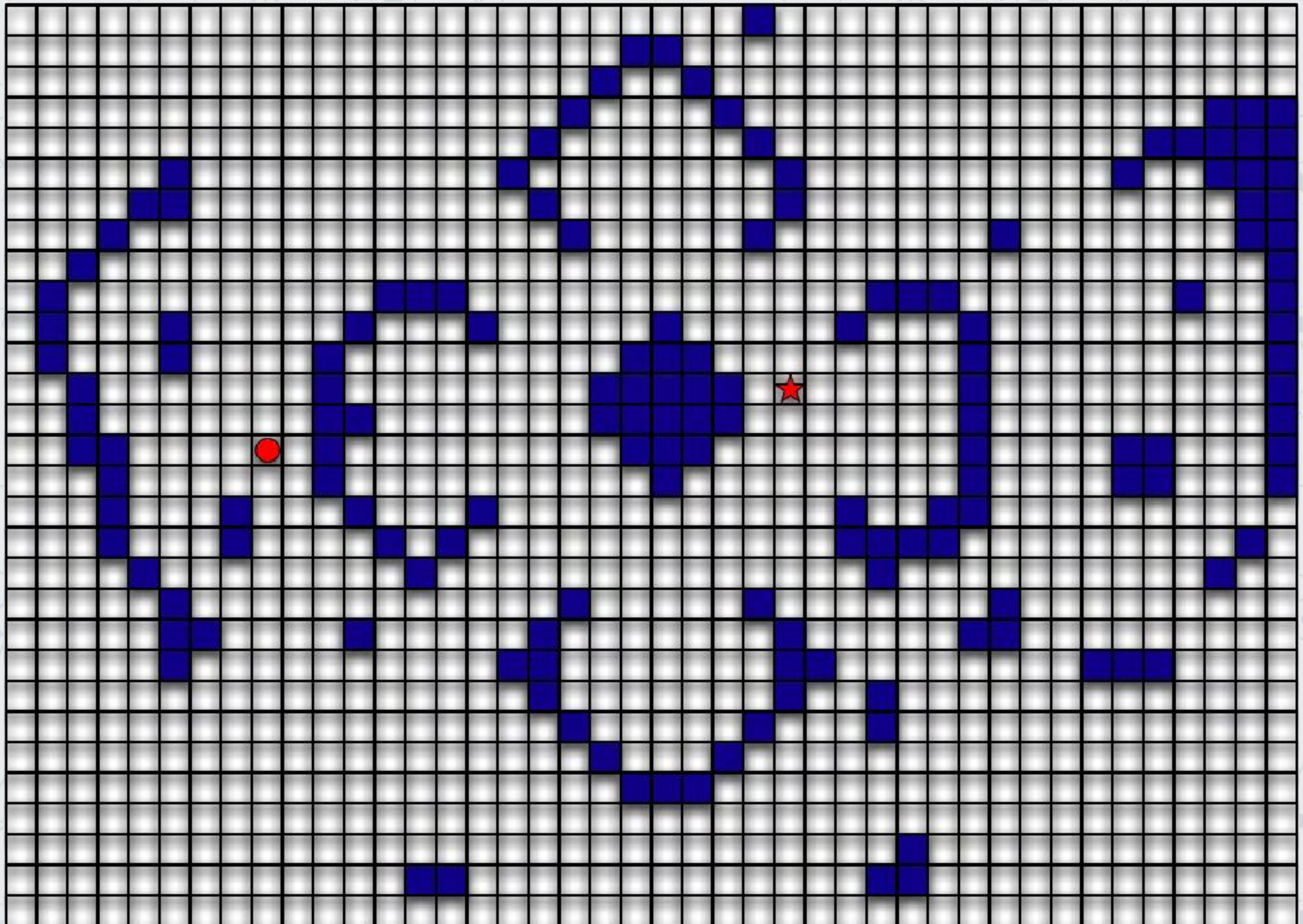
Example- states space

Ignore irrelevant data, keep only details that should be taken into account while computing the solution



Example- states space

Ignore irrelevant data, keep only details that should be taken into account while computing the solution



Pathfinding Problem Formulation

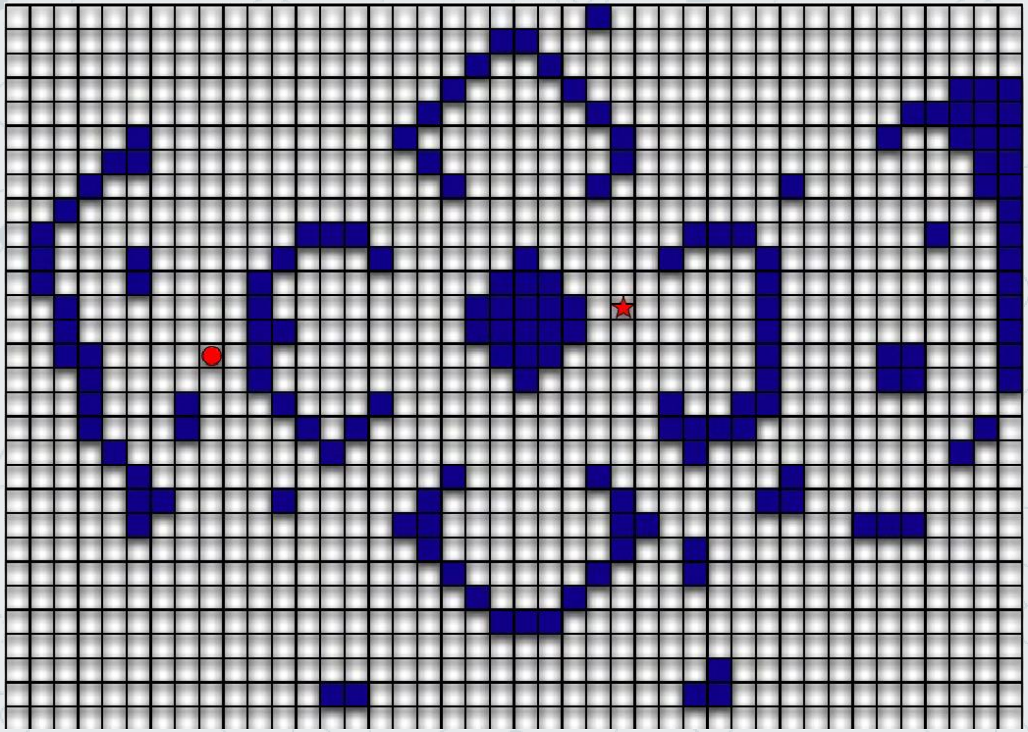
States?

Actions?

Initial state?

Goal state?

Solutions ?



Pathfinding Problem Formulation

States – each location (tile) is a state

Initial state – circle location

Successor function -
denote *Succ(s)* = set of action-state pairs
Each action (up, down, left, right) takes the agent to the corresponding adjacent tile

Goal test – star location

Solution – is a sequence of actions leading from the initial state to a goal state

Solution cost – here, number of actions executed

Formal Search Problem

- We denote a search problem as a tuple

$$< S, s_0, G, A, F, C >$$

  - S is a set of **states**
  - $s_0 \in S$ is the **start state**
  - G is a set of **goal states**
  - A is a set of **actions**
  - $F: S \times A \rightarrow S$ is a **transition function**
  - $C: S \times A \rightarrow \mathbb{R}^+$ is a **cost function**

