NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

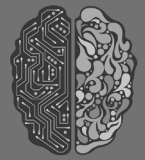
# Introduction to Data Science and Artificial Intelligence

## State-of-the-Art in DS&AI


Assoc Prof Bo AN

*Research area*: artificial intelligence, computational game theory, optimization
www.ntu.edu.sg/home/boan
*Email*: boan@ntu.edu.sg
*Office*: N4-02b-55


# Lesson Outline

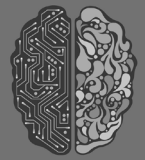


- A brief history of AI
- The state of the art

# What is Artificial Intelligence?

**AI** is intelligence demonstrated by machines, in contrast to the **natural intelligence** (**NI**) displayed by humans and other animals.

# Birth of Artificial Intelligence



"We can only see a short distance ahead, but we can see plenty there that needs to be done."

*Alan Turing*

*Father of Modern Computer Science*

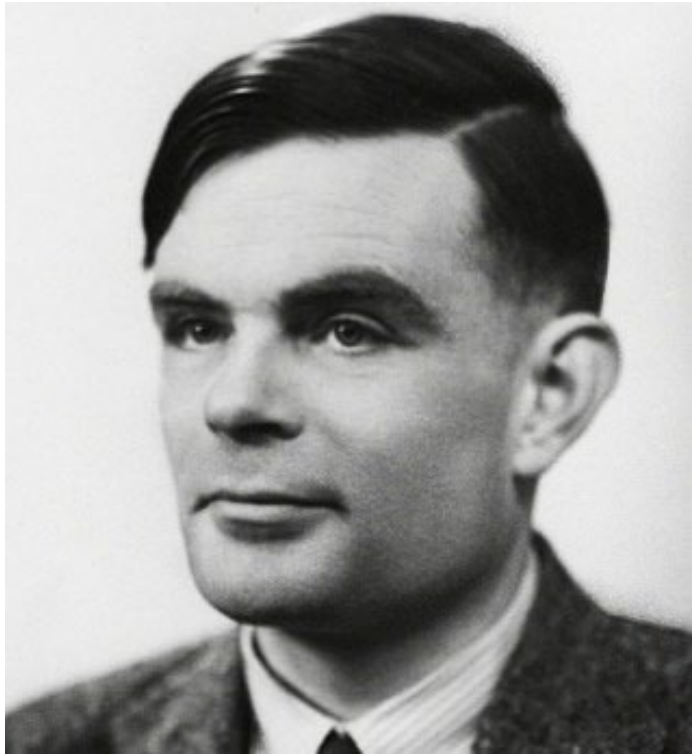


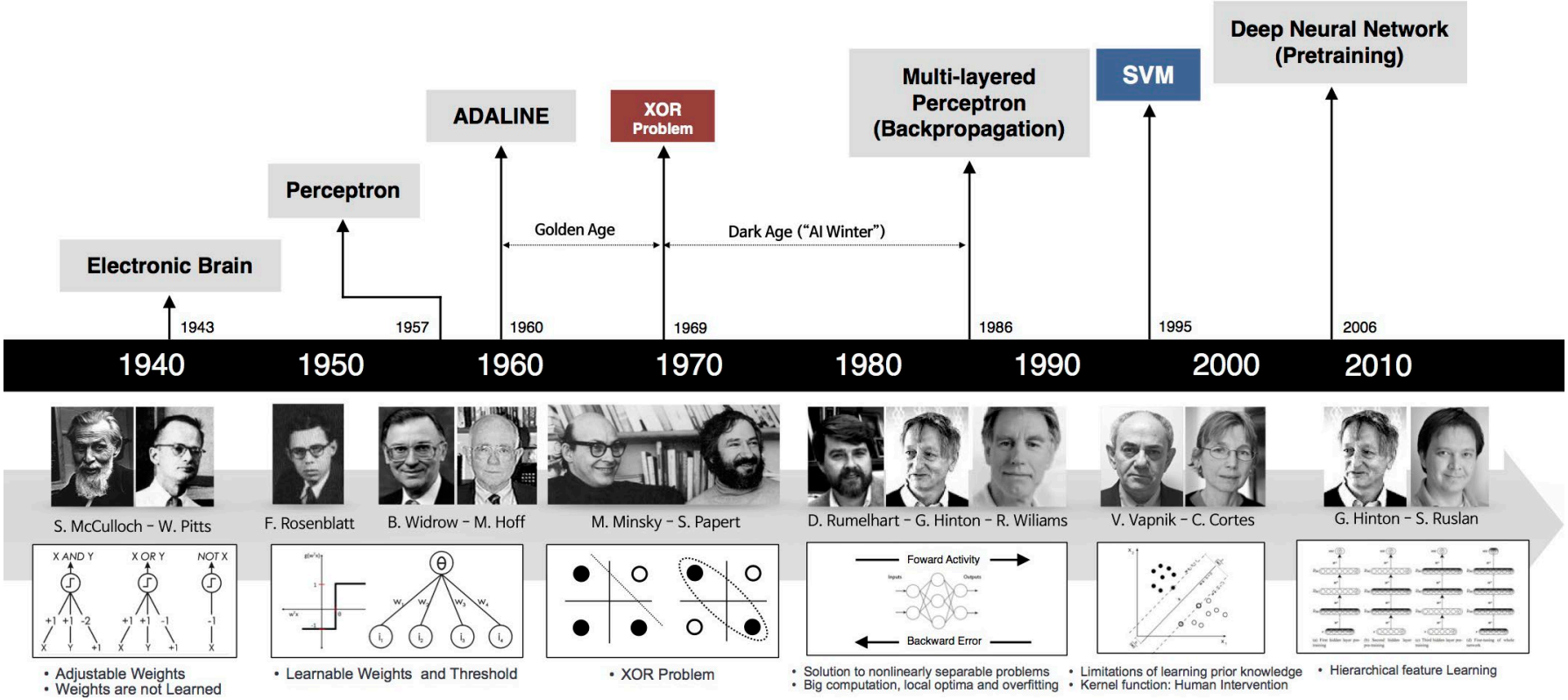
Image source: https://www.flickr.com/photos/23925401@N06/23022750663

# Founding Fathers of AI

## 1956 Dartmouth Conference


Image source: https://www.scienceabc.com/innovation/what-is-artificial-intelligence.html

# Timeline of AI Development

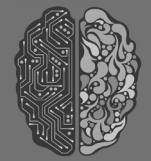


# Views of AI

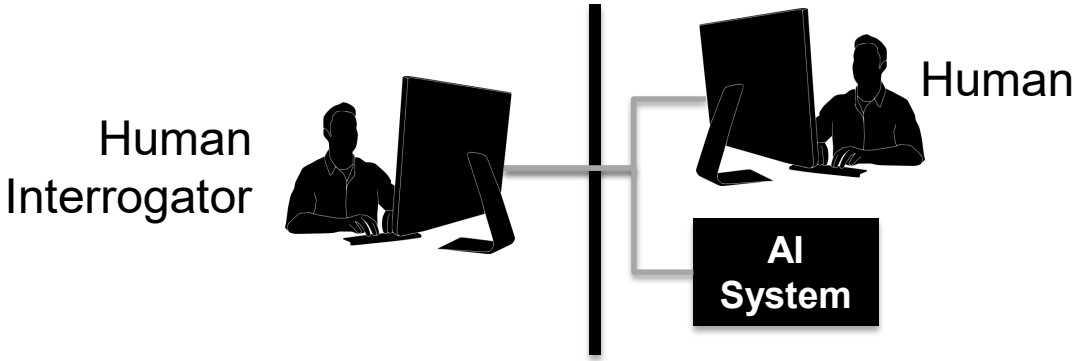
Views of AI fall into four categories:

1. Thinking Humanly
2. Acting Humanly
3. Thinking Rationally
4. Acting Rationally

# Acting Humanly: Turing Test



- Turing (1950) "Computing machinery and intelligence"
- Operational test for intelligent behavior: the Imitation Game

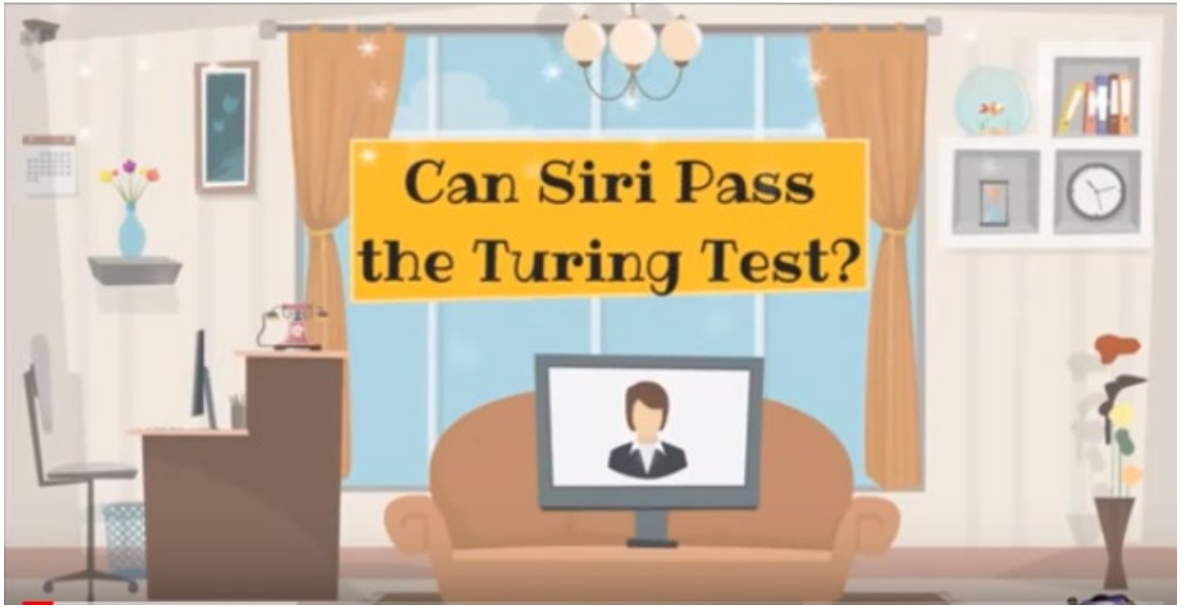



- Suggested major components of AI: knowledge, reasoning, language understanding, learning

Image source: https://svgsilh.com/image/2763393.html

# Can Siri Pass The Turing Test?

https://www.youtube.com/watch?v=0qcHhBSbyVw



<Video from previous slide will be inserted in learning sequence for students to watch>

# Thinking Humanly: Cognitive Modeling

- 1960s "cognitive revolution": information-processing psychology
- Requires scientific theories of internal activities of the brain
- Both approaches (Cognitive Science and Cognitive Neuroscience) are now distinct from AI

# Thinking Rationally: "Laws of Thought"

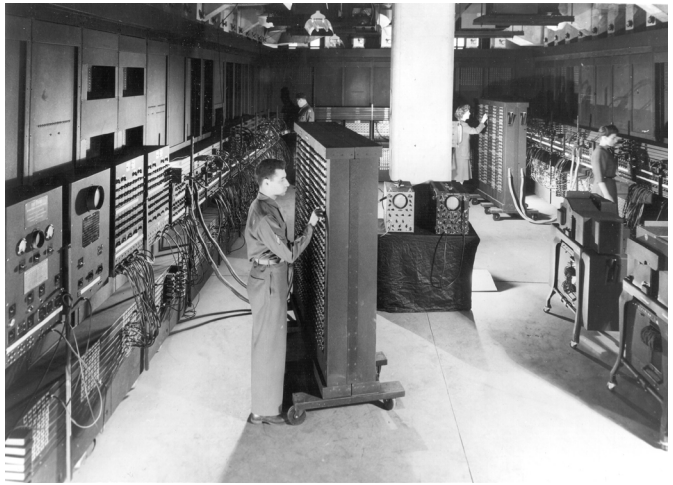
- Aristotle: what are correct arguments/thought processes?
  - Several Greek schools developed various forms of logic: notation and rules of derivation for thoughts; may or may not have proceeded to the idea of mechanization
- Direct line through mathematics and philosophy to modern AI
- Problems:
  - Not all intelligent behavior is mediated by logical deliberation
  - What is the purpose of thinking? What thoughts should I have?

