Problem Solving with AI Techniques
Reasoning as Search Problems

Paul Weng

UM-SJTU Joint Institute

VE593, Fall 2018

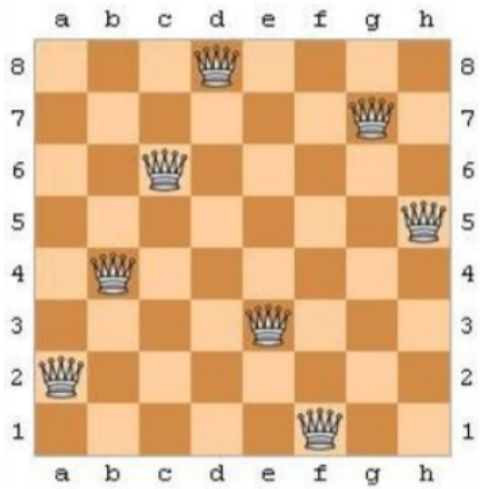
Mathematical and logical puzzles

- Mathematical puzzles, e.g., crypto-arithmetic, magic squares
- Logical puzzles, e.g., boolean formulas (SAT), N-queens
- Sudoku

$$\begin{array}{cccccc} & S & E & N & D \\ + & M & O & R & E \\ \hline = & M & O & N & E & Y \end{array}$$

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

A. Dürer, *Melencolia I* (1514)



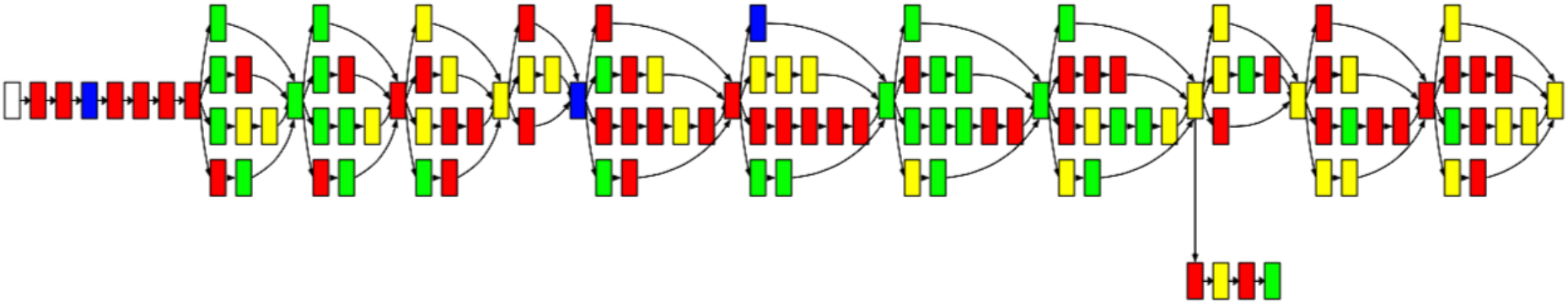
How long does it take to cross the bridge?

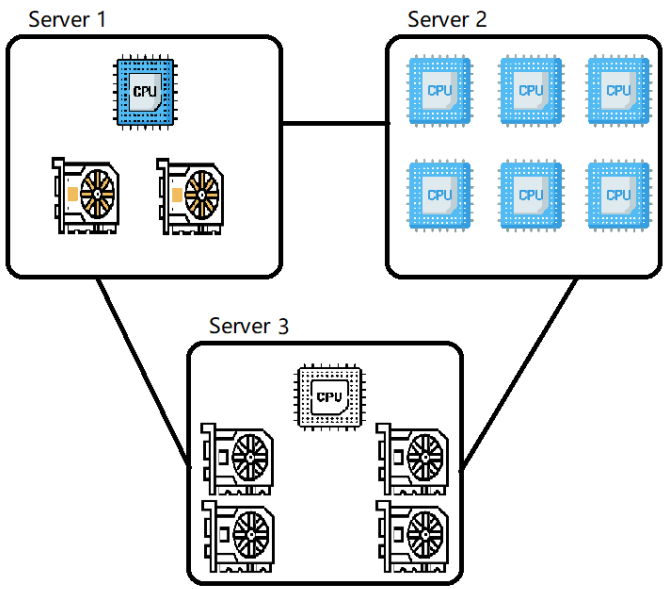
Four people come to a river in the night. There is a narrow bridge, but it can only hold two people at a time. They have one torch and, because it's night, the torch has to be used when crossing the bridge. Person A can cross the bridge in 1 minute, B in 2 minutes, C in 5 minutes, and D in 8 minutes. When two people cross the bridge together, they must move at the slower person's pace. The question is, can they all get across the bridge if the torch lasts only 15 minutes?