Formal Search Problem

- We denote the solution as pair

$$< \{s_i\}_{i=0}^{n}, \{a_i\}_{i=0}^{n-1} >$$

  - s_0 is the **start state**
  - s_n in G
  - $s_i = F(s_{i-1}, a_{i-1}) \ \ \forall\, 0 \le i \le n$
  - **Solution cost** is the sum of all costs of actions on the path

$$\sum_{i=0}^{n-1} C(s_i, a_i)$$

Example- The 15-puzzle

S = set of matrices describing configurations of 15 tiles

A = moving a tile to an empty place

s_0, G = sets of some configurations of the 15 tiles

15	2	1	12
8		6	11
4	9	10	7
3	14	13	5

Initial state

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

Goal state

Example- Rush hour

S = set of matrices describing configurations on the board

A = moving a car to an empty place

s_0, G = sets of some configurations of the cars

Initial state

Goal state

Example- Rush hour

S = set of matrices describing configurations on the board

A = moving a car to an empty place

s_0, G = sets of some configurations of the cars

Initial state

Goal state

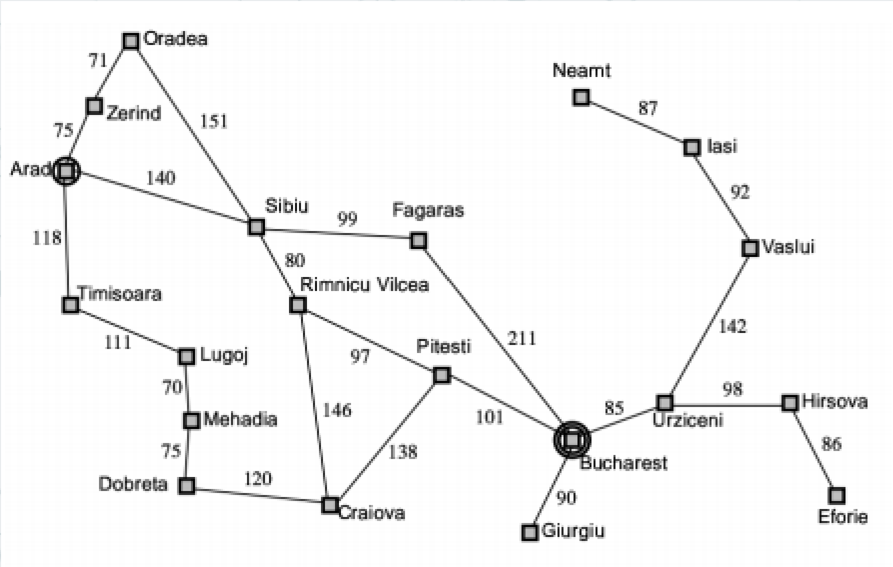
Example- Romania

S = various cities

A = drive between cities

G = be in Bucharest

C = cost of the path



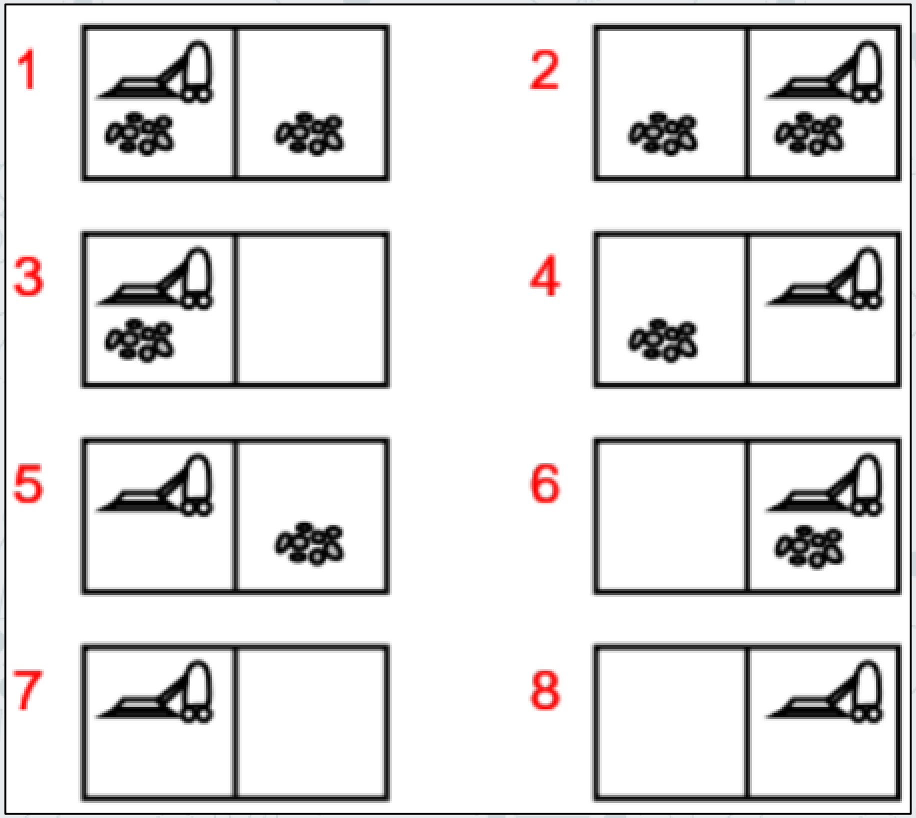
Example- Vacuum World

$S =$ dirt and robot location

$A =$
$\{Left, Right, Suck, NoOp\}$

$G =$ no dirt

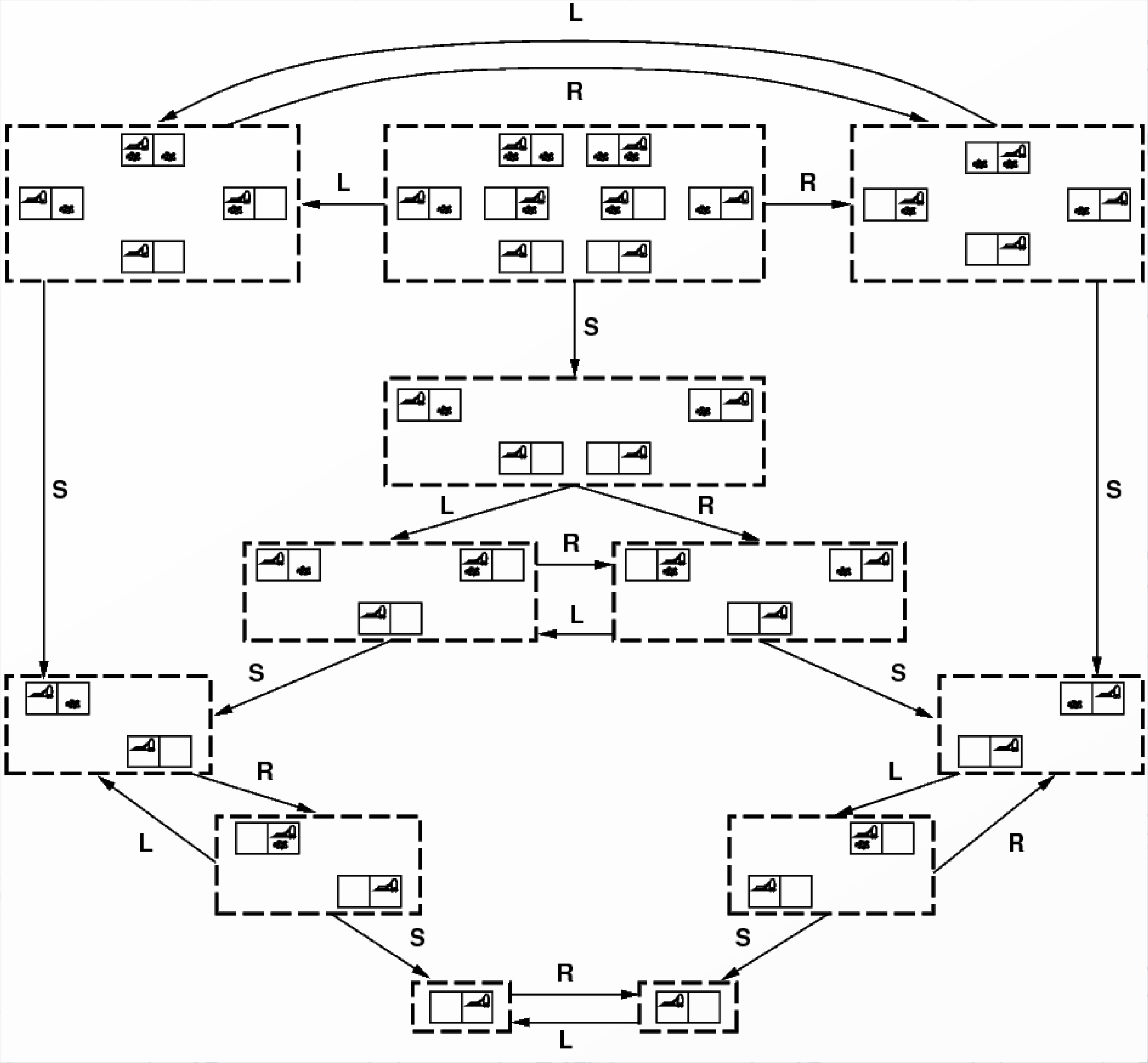
$C =$ 1 per action (0 for
$NoOp$)