# Acting Rationally: Rational Agents

- Rational behavior: doing the right thing
- The right thing: that which is expected to maximise goal achievement, given the available information
- This course is about designing rational agents
- For any given class of environments and tasks, we seek the agent(s) with the best performance

# Computer Chess

## Garry Kasparov VS Deep Blue



VS


ENIAC 1946

Deep Blue 1997

Image Sources:
https://www.flickr.com/photos/gageskidmore/37137186253
https://en.wikipedia.org/wiki/Deep_Blue_(chess_computer)#/media/File:Deep_Blue.jpg

# Robot Soccer

TED talk by
Peter Stone
from UT Austin

http://www.youtube.com/watch?v=FXhw0_-iKwQ

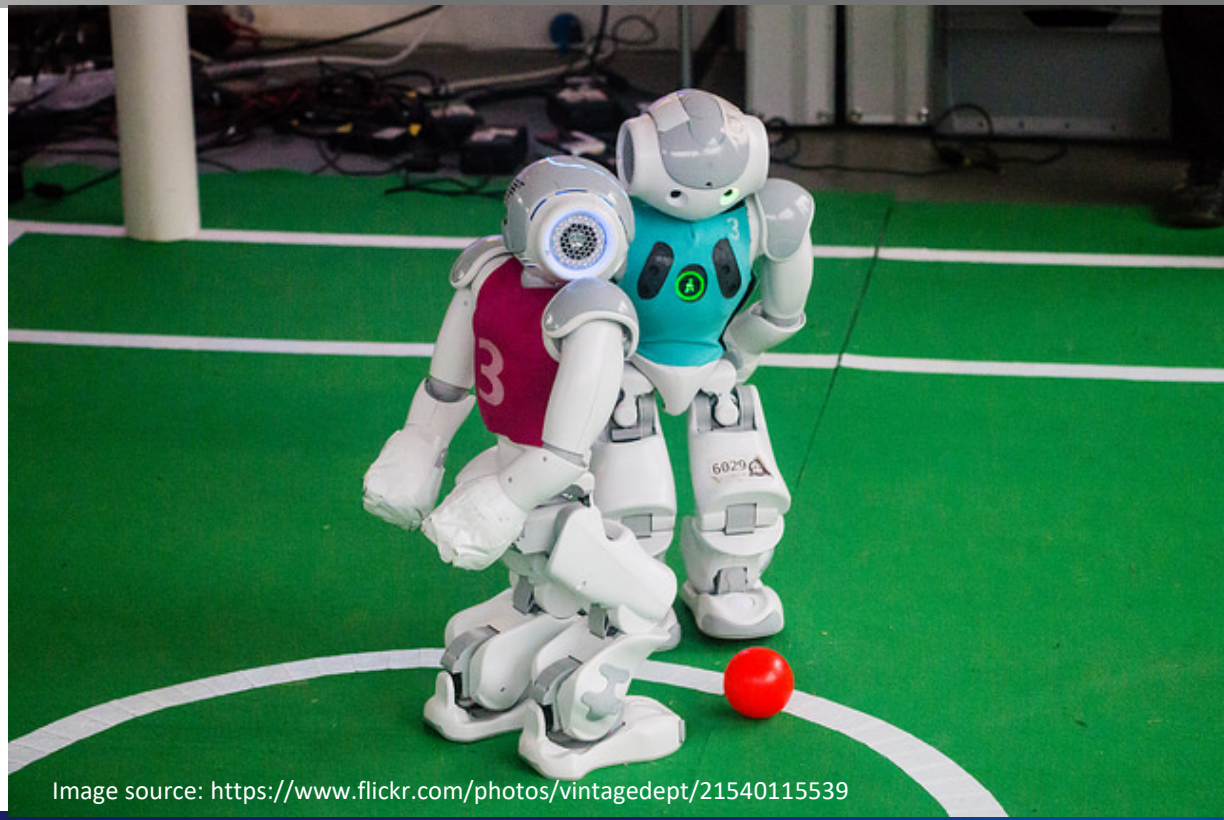



Image source: https://www.flickr.com/photos/vintagedept/21540115539

<Video from previous slide will be inserted in learning sequence for students to watch>

# Game Show


Image source: https://commons.wikimedia.org/wiki/File:IBM_Watson_w_Jeopardy.jpg

IBM's Watson Destroys Humans in Jeopardy

https://www.youtube.com/watch?v=P18EdAKuC1U

<Video from previous slide will be inserted in learning sequence for students to watch>

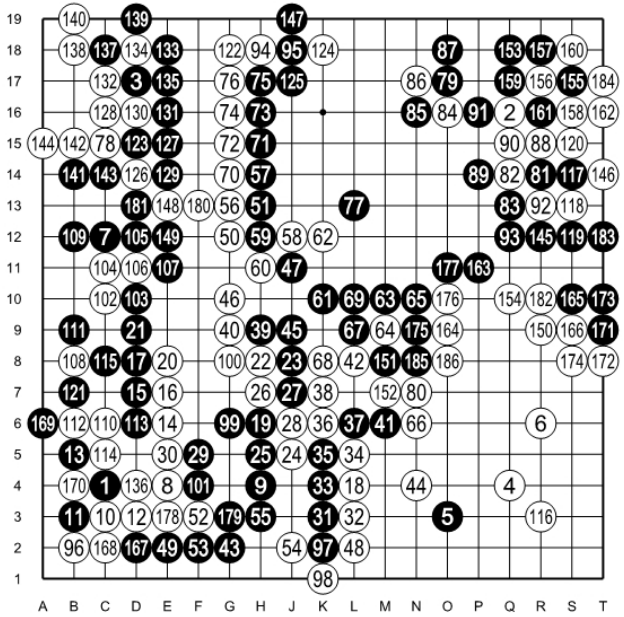
# AlphaGo vs World Champions

**March 9 – 15, 2016 (Lee Sedol)**

- Time limit: 2 hours
- Venue: Seoul, Four Seasons Hotel
- AlphaGo Wins (4:1)

**May 23 – 27, 2017 (Ke Jie)**

- Venue: Wuzhen, China
- AlphaGo Wins (3:0)



Lee Sedol (B) vs AlphaGo (W) - Game 1

Image source: https://commons.wikimedia.org/wiki/File:Lee_Sedol_(B)_vs_AlphaGo_(W)_-_Game_1_-_BW.jpg

# Libratus vs World Champions

The first AI to defeat top human poker players

**January 11 to 31, 2017**

- Venue:  Pittsburgh
- 120,000 hands

- Has nothing to do with deep learning
- Algorithms for solving large scale games

# Google Driverless Car

TED talk by Sebastian Thrun from Stanford

https://www.youtube.com/watch?v=bp9KBrH8H04

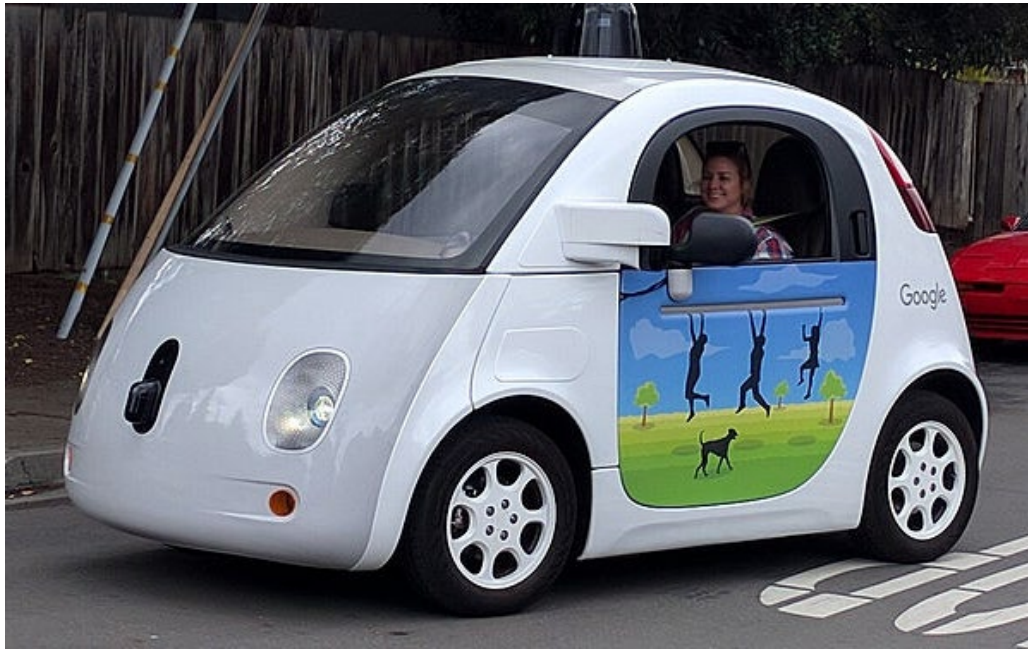



Image source: https://commons.wikimedia.org/wiki/File:Google_driverless_car_at_intersection.gk.jpg

<Video from previous slide will be inserted in learning sequence for students to watch>

# Google Robot Dog

https://www.youtube.com/watch?v=4NzcB6TMzjw

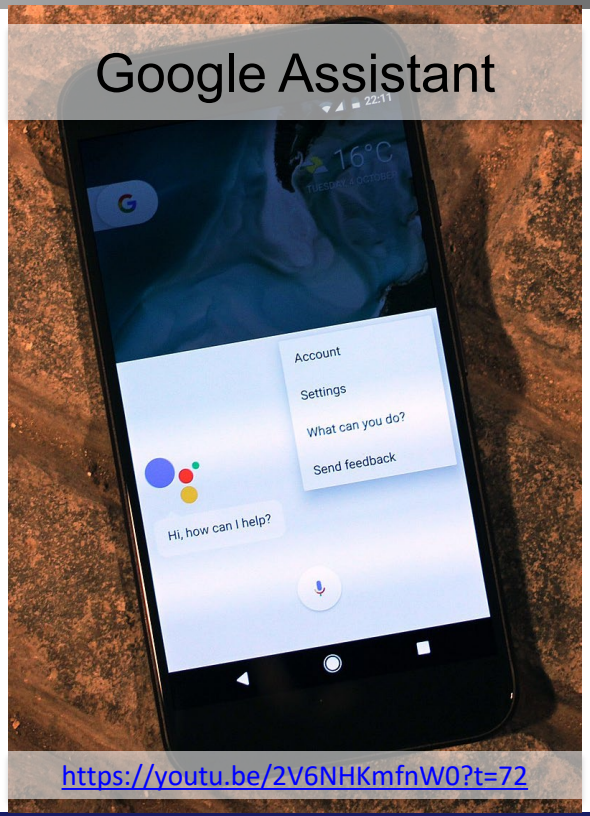
Image source: https://commons.wikimedia.org/wiki/File:Bio-inspired_Big_Dog_quadruped_robot_is_being_developed_as_a_mule_that_can_traverse_difficult_terrain.tiff