When can I move in?

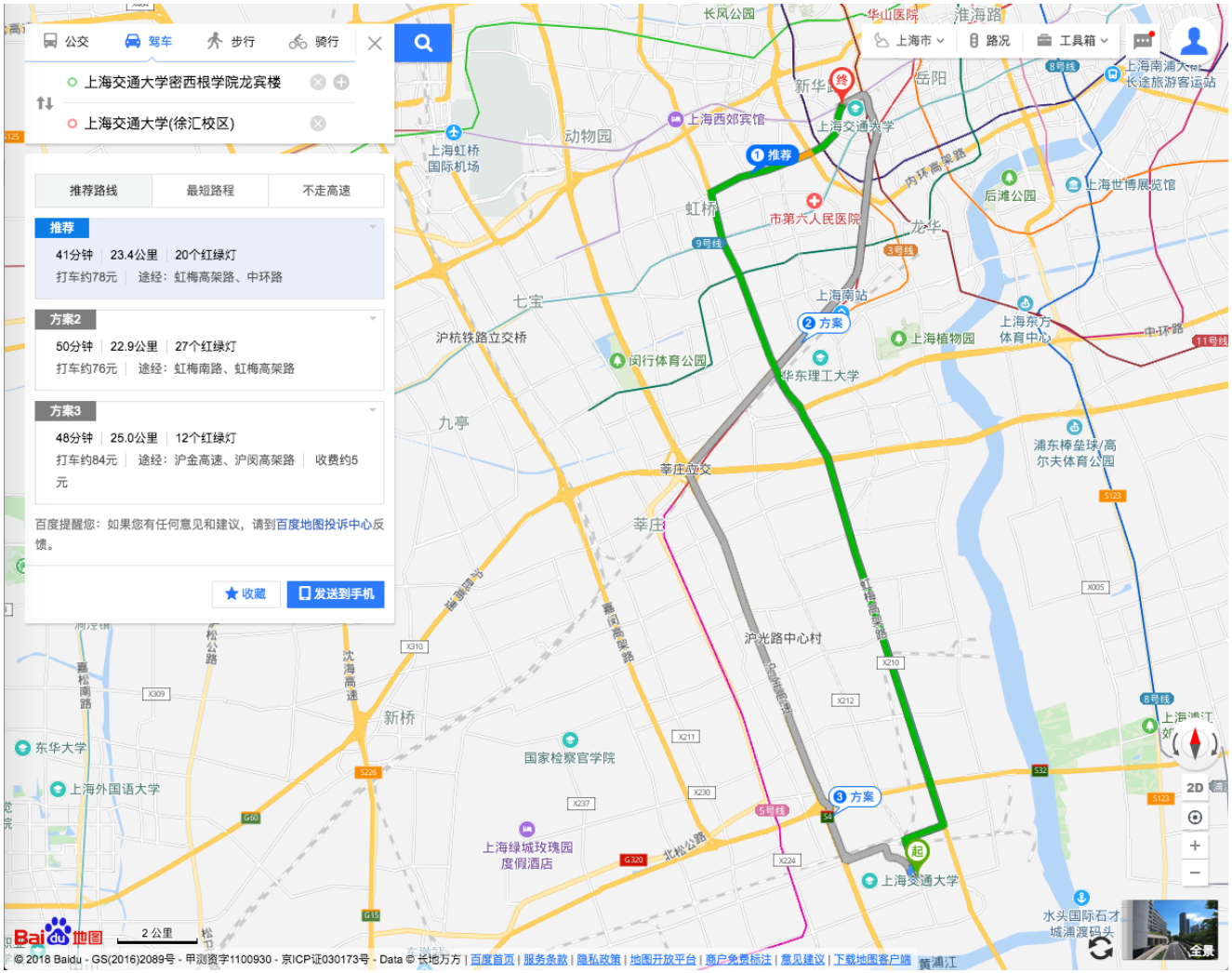
Task	Description	Duration	Predecessor
a	Erecting walls	7	none
b	Carpentry for roof	3	a
c	Roof	1	b
d	Installations	8	a
e	Facade painting	2	c & d
f	Windows	1	c & d
g	Garden	1	c & d
h	Ceilings	3	a
i	Painting	2	f & h
j	Moving in	1	i

How to optimize the execution of computational graphs?