Problem properties

	F Deterministic	F Non-deterministic
S Fully observable	the agent has percepts that can acquire his state Search	Agent can not know for sure the results of his actions
S Partially observable	agent might not know where it is – no percepts	online exploration problem

Example- Non observable problem



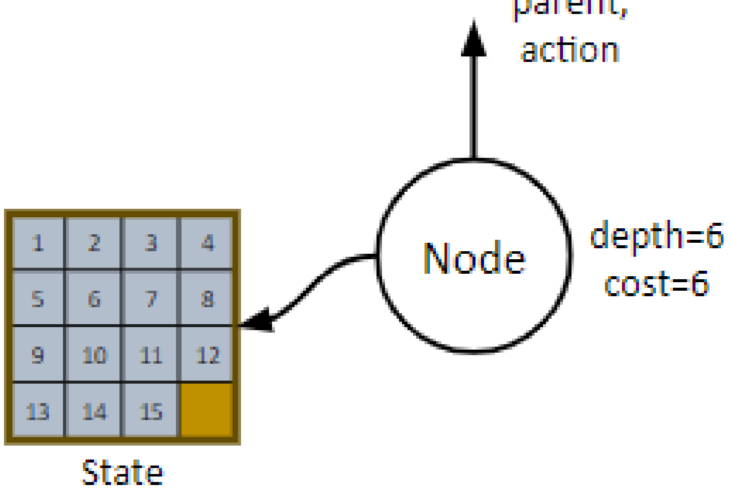
Implementing search

- A **state** is a representation of a physical configuration (in the real world)

- A **node** is a data structure that includes state, parents, actions, path cost, depth...

- Assume we have a label function that maps nodes to the states they represent:

$$L: V \to S$$



Implementing search – search tree

- Search tree is generated by exploring a state by a search algorithm
- Each node n corresponds to a unique state $L(n)$ (the same state can correspond to several nodes)
- The search tree is generated as follows:
  - The root corresponds to the initial state
- Generating the successors (children) of a node n:
  - $\forall a \in A, s' = F(L(n), a)$: create a new node n' and set $L(n') = s'$

- **Expanding a node** means generating its successors

Implementing search – search tree

- Let (V, E) be the search tree

- Label of nodes reflect states expanded –

$$\forall v \in V, L(v) \in S$$

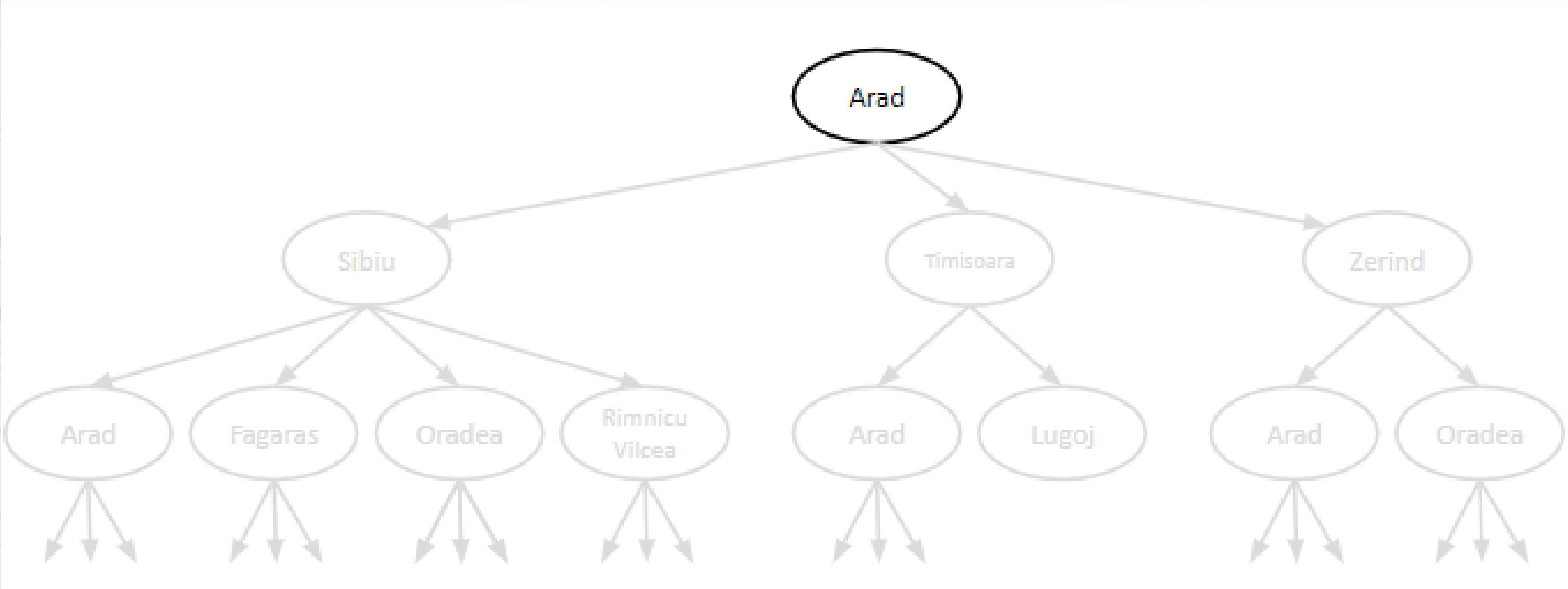
- The set of directed edges E reflect actions –
  $\forall\ (v1, v2) \in E$:

$$s_1 = L(v_1), s_2 = L(v_2) \Leftrightarrow \exists a \in A: s_2 = F(s_1, a)$$

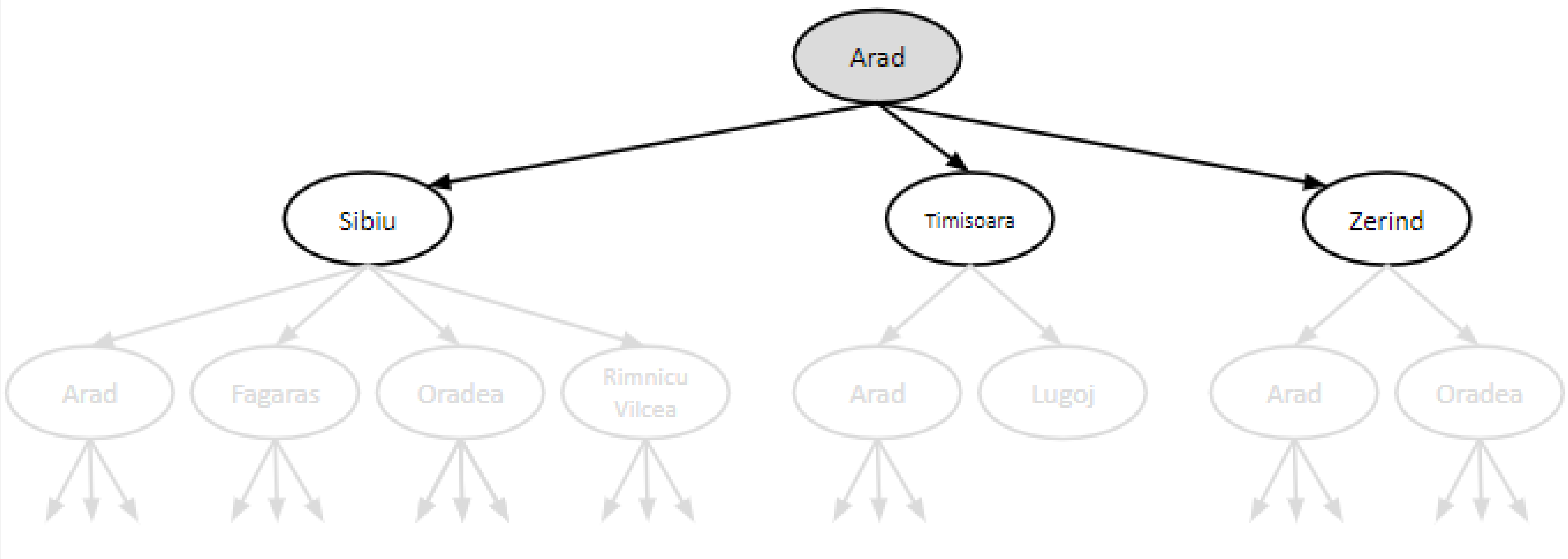
- The label of the root is the initial state