# Natural Language Processing

## Mi Box Voice Assistant

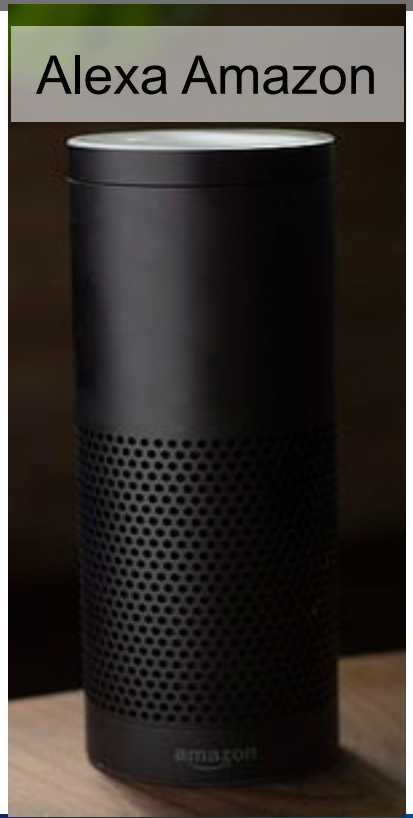
https://youtu.be/S38ZJELrMmQ?t=16

## Google Assistant




https://youtu.be/2V6NHKmfnW0?t=72

## Alexa Amazon



# Manufacturing

Tesla
https://www.youtube.com/watch?v=8_IfxPI5ObM

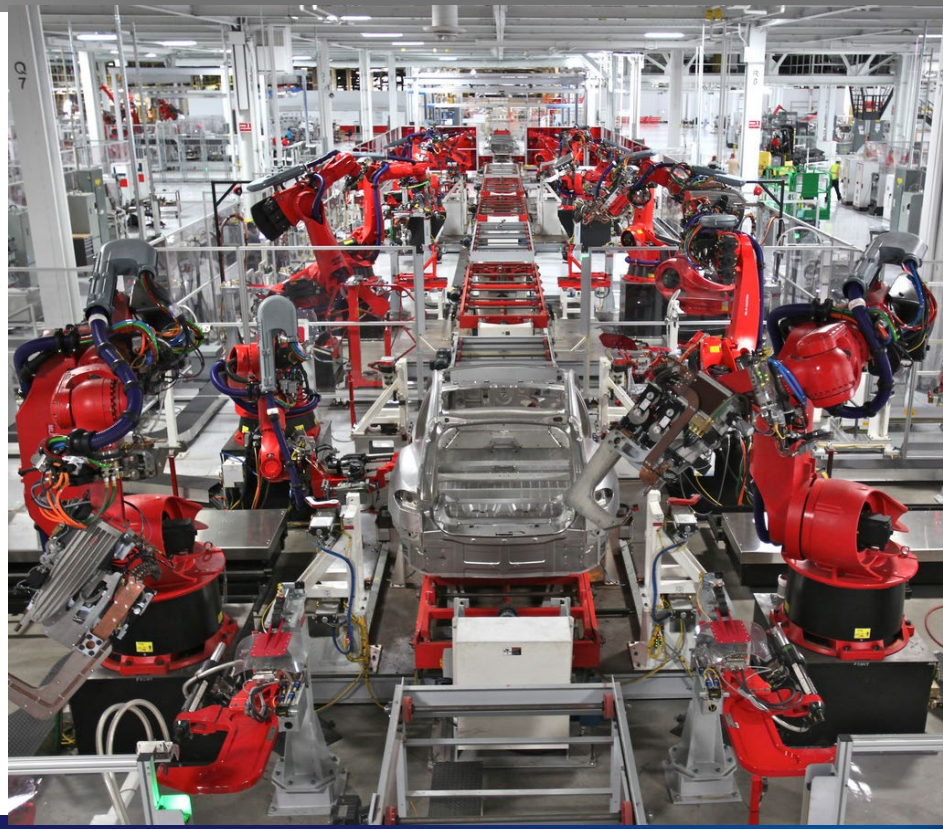


Image source: https://www.flickr.com/photos/jurvetson/7408451314

<Video from previous slide will be inserted in learning sequence for students to watch>

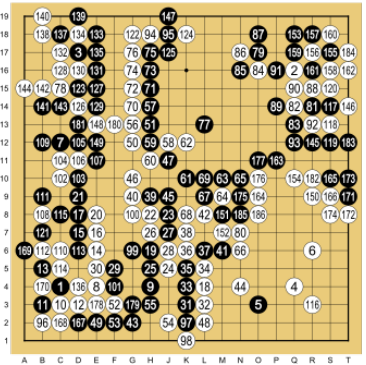
# Artificial Intelligence for Finance


Source: World Economic Forum, 2015

# AI for Complex Interactions

## Recent AI breakthrough

IMAGENET



# What's Next?

## For AI Complex Interactions

- Stochastic, open environment
- Multiple players
- Sequential decision, online
- Strategic (selfish) behaviour
- Distributed optimisation

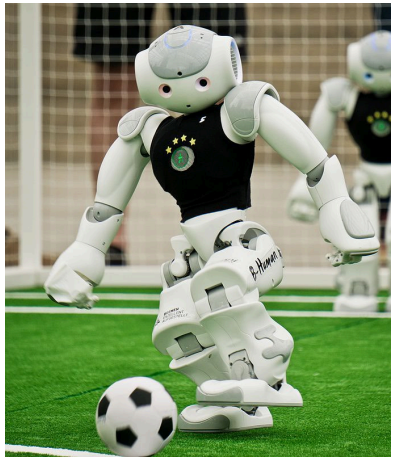


Image sources:

- https://en.wikipedia.org/wiki/RoboCup_Standard_Platform_League#/media/File:SPL_Team_B-Human,_RoboCup_2016.jpg
- https://en.wikipedia.org/wiki/Massively_multiplayer_online_role-playing_game#/media/File:Ryzom_screenshot_4.png
- https://pixabay.com/en/network-iot-internet-of-things-782707/
- https://www.flickr.com/photos/thebetterday4u/37556363291
- https://en.wikipedia.org/wiki/Google_Assistant#/media/File:Android_Assistant_on_the_Google_Pixel_XL_smartphone_(29526761674).jpg
- https://www.flickr.com/photos/stockcatalog/40095307924
- https://pixabay.com/en/woman-people-coffee-portrait-3083379/
- https://pixabay.com/en/microphone-audio-micro-recording-2104091/
- https://www.flickr.com/photos/gleonhard/34046646705
- https://commons.wikimedia.org/wiki/File:Lee_Sedol_(B)_vs_AlphaGo_(W)_-_Game_1.svg

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

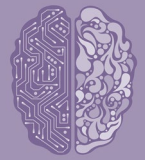
# Introduction to Data Science and Artificial Intelligence

## Intelligent Agents

Assoc Prof Bo AN

*Research area*: artificial intelligence, computational game theory, optimization
www.ntu.edu.sg/home/boan
*Email*: boan@ntu.edu.sg
*Office*: N4-02b-55

# Lesson Outline

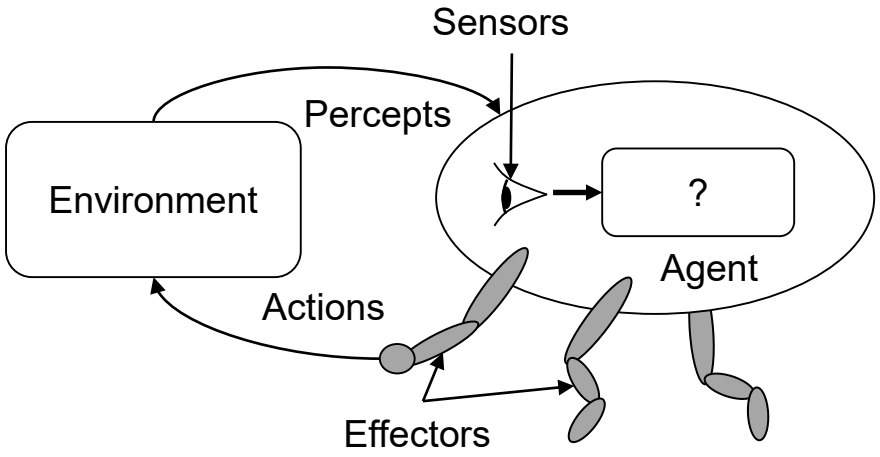


- What is an agent?
- How can one describe the task/problem for the agent?
- What are the properties of the task environment for the agent?
- What are the different basic kinds of agents for intelligent systems?
- Problem formulation

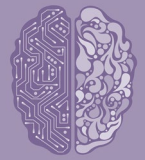
# Agent

An agent is an entity that

- Perceives through sensors (e.g. eyes, ears, cameras, infrared range sensors)
- Acts through effectors (e.g. hands, legs, motors)



# Rational Agents



- A rational agent is one that does the right thing
- Rational action: action that maximises the expected value of an objective performance measure given the percept sequence to date
- Rationality depends on
  - performance measure
  - everything that the agent has perceived so far
  - built-in knowledge about the environment
  - actions that can be performed

# Example: Google X2: Driverless Taxi

- **Percepts:** video, speed, acceleration, engine status, GPS, radar, …
- **Actions:** steer, accelerate, brake, horn, display, …
- **Goals:** safety, reach destination, maximise profits, obey laws, passenger comfort,…
- **Environment:** Singapore urban streets, highways, traffic, pedestrians, weather, customers, …

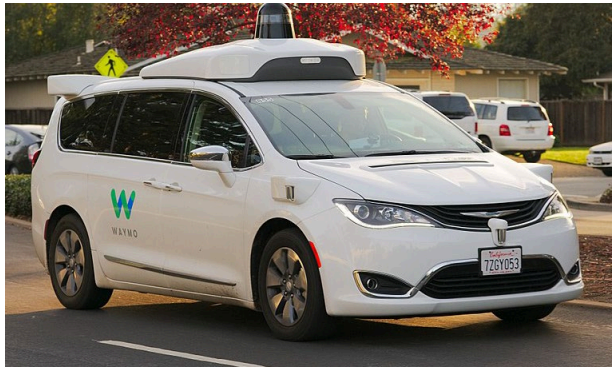



Image source: https://en.wikipedia.org/wiki/Waymo#/media/File:Waymo_Chrysler_Pacifica_in_Los_Altos,_2017.jpg