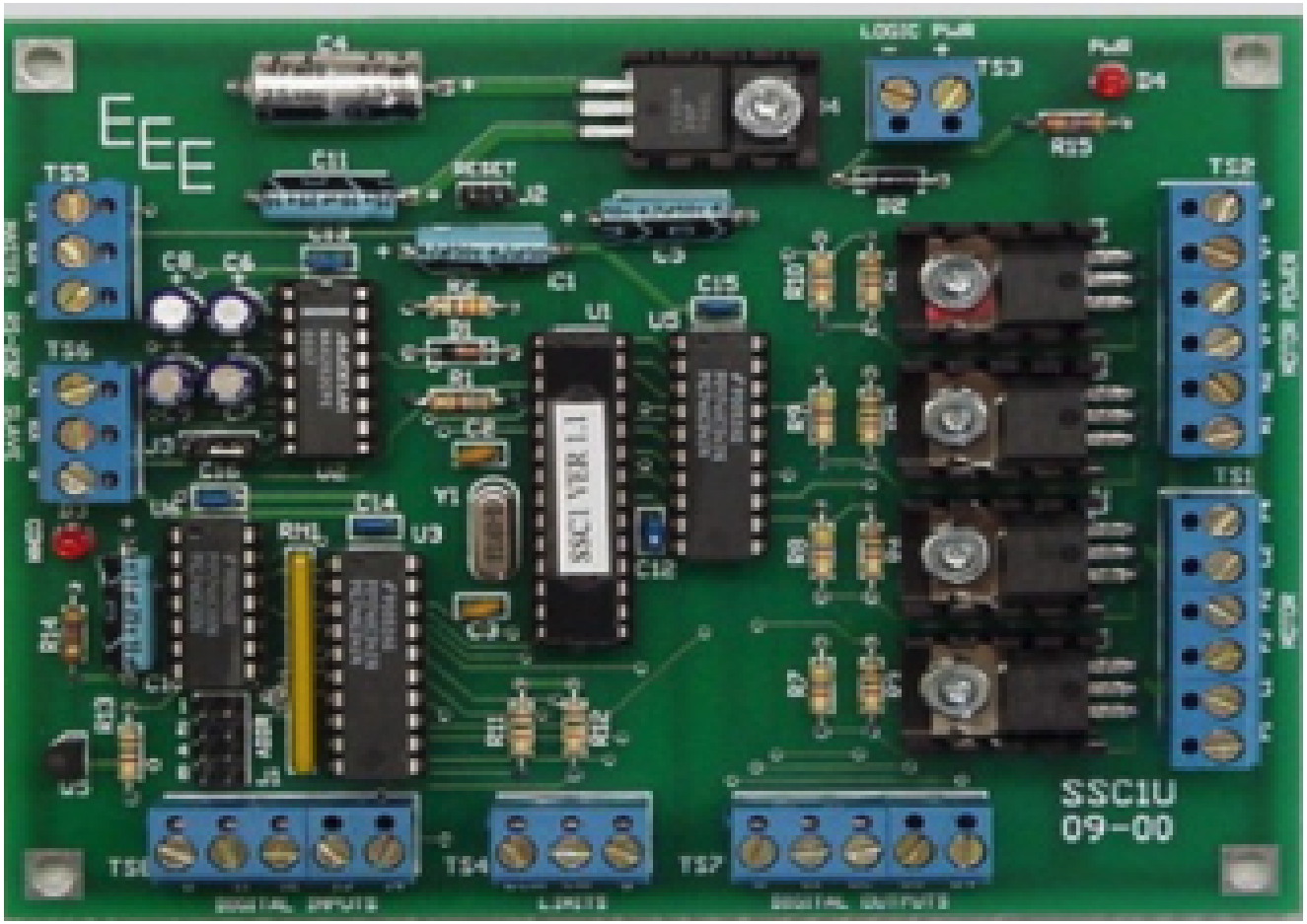




What is the shortest route from Minhang campus to Xuhui campus?