$$L\big(Root\ (V, E)\big) = s_0$$

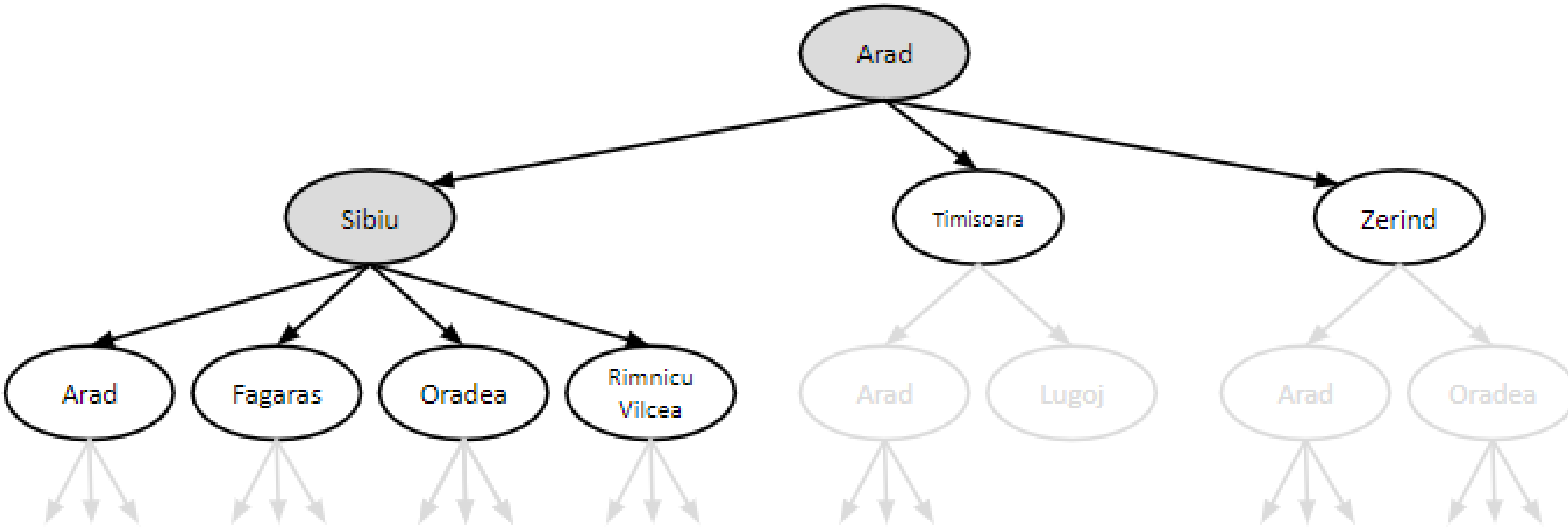
Example- Romania



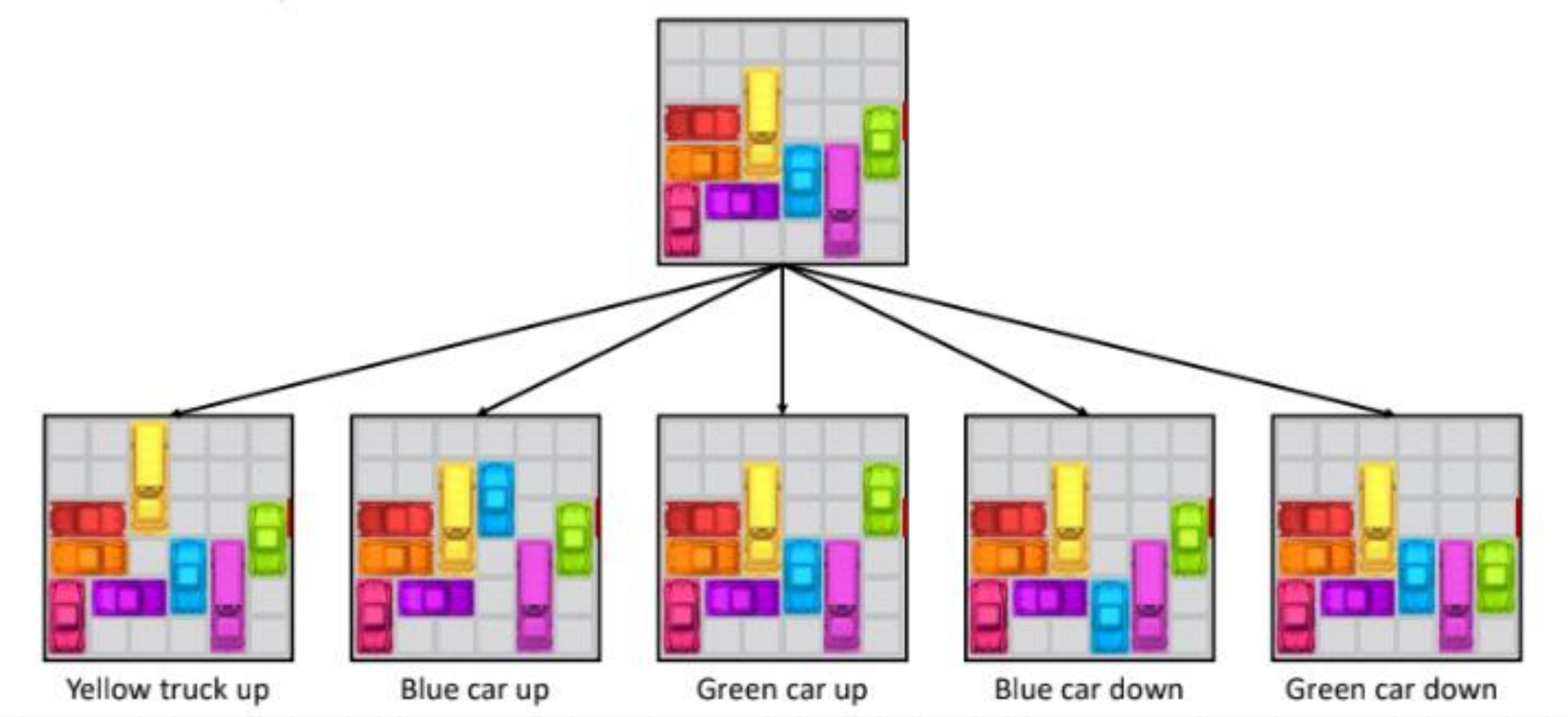
Example- Romania



Example- Romania



Example- Rush hour



Implementing search – general algorithm

Let $(V = \{v_1\}, E = \emptyset)$, s.t. $L(v_1) = s_0$, $fringe = \{v_1\}$

While $fringe \neq \emptyset$:

$Current \leftarrow choose(fringe)$

$fringe = fringe \backslash \{Current\}$

If $L(Current) \in G$, then DONE

return path to $Current$ from $Root(V, E)$

else:

let $V_{new} = \{v \mid \exists\, s, a\colon L(v) = s\, , s = f(L(Current\,), a)\,\}$

let $E_{new} = \{\, e = (Current, v\,)\colon v \in V_{new}\}$

$fringe = fringe \cup V_{new}$

$V = V \cup V_{new}, E = E \cup E_{new}$

Cannot solve the problem

Implementing search – correctness

- The $fringe$ (sometimes called active set) is the leaves (since we expand the search from the leaves) –

$$L(Leaves(V, E)) = Active$$

- Since the tree records our knowledge it is either that there is no $v \in V$ s.t. $L(v) \in G$
- or $\exists v \in V$ s.t. $L(v) \in (G \cap Active)$

Example- Water Jug

Given two empty jugs of 4L and 3L volume, fill the 4L jug with exactly 2 liters of water (no additional measuring devices are available)