# Example: Medical Diagnosis System

- **Percepts**: symptoms, findings, patient's answers, …
- **Actions:** questions, medical tests, treatments,…
- **Goals:** healthy patient, faster recovery, minimise costs, …
- **Environment:** Patient, hospital, clinic, …

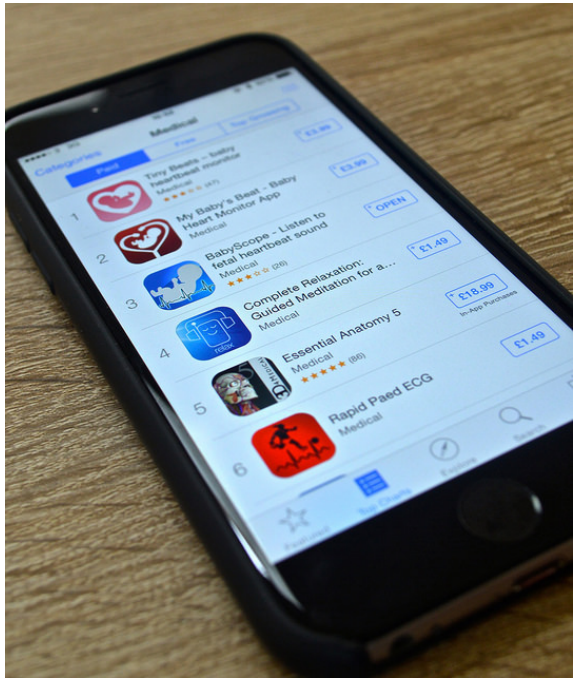



Image source: https://www.flickr.com/photos/134647712@N07/20008817459

# Autonomous Agents

- Do **not** rely entirely on built-in knowledge about the environment (i.e. not entirely pre-programmed)
- Otherwise,
  - The agent will **only** operates successfully when the built-in knowledge are all correct
- Adapt to the environments through experience

**Example: Driverless Car**

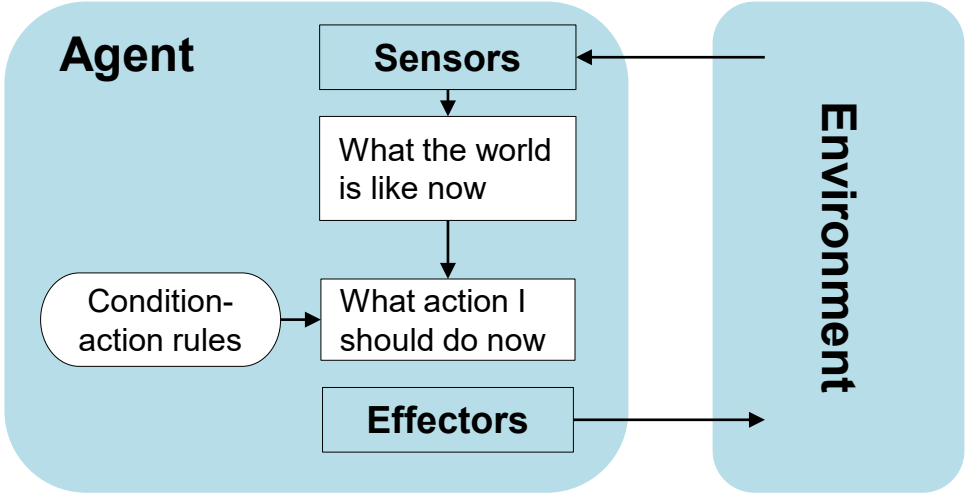
- Learn to drive in driving center
- Drive at NTU
- Drive on public roads
- Drive in highways
- Drive in City Hall

# Simple Reflex Agents

**Example**

**If** car-in-front-is-braking **then** initiate-braking

1. Find the rule whose condition matches the current situation (as defined by the percept)
2. Perform the action associated with that rule

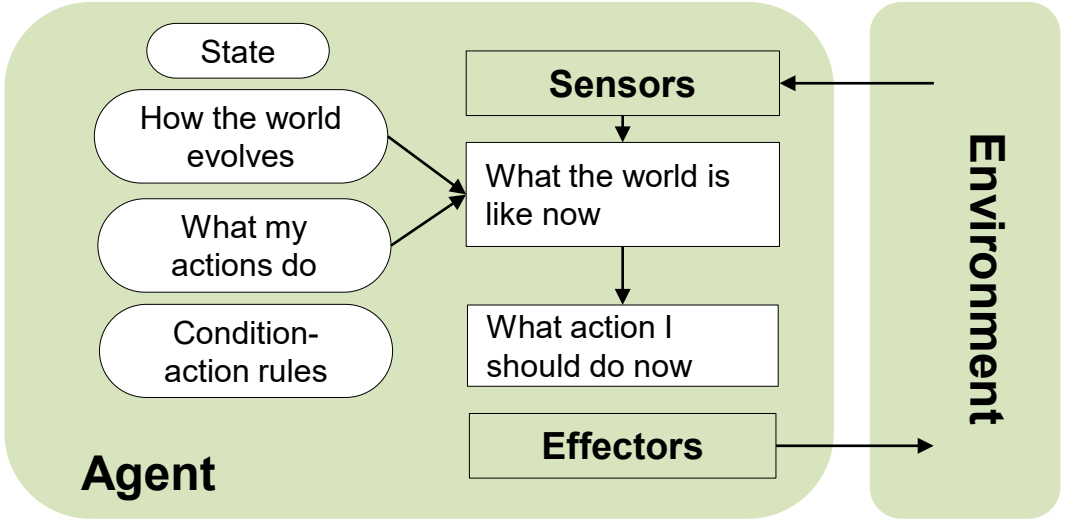


# Reflex Agents with State

**Example**

**If** yesterday-at-NTU **and** no-traffic-jam-now **then** go-Orchard

1. Find the rule whose condition matches the current situation (as defined by the percept and the stored internal state)
2. Perform the action associated with that rule

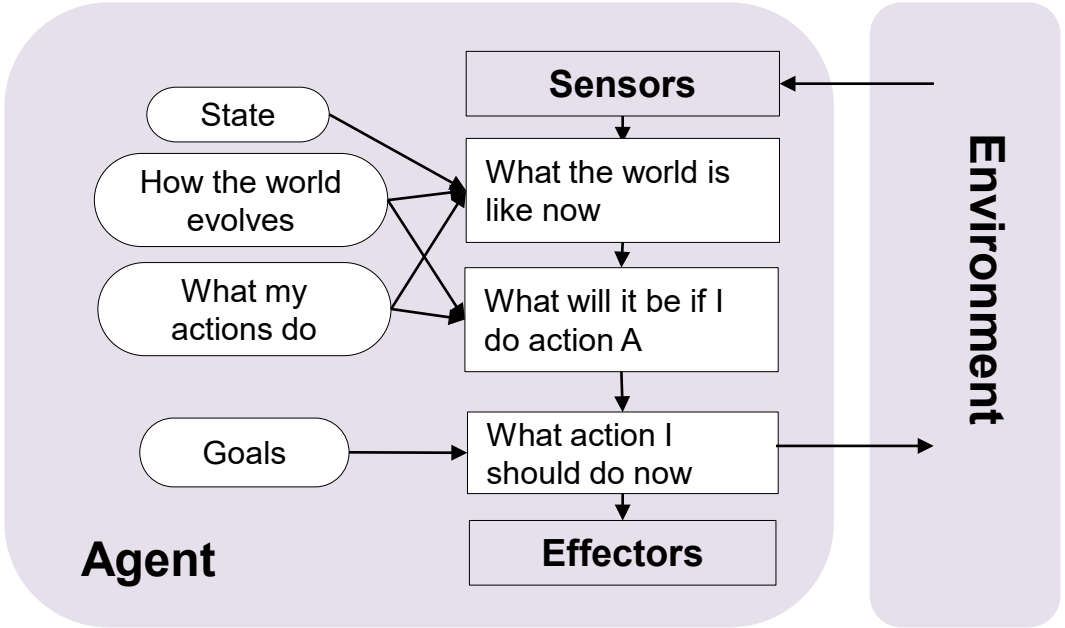


# Goal-Based Agents

Needs some sort of goal information

**Example:**
**Driverless Taxi**

- At a junction (known state), should I go left, right, or straight on?
- Reach Orchard (Destination)?

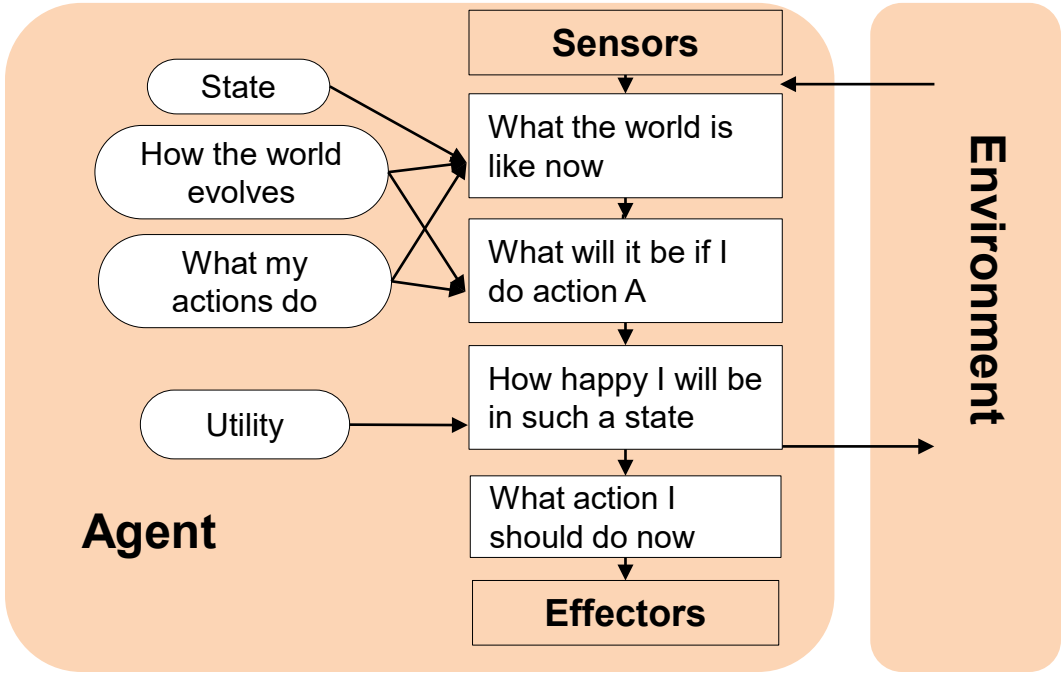


# Utility-Based Agents

- There may be many action sequences that can achieve the same goal, which action sequence should it take?
- How happy will agent be if it attains a certain state? → Utility

**Example: Driverless Taxi**

- Go to Orchard (Destination) via PIE? AYE?
- Which one charges lower fare?



Question to think:

- Why is problem formulation (e.g., characterize percept, goals etc) important?