Electronic Design Automation



Design automation and verification

What do those problems have in common?

- Generally large space of solutions
- Goal well-defined and easily specified
- Solution can be sequentially built from initial state
- They can be formalized as a search in a state-space graph

- Solving problem = searching for a path in this graph
  - start from initial node
  - an edge in the graph = possible action in the problem
  - end when reaching an end node
- Solution = path to an end node in this graph

Principle for Solving Reasoning Problems

- Idea: Reasoning as search
- Problem formalized as state-space graph
  - Graph $= (V, E)$ and valued graph $= (V, E, c)$ with $c : E \to \mathbb{R}$
  - V set of nodes/vertices/**states**/configurations
  - E set of edges
    - non-directed edge denoted $\{v, v'\}$
    - directed edge denoted (v, v')
  - $Path = \left(v_1, v_2, \ldots, v_k\right)$ where v_i, v_{i+1} define an edge
- Different algorithms to explore the graph
- Graph explored by *tree search* algorithms

Principle for Solving Reasoning Problems

- Idea: Reasoning as search
- Problem formalized as state-space graph
  - Graph $= (V, E)$ and valued graph $= (V, E, c)$ with $c : E \to \mathbb{R}$
  - V set of nodes/vertices/**states**/configurations
  - E set of edges
    - non-directed edge denoted $\{v, v'\}$
    - directed edge denoted (v, v')
  - $Path = \left(v_1, v_2, \ldots, v_k\right)$ where v_i, v_{i+1} define an edge
- Different algorithms to explore the graph
- Graph explored by *tree search* algorithms

- State-space graph and *search tree* only implicitly represented in memory

How to formalize a reasoning problem?

- What are the possible states?
- What are the possible actions for each state?
- What is the initial state?
- What is the end state? (possibly non unique)

How to formalize a reasoning problem?

- What are the possible states? Often, vector of variables
- What are the possible actions for each state?
- What is the initial state?
- What is the end state? (possibly non unique)

How to formalize a reasoning problem?

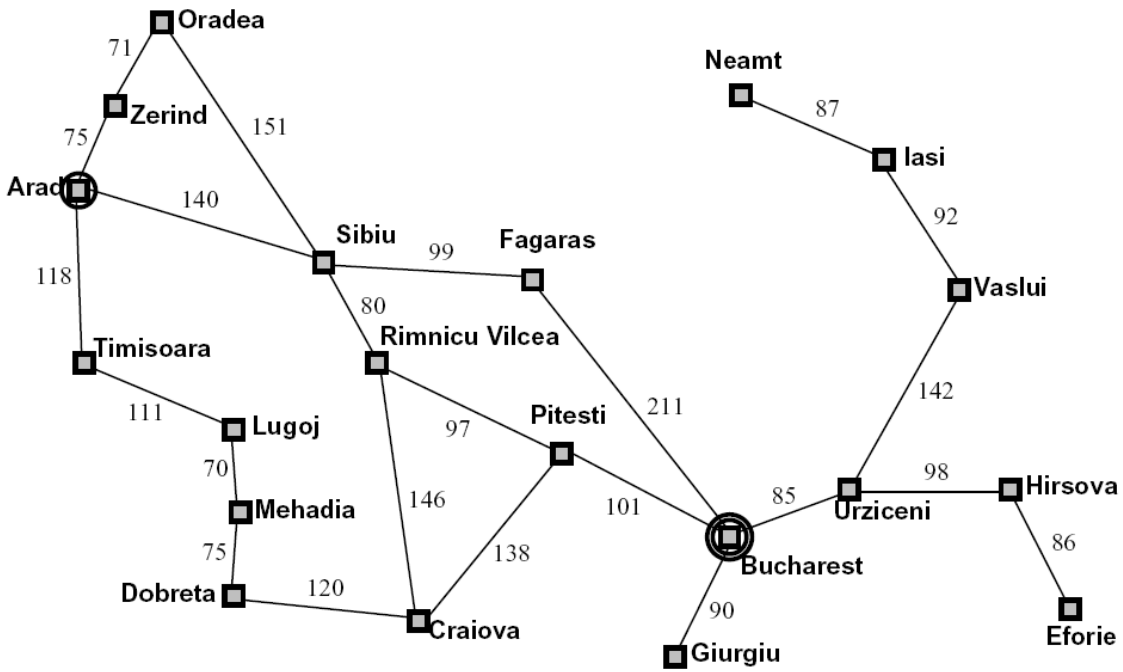
- What are the possible states? Often, vector of variables
- What are the possible actions for each state?
- What is the initial state?
- What is the end state? (possibly non unique)