- S = $\{ (x, y) \in [0{:}4] \times [0{:}3]\}$

- A = {Empty a jug, fill jug, pour water from one to another}

- $s_0 = (0,0)$

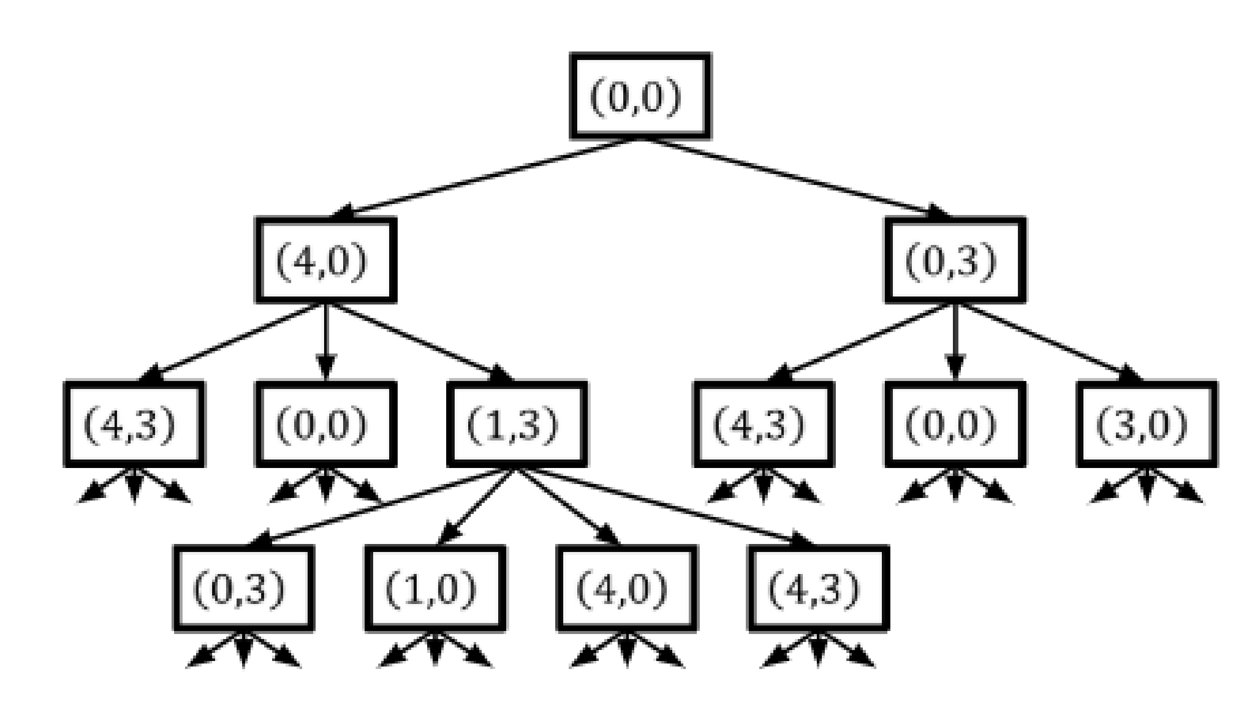
- $G \;= \{ (2,*) \}$

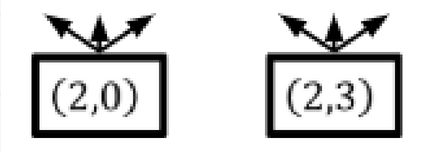
Example- Water Jug

Transition function:

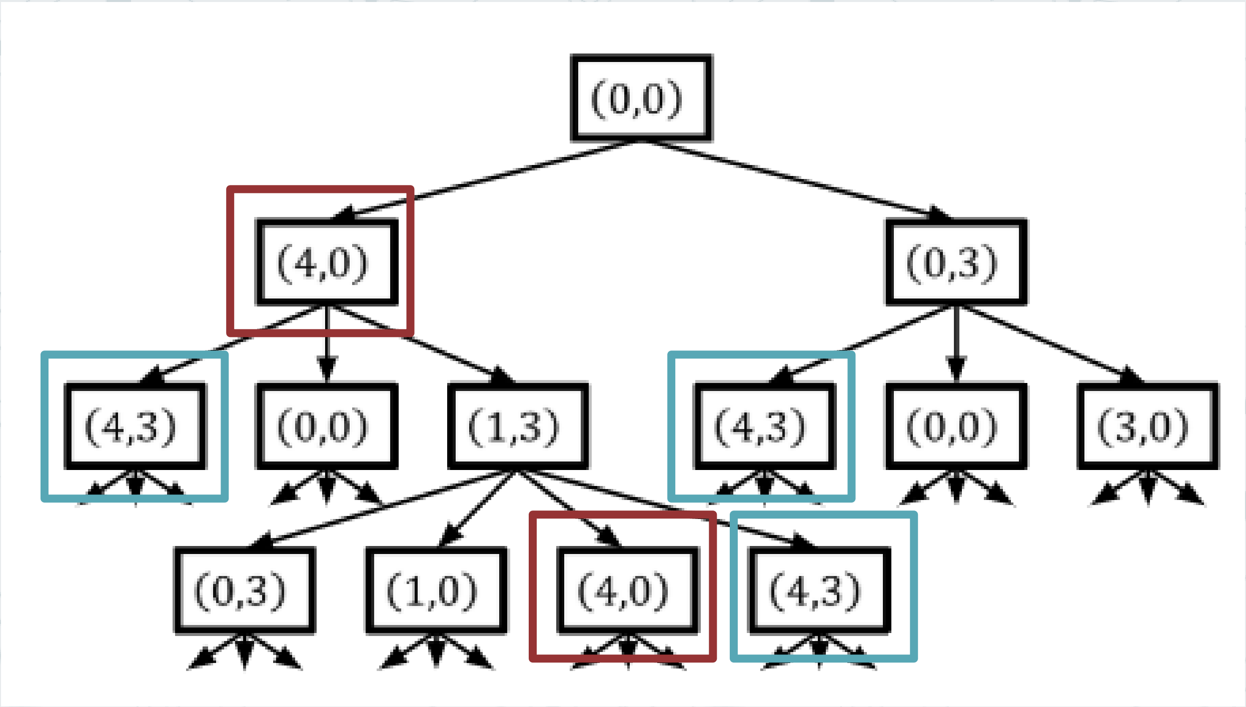
- Fill 4l : $(x, y \mid x < 4) \rightarrow (4, y)$
- Fill 3l : $(x, y \mid y < 3) \rightarrow (x, 3)$
- Empty 4l : $(x, y \mid x > 0) \rightarrow (0, y)$
- Empty 3l : $(x, y \mid y > 0) \rightarrow (x, 0)$
- Pour 3l to 4l : $(x, y \mid x + y \geq 4, y > 0) \rightarrow (4, y - (4 - x))$
- Pour 3l to 4l : $(x, y \mid x + y \leq 4, y > 0) \rightarrow (x + y, 0)$
- Pour 4l to 3l : $(x, y \mid x + y \geq 3, x > 0) \rightarrow (x - (3 - y), 3)$
- Pour 4l to 3l : $(x, y \mid x + y \leq 3, x > 0) \rightarrow (0, x + y)$

Example- Water Jug tree





Example- Water Jug tree



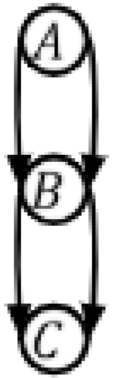
Can this plan cause a problem?