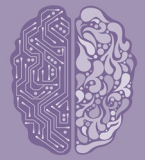
# Types of Environment

| | |
|---|---|
| Accessible (vs inaccessible) | Agent's sensory apparatus gives it access to the complete state of the environment |
| Deterministic (vs nondeterministic) | The next state of the environment is completely determined by the current state and the actions selected by the agent |
| Episodic (vs Sequential) | Each episode is not affected by the previous taken actions |
| Static (vs dynamic) | Environment does not change while an agent is deliberating |
| Discrete (vs continuous) | A limited number of distinct percepts and actions |

# Example: Driverless Taxi

| | |
|---|---|
| Accessible? | No. Some traffic information on road is missing |
| Deterministic? | No. Some cars in front may turn right suddenly |
| Episodic? | No. The current action is based on previous driving actions |
| Static? | No. When the taxi moves, Other cars are moving as well |
| Discrete? | No. Speed, Distance, Fuel consumption are in real domains |

# Example: Chess



| | |
|---|---|
| Accessible? | Yes. All positions in chessboard can be observed |
| Deterministic? | Yes. The outcome of each movement can be determined |
| Episodic? | No. The action depends on previous movements |
| Static? | Yes. When there is no clock, when are you considering the next step, the opponent can't move; Semi. When there is a clock, and time is up, you will give up the movement |
| Discrete? | Yes. All positions and movements are in discrete domains |

# Example: Minesweeper

| | |
|---|---|
| Accessible? | No. Mines are hidden |
| Deterministic? | No. Mines are randomly assigned in different positions |
| Episodic? | No. The action is based on previous outcomes |
| Static? | Yes. When are you considering the next step, no changes in environment |
| Discrete? | Yes. All positions and movements are in discrete domains |

# Slot machines - One-Armed Bandit

| | |
|---|---|
| Accessible? | No |
| Deterministic? | No |
| Episodic? | Yes |
| Static? | Yes |

Image source: https://commons.wikimedia.org/wiki/File:Las_Vegas_slot_machines.jpg

Question to think:

- Can you find another scenario whose environment is episodic?

# Design of Problem-Solving Agent

Idea

- Systematically considers the expected outcomes of different possible sequences of actions that lead to states of known value
- Choose the best one
    - shortest journey from A to B?
    - most cost effective journey from A to B?

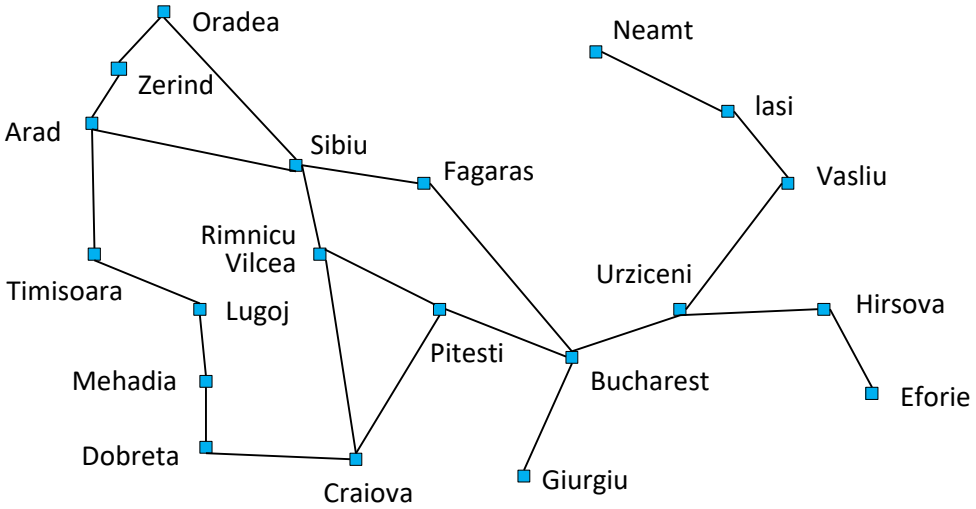
# Design of Problem-Solving Agent

Steps

1. Goal formulation
2. Problem formulation
3. Search process
    - No knowledge → uninformed search
    - Knowledge → informed search
4. Action execution (follow the recommended route)

# Goal-based Agent: Example

On holiday in Romania




- Currently in Arad (Initial state). Flight leaves tomorrow from Bucharest.
- Goal: be in Bucharest (other factors: cost, time, most scenic route, etc.)
- State: be in a city (defined by the map)
- Action: transition between states (highways defined by the map)

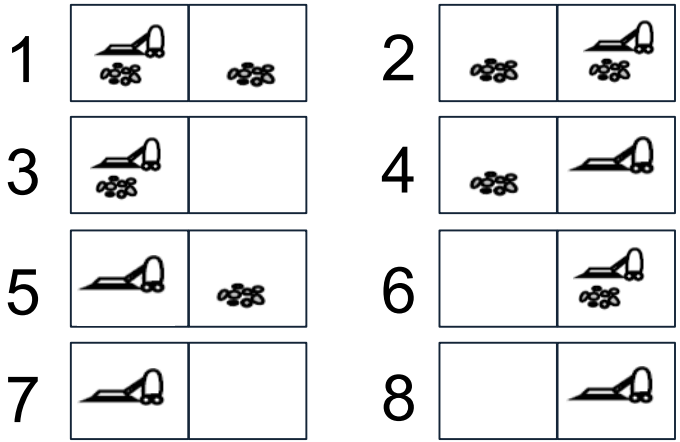
# Example: Vacuum Cleaner Agent

- Robotic vacuum cleaners move autonomously
- Some can come back to a docking station to charge their batteries
- A few are able to empty their dust containers into the dock as well


Image source: https://commons.wikimedia.org/wiki/File:IRobot_Roomba_780.jpg

# Example: A Simple Vacuum World

Two locations, each location may or may not contain dirt, and the agent may be in one location or the other.




- 8 possible world states
- Possible actions: left, right, and suck
- Goal: clean up all dirt → Two goal states, i.e. {7, 8}

# Well-Defined Formulation

| | |
|---|---|
| Definition of a problem | The information used by an agent to decide what to do |
| Specification | • Initial state<br>• Action set, i.e. available actions (successor functions)<br>• State space, i.e. states reachable from the initial state<br>  • Solution path: sequence of actions from one state to another<br>• Goal test predicate<br>  • Single state, enumerated list of states, abstract properties<br>• Cost function<br>  • Path cost g(n), sum of all (action) step costs along the path |
| Solution | A path (a sequence of operators leading) from the Initial-State to a state that satisfies the Goal-Test |

# Measuring Problem-Solving Performance

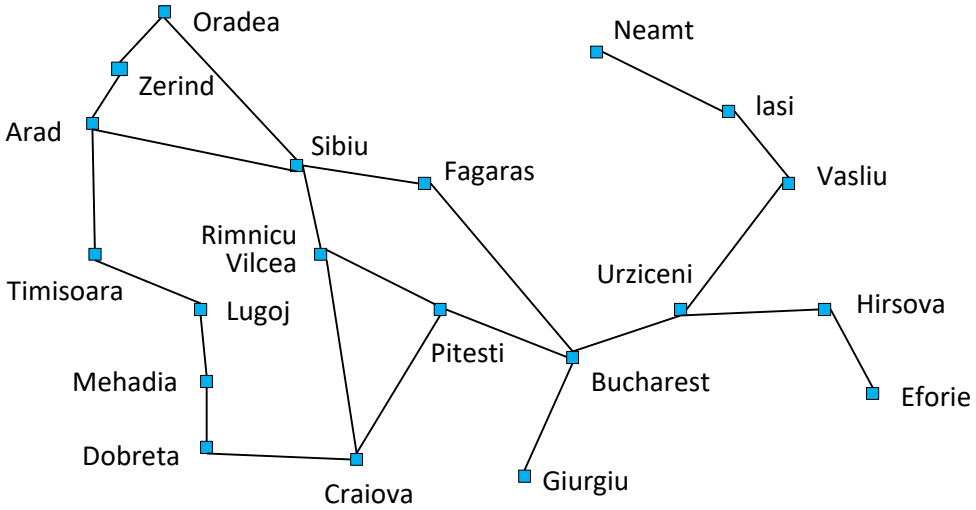
Search Cost

- What does it cost to find the solution?
    - e.g. How long (time)? How many resources used (memory)?

Total cost of problem-solving

- Search cost (“offline") + Execution cost (“online")
- Trade-offs often required
    - Search a very long time for the optimal solution, or
    - Search a shorter time for a “good enough" solution

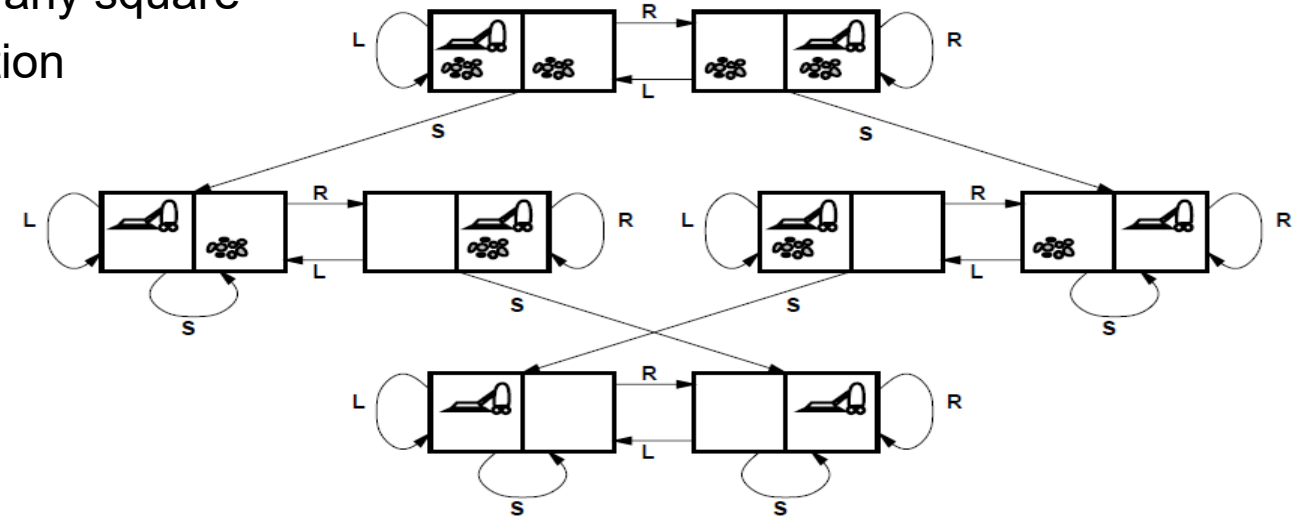
# Single-State Problem Example




- Initial state: e.g., “at Arad“
- Set of possible actions and the corresponding next states
  - e.g., Arad → Zerind
- Goal test:
  - explicit (e.g., x = “at Bucharest")
- Path cost function
  - e.g., sum of distances, number of operators executed solution: a sequence of operators leading from the initial state to a goal state

# Example: Vacuum World (Single-state Version)

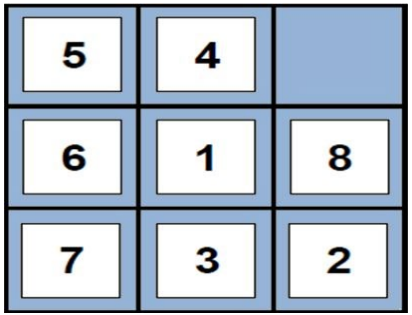
- Initial state: one of the eight states shown previously
- Actions: left, right, suck
- Goal test: no dirt in any square
- Path cost: 1 per action