Note:

- The state-space graph generally huge, so not completely represented in memory
- Edges or actions represented by successor function
- End state expressed in
  - extension: list of states
  - comprehension: e.g., logic formula or constraints

Example: Traveling in Romania

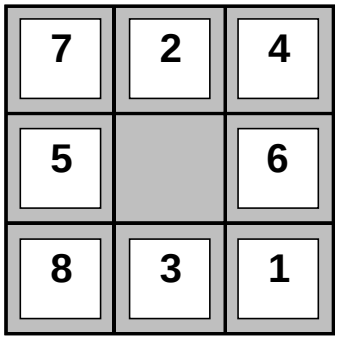
- State space
  - Cities
- Successor function
  - Roads: Go to adjacent city with cost = distance
- Start state
  - Arad
- Goal test:
  - is state = Bucharest?
- Solution?



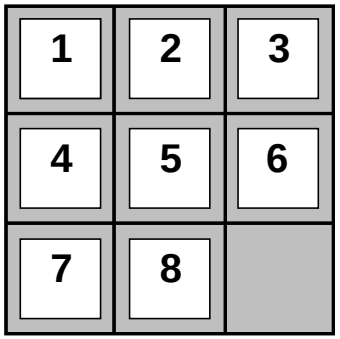
Definition of Graph Search Problem

- A Graph Search Problem:
  1. Set of states S
  2. Set of actions A
  3. Successor function $succ(s, a) \in S$
  4. Cost over actions $c(s, a, s') \geq 0$
  5. Initial state s_0
  6. Set of goal (or end) states $G \subset S$
- Example:
  1. Romanian cities
  2. Road between cities
  3. Given by the road network
  4. $c(s, a, s')$ = distance
  5. s_0 = Arad
  6. $G = \{\text{Bucharest}\}$
- Not all actions are available in every state
  - Function *succ* can be described by preconditions and effects
- Path cost = sum of cost over its actions
- Feasible solution = sequence of actions from initial state to a goal
- Optimal solution = solution with minimal cost

The 8-Puzzle

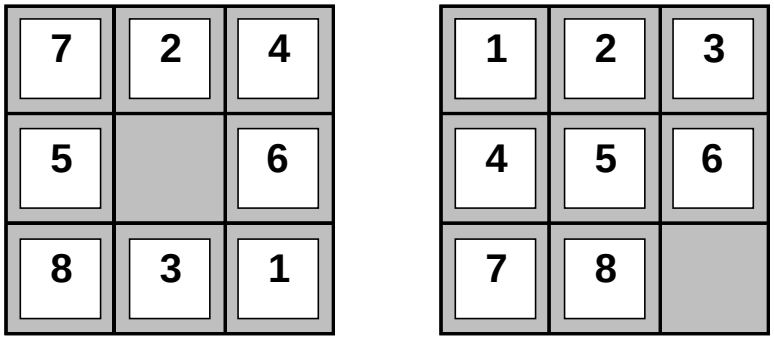


Start State



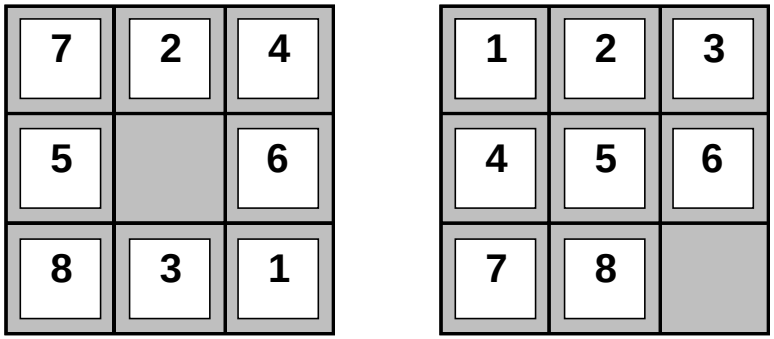
Goal State

The 8-Puzzle



1. State = Integer location of tiles
2. Action = Move of blank left, right, up or down
3. Action cost = 1
4. Initial state given
5. Unique goal state given