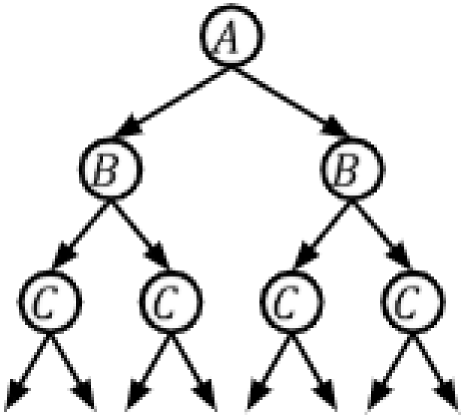
Repeated States

- Failure to detect repeated states can turn:

- A linear problem into an exponential one
- A finite state space into an infinite search tree

- To handle this, we turn from tree search to graph search





Implementing search – general algorithm

Let $(V = \{v_1\}, E = \emptyset)$, s.t. $L(v_1) = s_0$, $fringe = \{v_1\}$

While $fringe \neq \emptyset$:

$Current \leftarrow choose(fringe)$

$fringe = fringe \backslash \{Current\}$

If $L(Current) \in G$, then DONE

return path to $Current$ from $Root(V, E)$

else:

let $V_{new} = \{v \mid \exists\, s, a{:}\ L(v) = s\,, s = f(L(Current\,), a)\,\}$

let $E_{new} = \{\, e = (Current, v\,){:}\ v \in V_{new}\}$

$fringe = fringe \cup V_{new}$

$V = V \cup V_{new}, E = E \cup E_{new}$

Cannot solve the problem

Implementing search – general algorithm

Let $(V = \{v_1\}, E = \emptyset)$, s.t. $L(v_1) = s_0$, $fringe = \{v_1\}$, $closed = \emptyset$

While $fringe \neq \emptyset$:

$Current \leftarrow choose(fringe)$

$fringe = fringe \setminus \{Current\}$

If $L(Current) \in G$, then DONE

return path to $Current$ from $Root(V, E)$

else if $L(current) \notin closed$:

let $V_{new} = \{v \mid \exists\, s, a: L(v) = s, s = f(L(Current), a)\}$

let $E_{new} = \{e = (Current, v): v \in V_{new}\}$

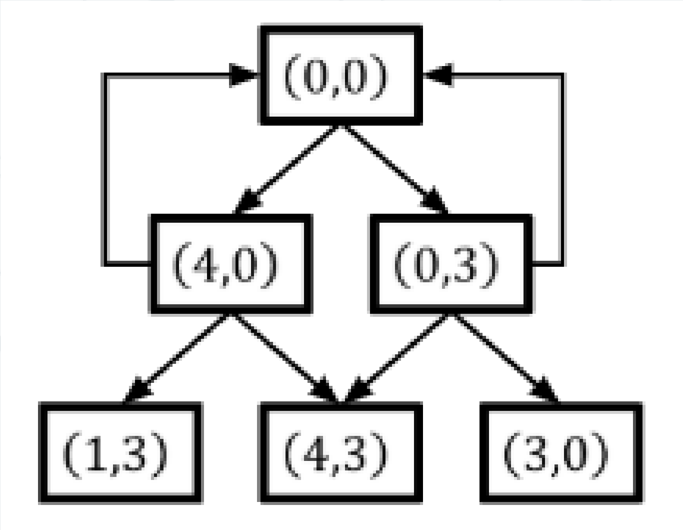
$fringe = fringe \cup V_{new}$

$V = V \cup V_{new}, E = E \cup E_{new}$

$closed = closed \cup \{L(current)\}$

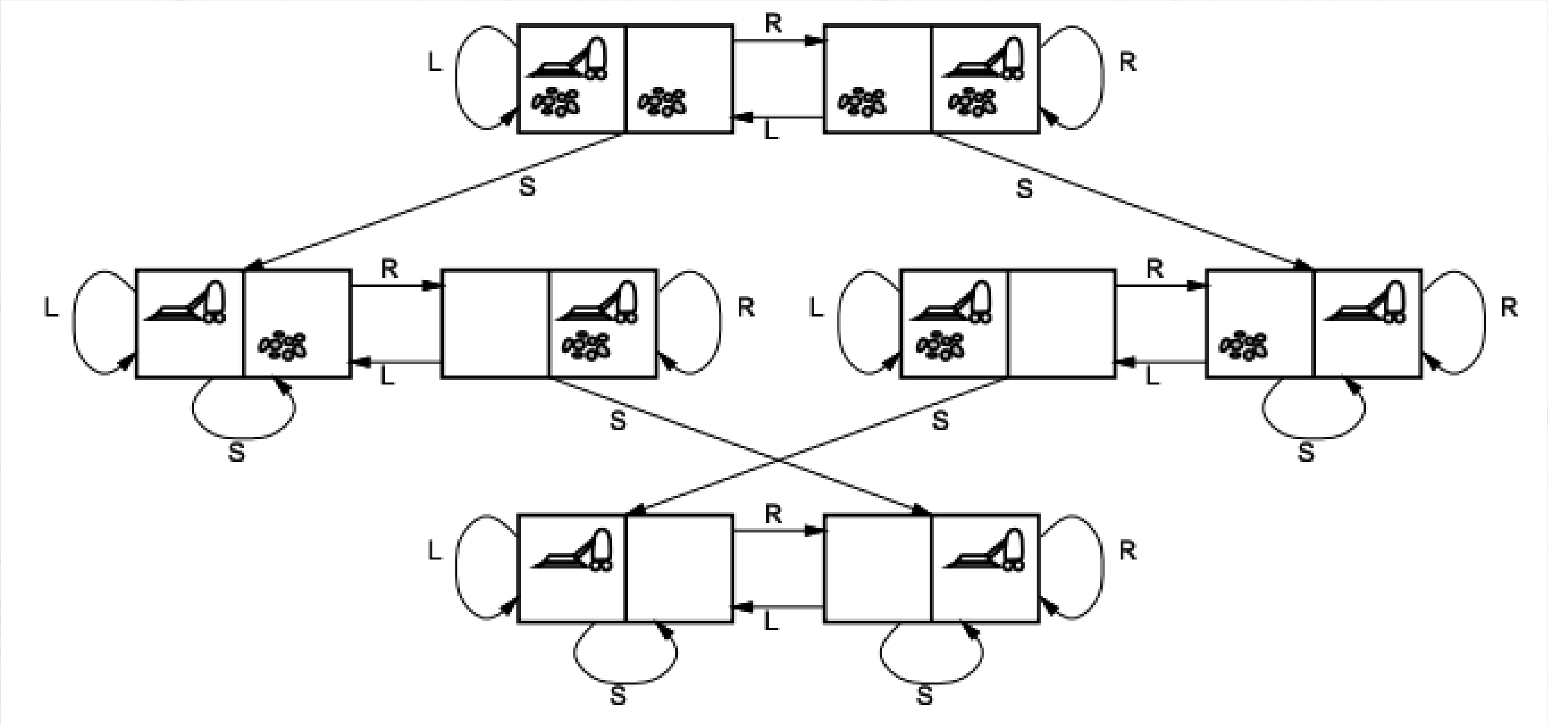
Cannot solve the problem

Example- Water Jug tree



Much better

Example - Vacuum World Search Graph



Search Quality - Formal Parameters

- **Completeness**:
  - guarantees to find a solution if exists
- **Soundness**:
  - guarantees a failure signal if no solution exist
- **Optimality**:
  - Find the solution that has the lowest cost among all solutions

- Computational Complexity:
- **Time** (number of operations applied during search);
- **Space** (number of nodes stored during search)

Computational complexity parameters

b	maximum branching factor of the search tree
d	depth of the shallowest solution
m	maximum depth of the state space (may be ∞)

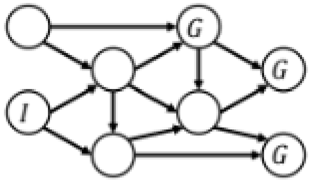
Design Choice

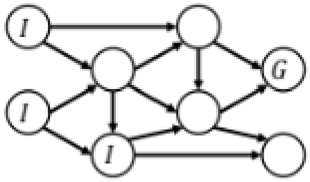
- Search algorithms work with abstract problem formulation, and are of the simplest forms of artificial intelligence.
- But how does it interface with the environment ?
- The Human Search Designer is the Interface, he uses known problems characteristics to formulate an efficient search problem:

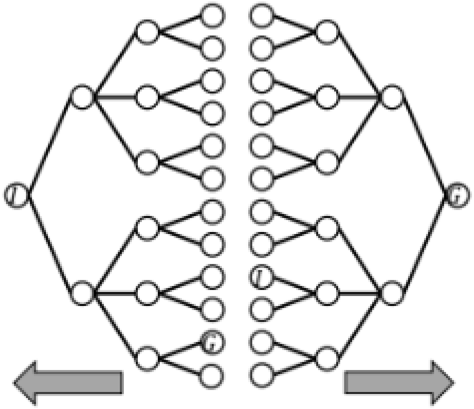
- Use Tree or Graph?
- How to handle state repetitions?
- Expanding direction (from start to goal, goal to start or both)?