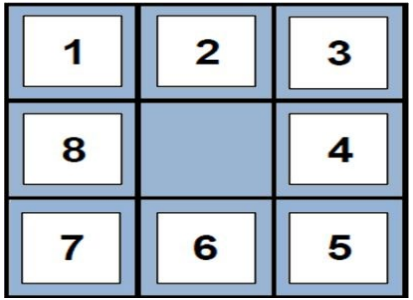
# Example: 8-puzzle

- States: integer locations of tiles
  - number of states = 9!
- Actions: move blank left, right, up, down
- Goal test: = goal state (given)
- Path cost: 1 per move

**Start state**

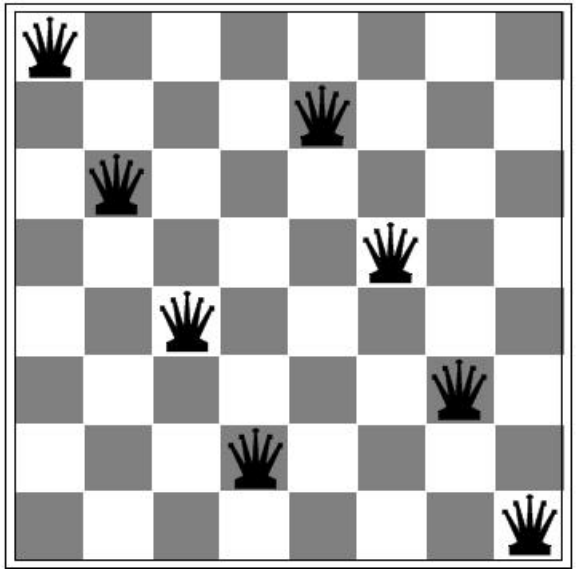



**Goal state**



# Example: 8-queens

- States: Any arrangement of 0 to 8 queens on the board
- Actions: Add a queen to any empty square
- Goal test: 8 queens are on the board, none attacked
- Path cost: Not necessary



# Real-World Problems

Route finding problems:

- Routing in computer networks
- Robot navigation
- Automated travel advisory
- Airline travel planning

Touring problems:

- Traveling Salesperson problem
- “Shortest tour": visit every city exactly once

Image source: https://commons.wikimedia.org/wiki/File:Font_Awesome_5_solid_route.svg

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

# Introduction to Data Science and Artificial Intelligence

## Solving Problems by Search: Uninformed Search and Informed Search

Assoc Prof Bo AN

*Research area*: artificial intelligence, computational game theory, optimization
www.ntu.edu.sg/home/boan
*Email*: boan@ntu.edu.sg
*Office*: N4-02b-55

# Lesson Outline

- Uninformed search strategies
- Informed search strategies
  - Greedy search
  - A * search

# Review: Well-Defined Formulation

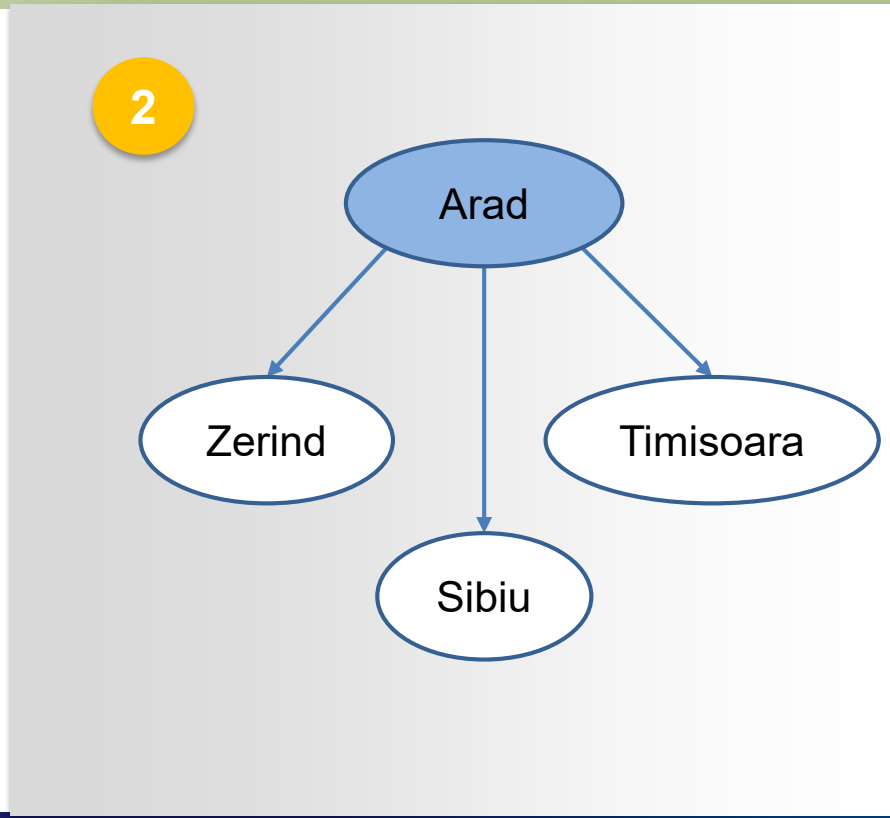
| | |
|---|---|
| Definition of a problem | The information used by an agent to decide what to do |
| Specification | • Initial state<br>• Action set, i.e. available actions (successor functions)<br>• State space, i.e. states reachable from the initial state<br>  • Solution path: sequence of actions from one state to another<br>• Goal test predicate<br>  • Single state, enumerated list of states, abstract properties<br>• Cost function<br>  • Path cost *g(n),* sum of all (action) step costs along the path |
| Solution | A path (a sequence of operators leading) from the Initial-State to a state that satisfies the Goal-Test |

# Search Algorithms

- Exploration of state space by generating successors of already-explored states
  - Frontier: candidate nodes for expansion
  - Explored set

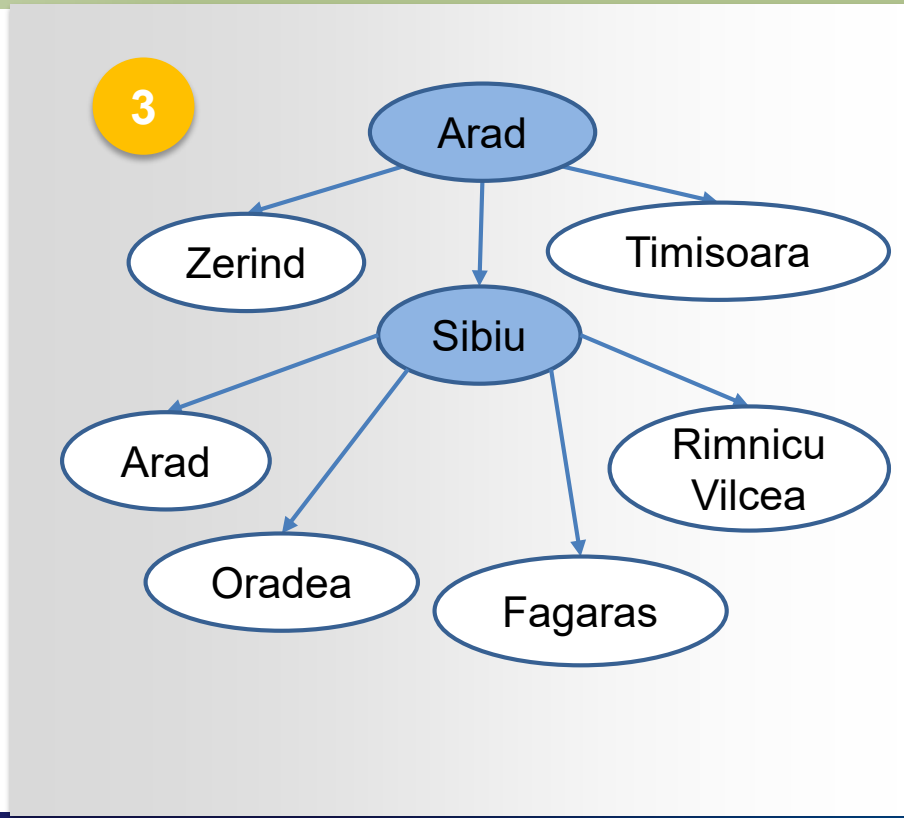
# Search Algorithms

- Exploration of state space by generating successors of already-explored states
  - Frontier: candidate nodes for expansion
  - Explored set



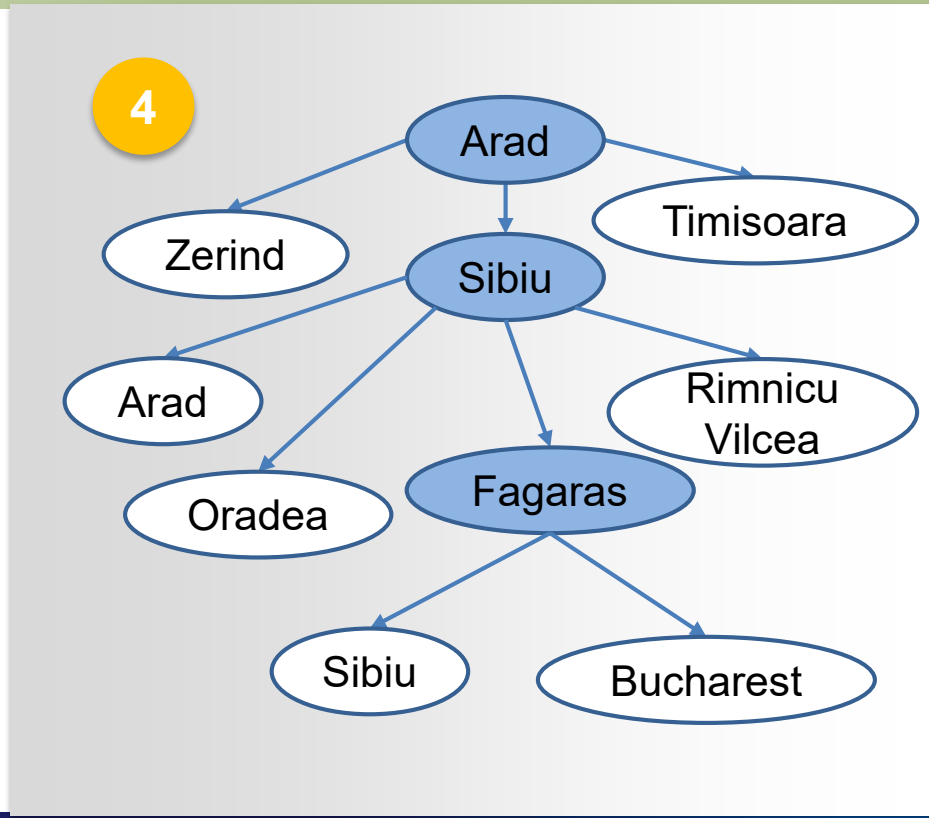
# Search Algorithms

- Exploration of state space by generating successors of already-explored states
  - Frontier: candidate nodes for expansion
  - Explored set