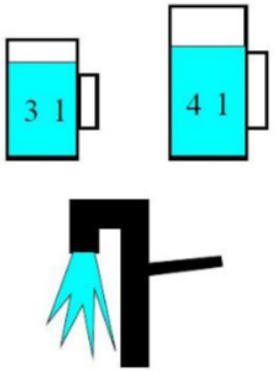
The 8-Puzzle



1. State = Integer location of tiles
2. Action = Move of blank left, right, up or down
3. Action cost = 1
4. Initial state given
5. Unique goal state given

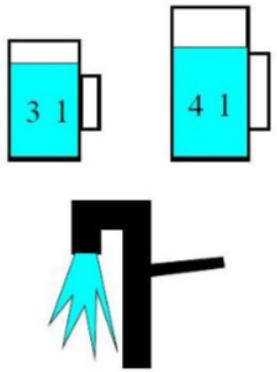
- Can be generalized to $n^2 - 1$ puzzle for $n \geq 3$
- Optimal solution of this puzzle family is NP-hard

Water Jug Problem



- Problem: We have one 3L-jug and one 4L-jug. We want to put exactly 2 liters of water in the 4L-jug.

Water Jug Problem



- Problem: We have one 3L-jug and one 4L-jug. We want to put exactly 2 liters of water in the 4L-jug.

1. State = Content of jugs (J_3, J_4)
2. Action = Fill a jug, empty a jug, what else?
3. Action cost = 1
4. Initial state $(0, 0)$
5. Unique goal state $(\_, 2)$

Water Jug Problem

F4: fill jug4 from the pump.

precond: $J_4 < 4$ **effect:** $J_4' = 4$

E4: empty jug4 on the ground.

precond: $J_4 > 0$ **effect:** $J_4' = 0$

E4-3: pour water from jug4 into jug3 until jug3 is full.

precond: $J_3 < 3,\ J_4 \geq 3 - J_3$ **effect:** $J_3' = 3,\ J_4' = J_4 - (3 - J_3)$

P3-4: pour water from jug3 into jug4 until jug4 is full.

precond: $J_4 < 4,\ J_3 \geq 4 - J_4$ **effect:** $J_4' = 4,\ J_3' = J_3 - (4 - J_4)$

E3-4: pour water from jug3 into jug4 until jug3 is empty.