Design Choice - Expansion Direction

- Start and Goal states are abstract, so it is possible to reverse them (from the search algorithm point of view). But why?
- Size of start and goal sets - easier to move towards the larger set
- Branching factor - move in the direction with lower branching factor







Search Strategies

- A search strategy is defined by picking the order of node expansion, i.e. the management of the fringe. Issues considered are:
- Ordering of the $fringe$
- Addition of new nodes to the $fringe$
- Reset of the $fringe$

- Note - Uninformed strategies use only the information available in the (general search) problem definition

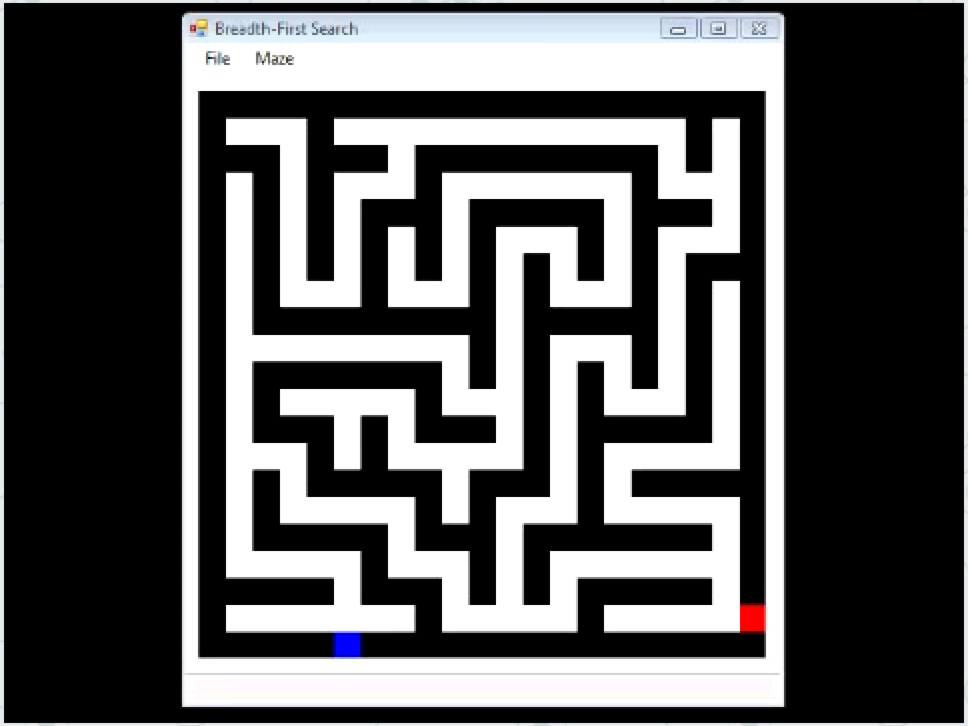
Breadth First Search

- Expand shallowest unexpanded node –
  - fringe Ordering : **Queue**

- Adding new nodes to the fringe: <u>Repetition check</u> in optimized version, but usually all nodes added

- Reset of the fringe : No reset, single run

Example – BFS Maze Traverse



BFS – Quality analysis

- **Is it complete?**
  - If a solution exists will we find it?
  - Yes
- **Is it sound?**
  - Assuming m is finite - If there is no solution, will we eventually stop?
  - Yes
- **Is it optimal?**
  - Does using this algorithm promise us that we find the solution with lowest cost?
  - If all actions cost 1 – Yes
  - In the general case - No

BFS – Quality analysis

Time?

- $1 + b + b^2 + b^3 + \cdots + b^d + b \cdot \left(b^d - 1\right) = O(b^{d+1})$

Space?

- Same as time. We keeps every node in memory. $O(b^{d+1})$

b	maximum branching factor of the search tree
d	depth of the shallowest solution
m	maximum depth of the state space (may be ∞)

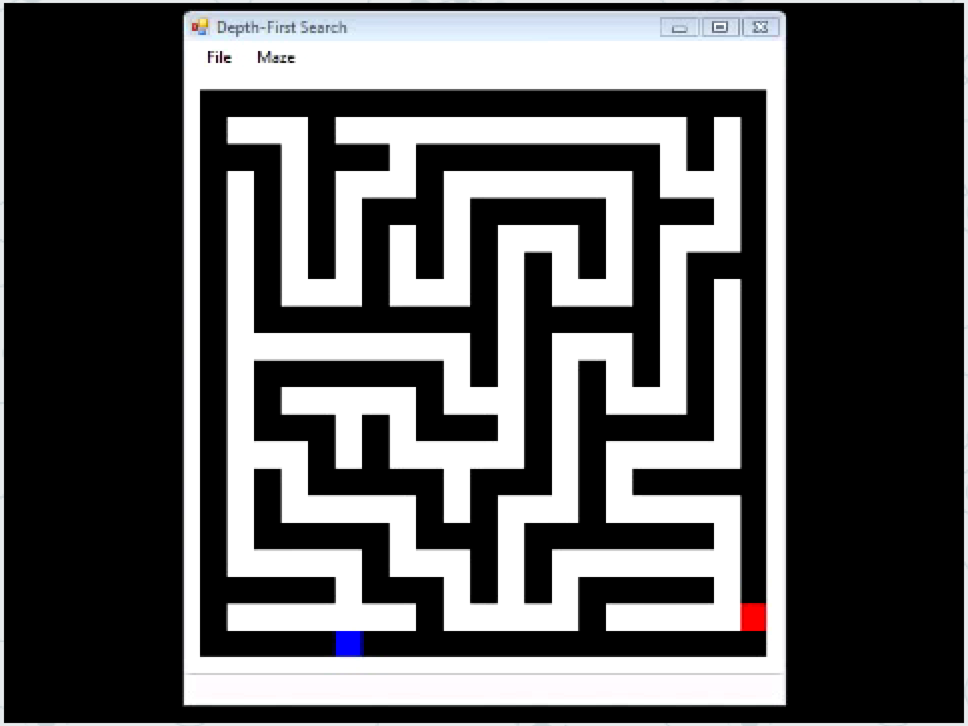
Depth First Search

- Expand deepest unexpanded node –
  - fringe Ordering : **Stack**

- Adding new nodes to the fringe: Repetition check in optimized version, but usually all nodes added

- Reset of the fringe : No reset, single run

Example – DFS Maze Traverse



DFS – Quality analysis

- **Is it complete?**
  - If a solution exists will we find it?
  - No. (fails in infinite-depth state spaces or spaces with loops).
  - If we avoid repeated states then it is complete for finite spaces.
- **Is it sound?**
  - Assuming m is finite - If there is no solution, will we eventually stop?
  - Yes
- **Is it optimal?**
  - No

DFS – Quality analysis

Time?

- $O(b^m)$
- May scan almost all states before finding a solution
- (if solutions are dense it might be faster then BFS)

Space?

- $O(b \cdot m)$
- Fringe holds only a thin string at any time
- - i.e. linear space

b	maximum branching factor of the search tree
d	depth of the shallowest solution
m	maximum depth of the state space (may be ∞)

Uniform-Cost Search

- Expand the next node which has the least total cost from the root –
  - fringe Ordering : **priority-queue**
- Adding new nodes to the fringe: Repetition check in optimized version, but usually all nodes added
- Reset of the fringe : No reset, single run

UCS – Quality analysis

- **Is it complete?**
  - If a solution exist we will find a solution
  - Yes
- **Is it sound?**
  - assuming m is finite- If no solution will we stop eventually?
  - Yes
- **Is it optimal?**
  - Does using this algorithm promise us that we find the solution with lowest cost?
  - Yes!

UCS – Quality analysis

Time?

- $1 + b + b^2 + b^3 + \cdots + b^d + b \cdot (b^d - 1) = O(b^{d+1})$
- (if cost = 1 per step)

Space?