# Search Algorithms

- Exploration of state space by generating successors of already-explored states
  - Frontier: candidate nodes for expansion
  - Explored set



# Search Strategies

- A strategy is defined by picking the order of node expansion.
- Strategies are evaluated along the following dimensions:

| | |
|---|---|
| Completeness | Does it always find a solution if one exists? |
| Time Complexity | How long does it take to find a solution: the number of nodes generated |
| Space Complexity | Maximum number of nodes in memory |
| Optimality | Does it always find the best (least-cost) solution? |

# Search Strategies

- Branching factor
  - Maximum number of successors of any node
  - Or average branching factor

# Uninformed vs Informed

**Uninformed search strategies**

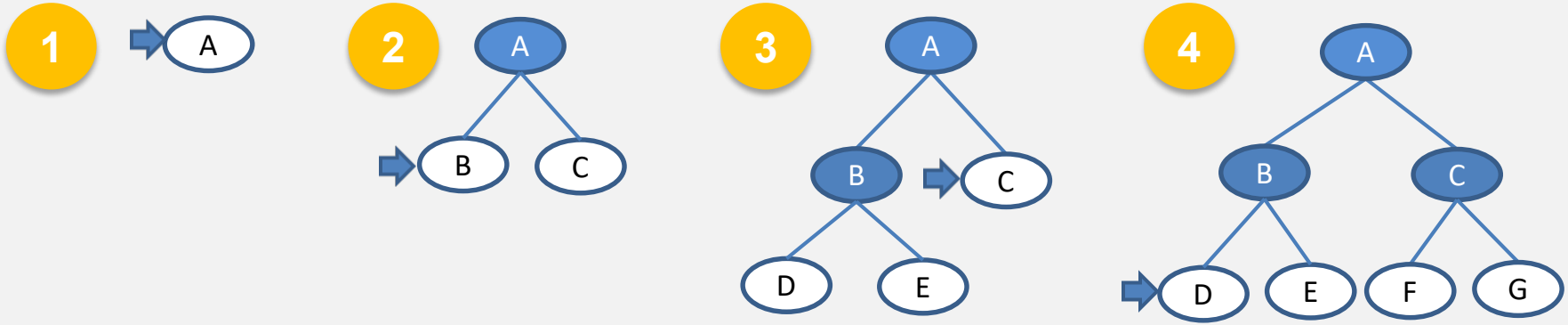
- Use only the information available in the problem definition
  1. Breadth-first search
  2. Uniform-cost search
  3. Depth-first search
  4. Depth-limited search
  5. Iterative deepening search

**Informed search strategies**

- Use problem-specific knowledge to guide the search
- Usually more efficient

# Breadth-First Search

Expand shallowest unexpanded node which can be implemented by a First-In-First-Out (FIFO) queue




Denote

- b: maximum branching factor of the search tree
- d: depth of the least-cost solution
- Complete: Yes
- Optimal: Yes when all step costs equally

# Complexity of BFS

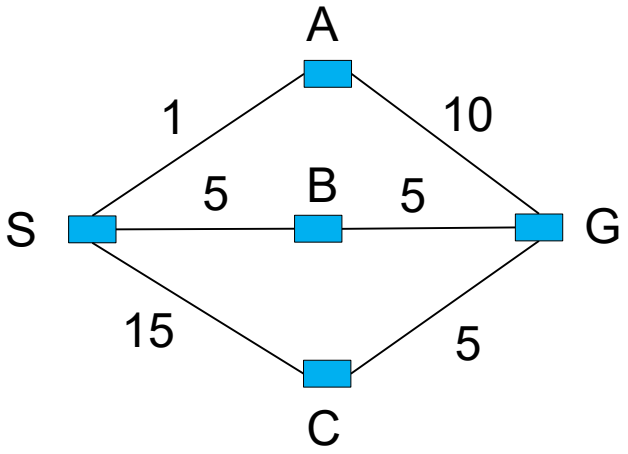
- Hypothetical state-space, where every node can be expanded into *b* new nodes, solution of path-length *d*
- Time: $1 + b + b^2 + b^3 + \ldots + b^d = O(b^d)$
- Space: (keeps every node in memory) $O(b^d)$are equal

| Depth | Nodes | | Time | | Memory |
|---|---|---|---|---|---|
| 0 | 1 | 1 | millisecond | 100 | bytes |
| 2 | 111 | 0.1 | seconds | 11 | kilobytes |
| 4 | 11111 | 11 | seconds | 1 | kilobytes |
| 6 | $10^6$ | 18 | minutes | 111 | megabyte |
| 8 | $10^8$ | 31 | hours | 11 | gigabytes |
| 10 | $10^{10}$ | 128 | days | 1 | terabyte |
| 12 | $10^{12}$ | 35 | years | 111 | terabytes |
| 14 | $10^{14}$ | 3500 | years | 11111 | terabytes |

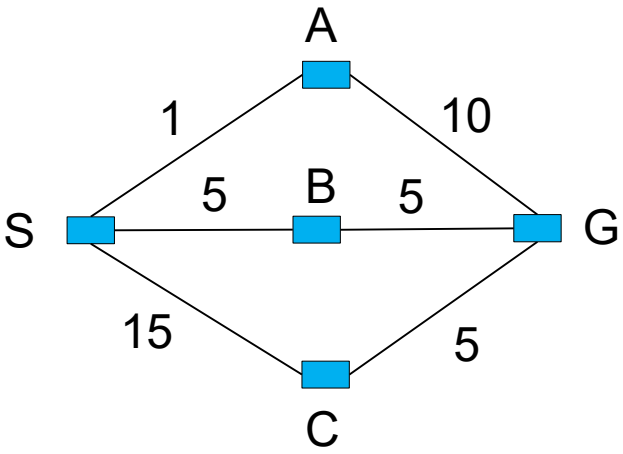
# Uniform-Cost Search

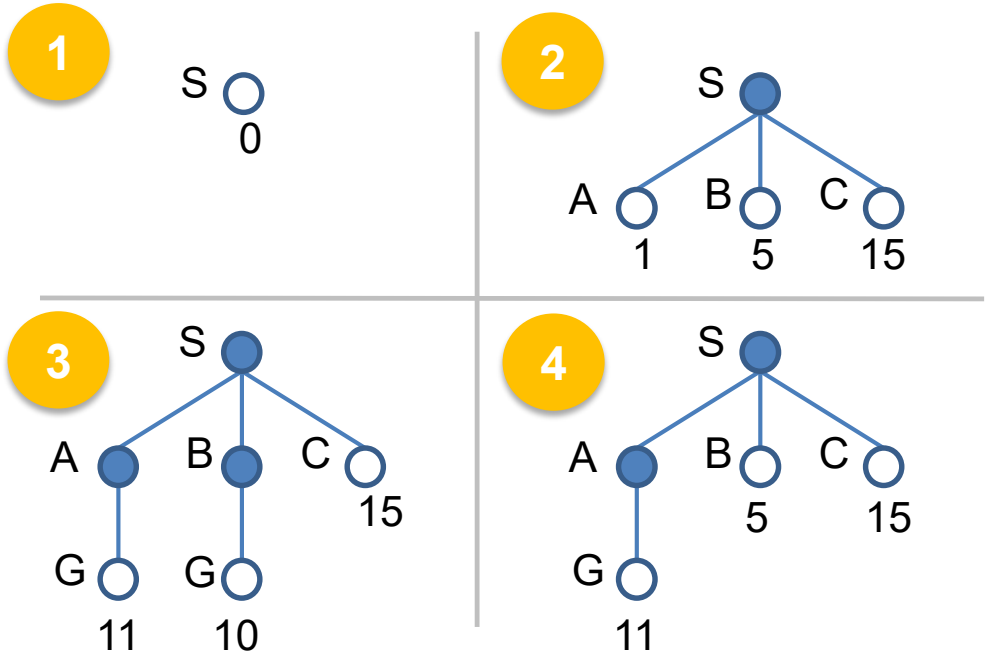
To consider edge costs, expand unexpanded node with the least path cost *g*

- Modification of breath-first search
- Instead of First-In-First-Out (FIFO) queue, using a priority queue with path cost *g(n)* to order the elements
- BFS = UCS with *g(n)* = Depth(n)



# Uniform-Cost Search



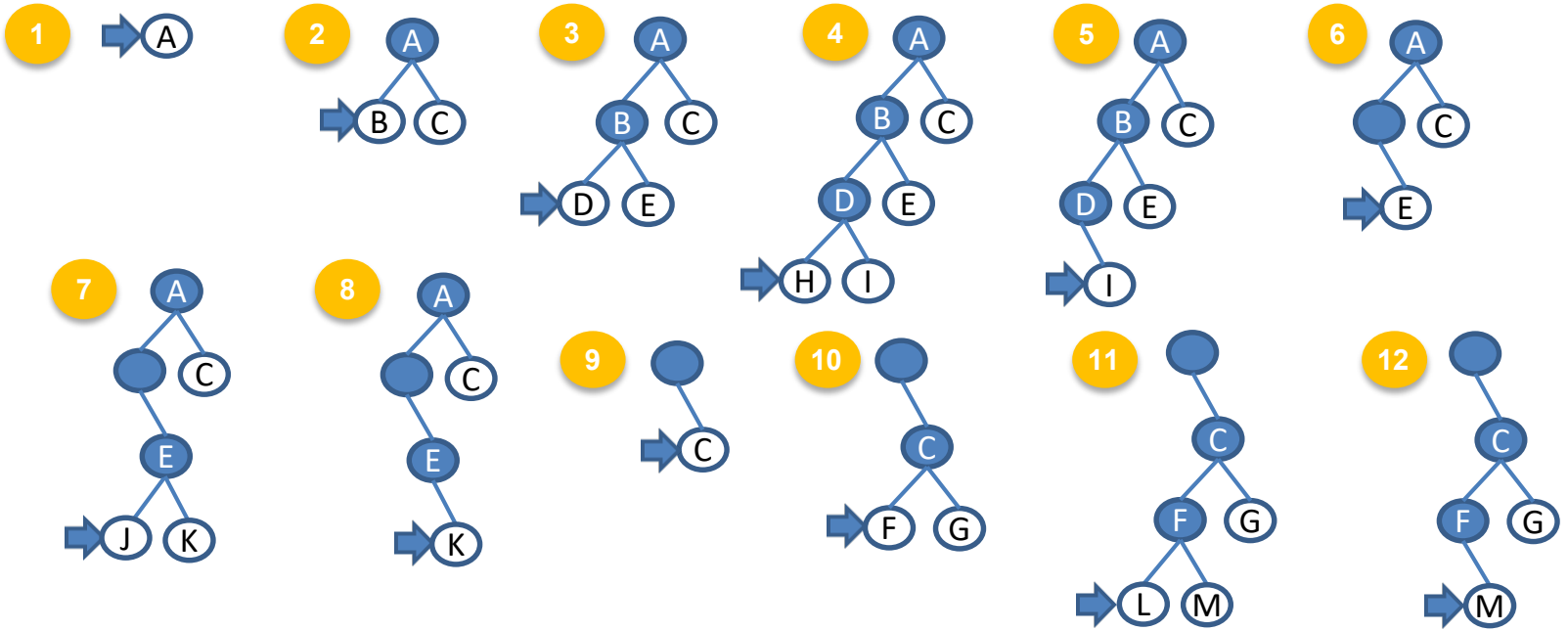

Here we do not expand notes that have been expanded.