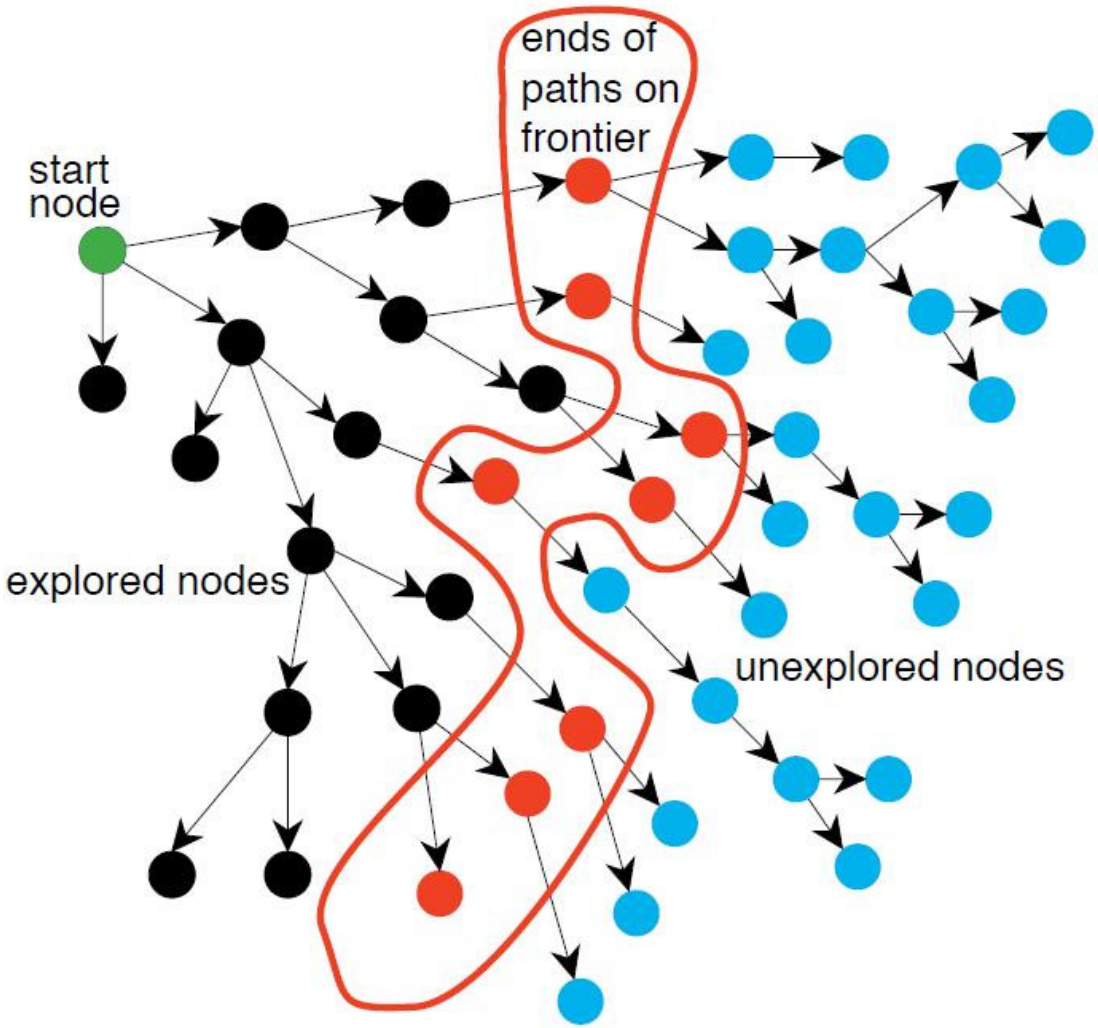
precond: $J_3 + J_4 < 4,\ J_3 > 0$ **effect:** $J_4' = J_3 + J_4,\ J_3' = 0$

Search Tree

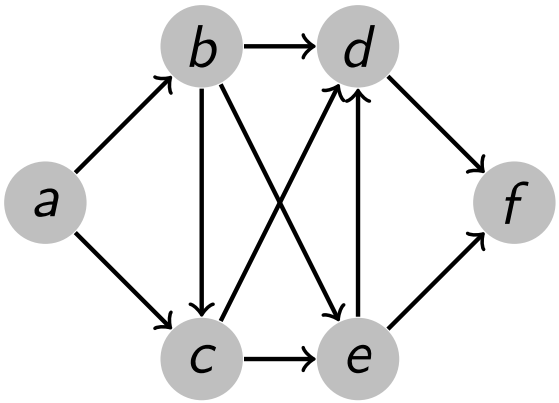
The set of all possible paths of a graph from an initial state can be represented as a tree:

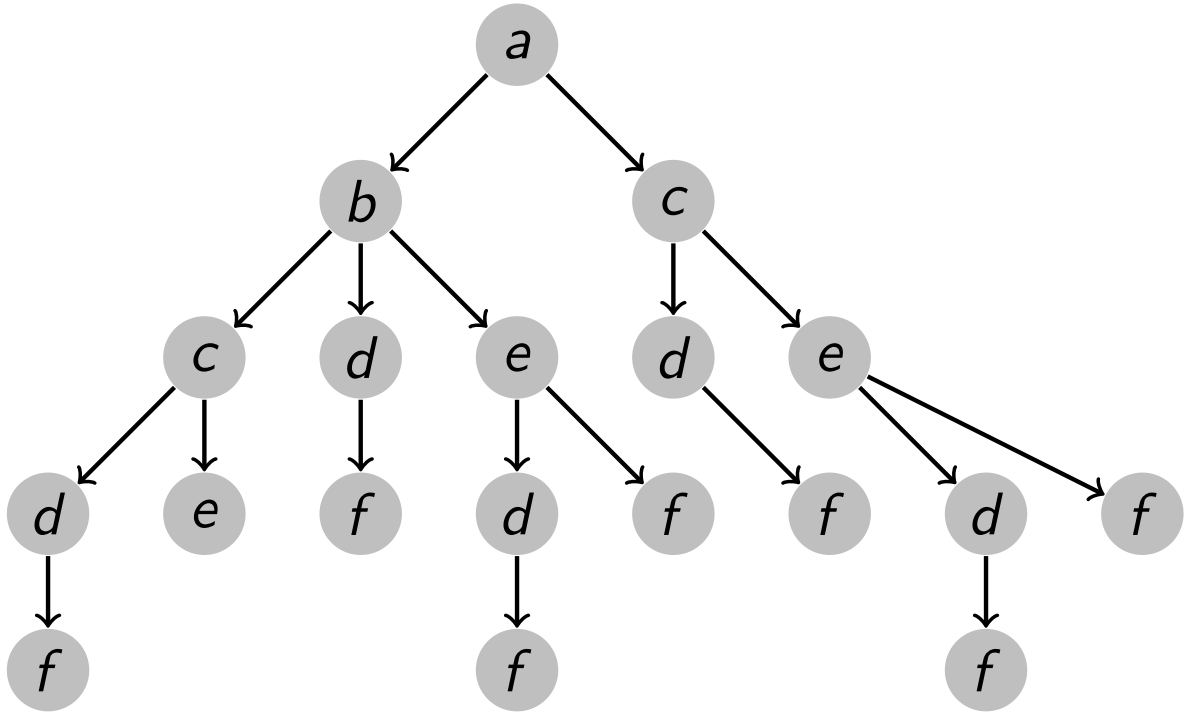
- *Tree* = directed acyclic graph where each node has at most one predecessor (=*parent*)
- *Root* = a node with no parents
- *Leaf* = a node with no children
- *Branching factor* of a node = number of its children

Example of a Search Tree



Relationship Between State Graph and Search Tree





General Tree Search Algorithm

Basic idea: explore state space graph by generating successors of already-visited states (i.e., expanding states)

Algorithm 1: TreeSearch

Data: problem, strategy
Result: solution or failure

```
1 initialize search tree using initial state of problem;
2 while there are candidates for expansion do
3     choose a leaf node for expansion according to strategy;
4     if node corresponds to goal state then
5         return corresponding solution;
6     else
7         expand node and add resulting nodes to search tree;
8 return failure;
```

Search Strategies

- A search strategy defines the order of node expansion.
- Strategies are evaluated along the following dimensions:
  - Completeness: does it always find a solution if one exists?
  - Time complexity: number of nodes generated
  - Space complexity: maximum number of nodes in memory
  - Optimality: does it always find a least-cost solution?

Search Strategies

- A search strategy defines the order of node expansion.
- Strategies are evaluated along the following dimensions:
  - Completeness: does it always find a solution if one exists?
  - Time complexity: number of nodes generated
  - Space complexity: maximum number of nodes in memory
  - Optimality: does it always find a least-cost solution?
- Time/space complexity measured in terms of
  - b: maximum branching factor of search tree
  - d: depth of least-cost solution
  - m: maximum depth of state space (may be ∞)
  - l: depth limit (for depth-limited search)
  - c^*: cost of optimal solution (for uniform-cost search)
  - $\epsilon > 0$: minimum step cost (for uniform-cost search)

Two Types of Search

- Uninformed or blind search
  - No extra information apart from the given graph
  - Takes into account only the path already explored
- Informed search
  - Uses extra information (e.g., heuristics, constraints)
  - Guided by this information

Search Algorithms in this Course

Covered:

- Uninformed Search
  - Depth-First Search
  - Breadth-First Search
  - Uniform-Cost Search
  - Depth-Limited Search
  - Iterative Deepening Search
- Informed Search
  - Greedy Best-First Search
  - A* Search
  - Simple Memory-Bounded A*
- Stochastic Search
  - Simulated Annealing
  - Monte Carlo Tree Search

Not covered:

- Constraint Satisfaction Problems
  - Forward Checking
  - Constraint Propagation
- Meta-heuristics
  - Evolution Strategies (genetic algorithm, CMA-ES...)
  - Particle Swarm Optimization
- Stochastic Search
  - Cross-Entropy Method