- Same as time. We keeps every node in memory. $O(b^{d+1})$
- (if cost = 1 per step)

b	maximum branching factor of the search tree
d	depth of the shallowest solution
m	maximum depth of the state space (may be ∞)

Depth-limited Search

- Depth-limited DFS is a simple parameterized search algorithm.
- D-DFS has a depth parameter, denote by d_{max}
  - fringe Ordering : **stack**
- Adding new nodes to the fringe: as DFS unless further from start then d_{max}
- Reset of the fringe : as DFS

D-DFS – Quality analysis

- **Is it complete?**
  - No
  - (if $d > d_{max}$)
- **Is it sound?**
  - assuming m is finite- If no solution will we stop eventually?
  - Yes
- **Is it optimal?**
  - No

D-DFS – Quality analysis

- **Time?**
  - $O(b^{d_{max}})$
- **Space?**
  - $O(b \cdot \ d_{max})$

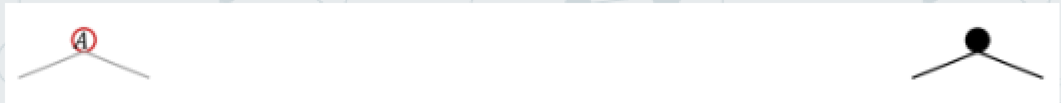
b	maximum branching factor of the search tree
d	depth of the shallowest solution
m	maximum depth of the state space (may be ∞)

Iterative Deepening DFS

- ID-DFS is based on D-DFS, incrementally increasing d_{max}
  - fringe Ordering : **stack**
- Adding new nodes to the fringe: as D-DFS
- Reset of the fringe : upon failure increase d_{max}

Example – ID-DFS

$d_{max} = 0$

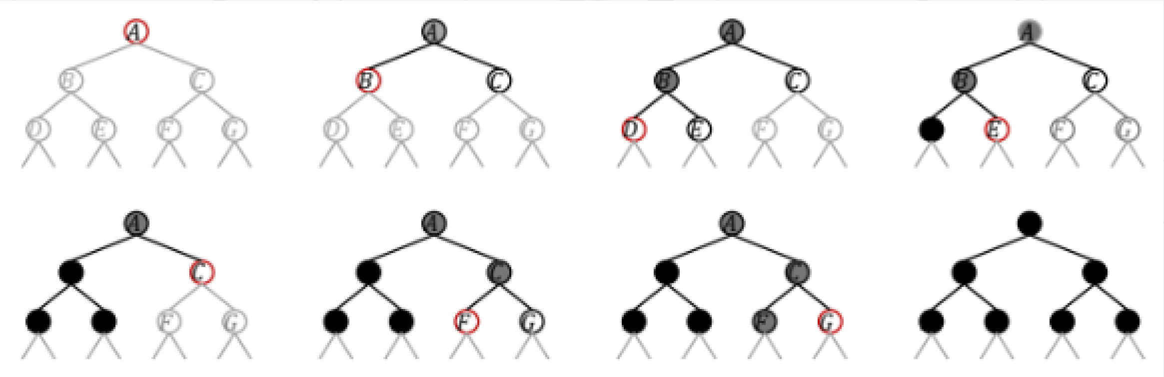


Example – ID-DFS

$d_{max} = 1$

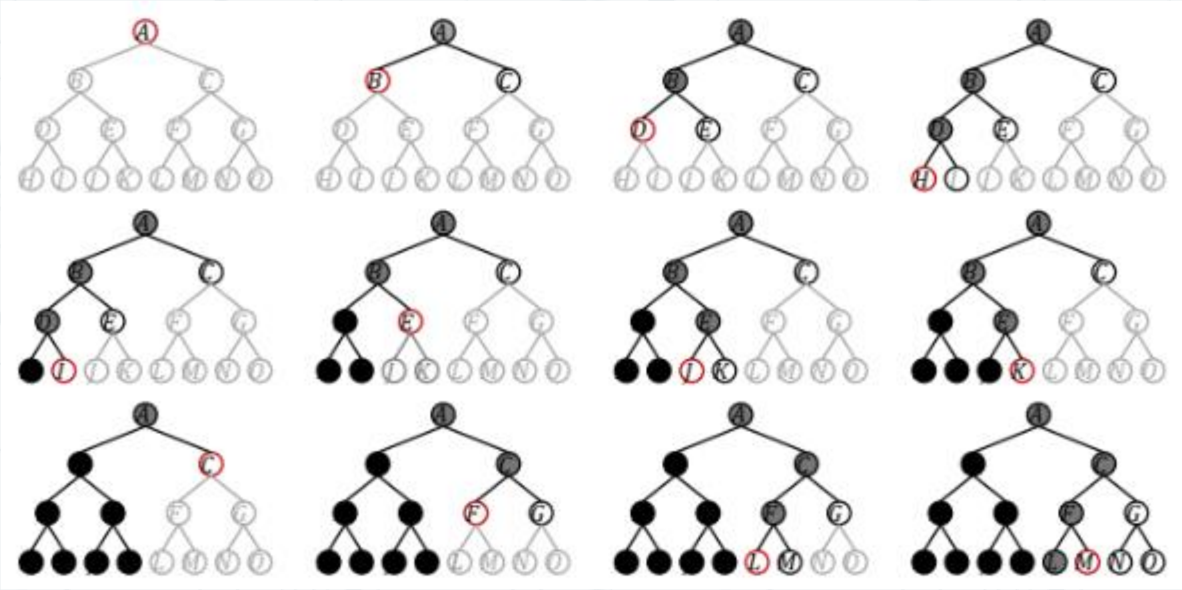
Example – ID-DFS

$d_{max} = 2$



Example – ID-DFS

$d_{max} = 3$



ID-DFS – Quality analysis

- **Is it complete?**
  - Yes
- **Is it sound?**
  - Yes
- **Is it optimal?**
  - If we increase d_{max} by 1 per step – Yes
  - In the general case – No

ID-DFS – Quality analysis

Time?

- $b^1 + b^2 + b^3 + \cdots + b^d + b^{d+1}$
- $= O(b^{d+1})$

Space?

- $O(b \cdot d)$

b	maximum branching factor of the search tree
d	depth of the shallowest solution
m	maximum depth of the state space (may be ∞)

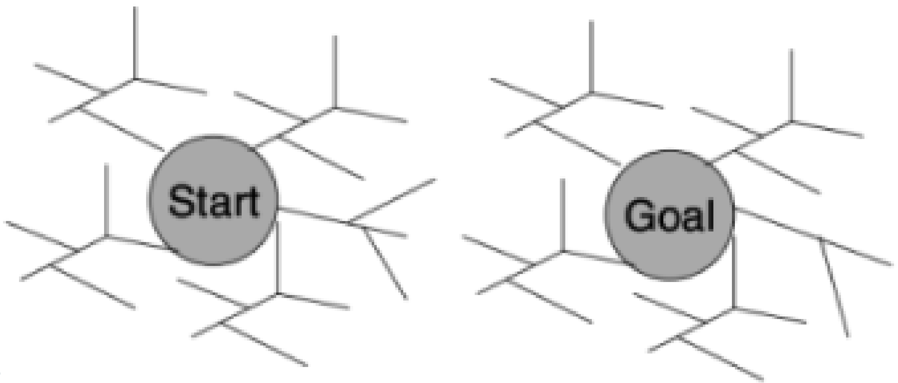
Performance summary

	BFS	DFS	UCS	D-DFS	ID-DFS
Complete?	Yes	Yes**	Yes	No	Yes
Sound?	Yes**	Yes**	Yes**	Yes	Yes**
Optimal	Yes*	No	Yes	No	Yes*
Time	$O(b^{d+1})$	$O(b^m)$	$O(b^{d+1})$	$O(b^{d_{max}})$	$O(b^{d+1})$
Space	$O(b^{d+1})$	$O(b \cdot m)$	$O(b^{d+1})$	$O(b \cdot d_{max})$	$O(b \cdot d)$

Assuming that b < ∞
* if cost=1 per step
**if m < ∞

Bidirectional Search

- Recall we considered expansion direction. We may run two simultaneous searches, forward from the start state, backwards from the goal
- $b^{\frac{d}{2}} + b^{\frac{d}{2}} \ll b^d$
- Where could problems arise?
  - Is finding a predecessor state as easy as finding successor state ?
  - How do you check whether the frontiers of the two searches intersect?



Summary

- Problem formulation usually requires abstracting away real-world details to define a state space that can feasibly be explored.
- Variety of uninformed search strategies
- Iterative deepening search uses only linear space and not much more time than other uninformed algorithms
- Graph search can be exponentially faster and more efficient than tree search