# Uniform-Cost Search

| | |
|---|---|
| Complete | Yes |
| Time | # of nodes with path cost g <= cost of optimal solution (eqv. # of nodes pop out from the priority queue) |
| Space | # of nodes with path cost g <= cost of optimal solution |
| Optimal | Yes |

# Depth-First Search

Expand deepest unexpanded node which can be implemented by a Last-In-First-Out (LIFO) stack, Backtrack only when no more expansion



# Depth-First Search

Denote

- m: maximum depth of the state space

| | |
|---|---|
| Complete | • infinite-depth spaces: No<br>• finite-depth spaces with loops: No<br>  • with repeated-state checking: Yes<br>• finite-depth spaces without loops: Yes |
| Time | $O(b^m)$<br>If solutions are dense, may be much faster than breadth-first |
| Space | $O(bm)$ |
| Optimal | No |

# Depth-Limited Search

To avoid infinite searching, Depth-first search with a cutoff on the max depth / of a path

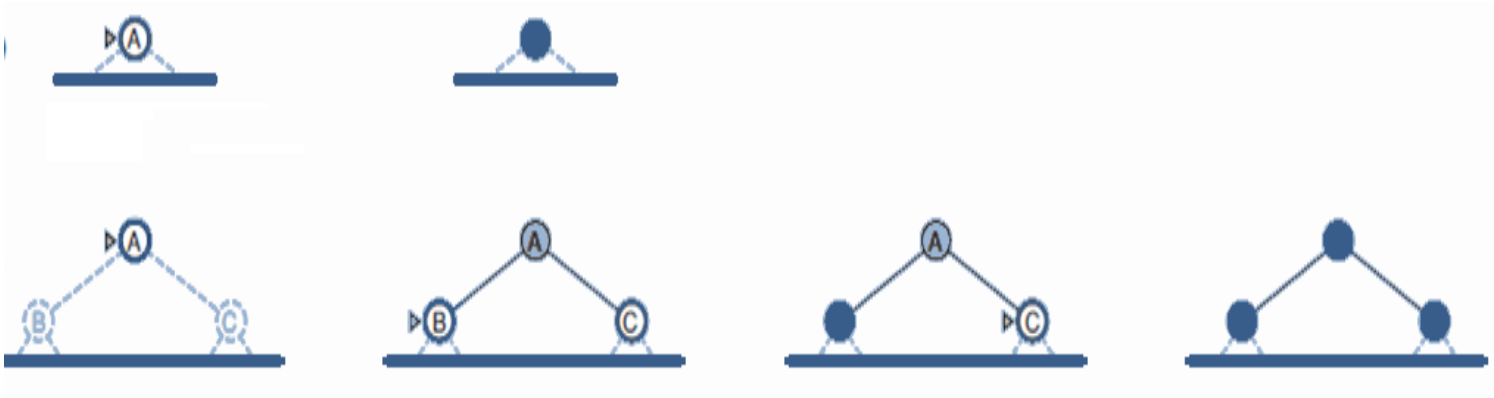
| | |
|---|---|
| Complete | Yes, if $I \geq d$ |
| Time | $O(b^I)$ |
| Space | $O(bI)$ |
| Optimal | No |

# Iterative Deepening Search

Iteratively estimate the max depth / of DLS one-by-one

Limit = 0

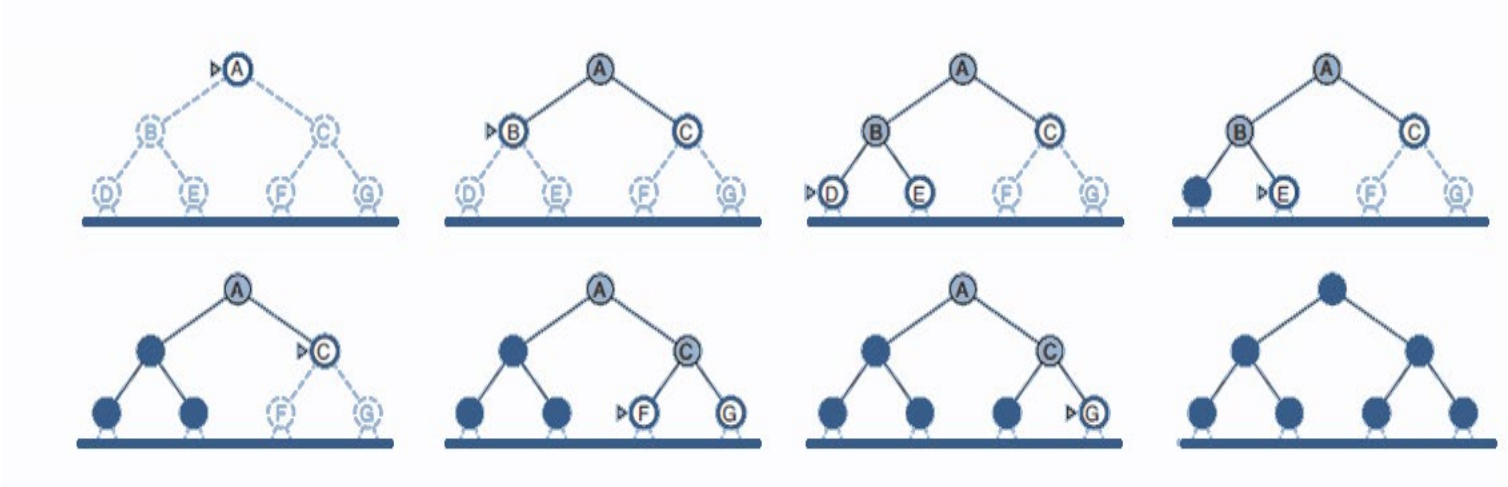
Limit = 1



# Iterative Deepening Search

Iteratively estimate the max depth / of DLS one-by-one

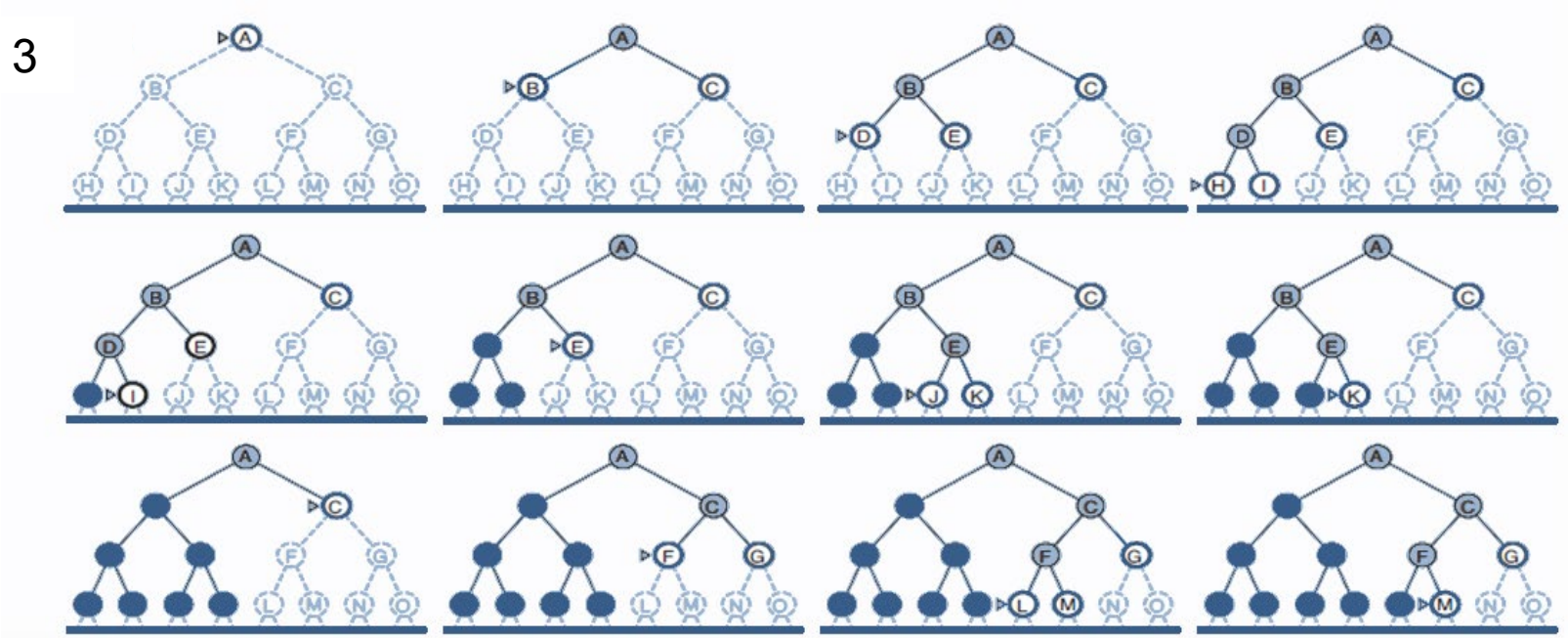
Limit = 2



# Iterative Deepening Search

Iteratively estimate the max depth / of DLS one-by-one

Limit = 3



# Iterative Deepening Search...

```
Function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution sequence
   inputs: problem, a problem

   for depth   0 to ∞ do
    if DEPTH-LIMITED-SEARCH(problem, depth) succeeds then return its result
   end
   return failure
```

| | |
|---|---|
| Complete | Yes |
| Time | $O(b^d)$ |
| Space | $O(bd)$ |
| Optimal | Yes |

# Summary (we make assumptions for optimality)

| Criterion | Breadth-first | Uniform-Cost | Depth-First | Depth-Limited | Iterative Deepening | Bidirectional (if applicable) |
|---|---|---|---|---|---|---|
| Time | $b^d$ | $b^d$ | $b^m$ | $b^l$ | $b^d$ | $b^{d/2}$ |
| Space | $b^d$ | $b^d$ | $bm$ | $bl$ | $bd$ | $b^{d/2}$ |
| Optimal | Yes | Yes | No | No | Yes | Yes |
| Complete | Yes | Yes | No | Yes, if $l \geq d$ | Yes | Yes |

Question to think:

- If a search strategy is optimal, is it also complete?

# General Search

Uninformed search strategies

- Systematic generation of new states (→Goal Test)
- Inefficient (exponential space and time complexity)

Informed search strategies

- Use problem-specific knowledge
  - To decide the order of node expansion
- Best First Search: expand the most desirable unexpanded node
  - Use an evaluation function to estimate the “desirability" of each node

# Evaluation function

- Path-cost function $g(n)$
  - Cost from initial state to current state (search-node) $n$
  - No information on the cost toward the goal
- Need to estimate cost to the closest goal
- "Heuristic" function $h(n)$
  - Estimated cost of the cheapest path from $n$ to a goal state $h(n)$
    - Exact cost cannot be determined
  - depends only on the state at that node
  - $h(n)$ is not larger than the real cost (admissible)

# Greedy Search

Expands the node that appears to be closest to goal

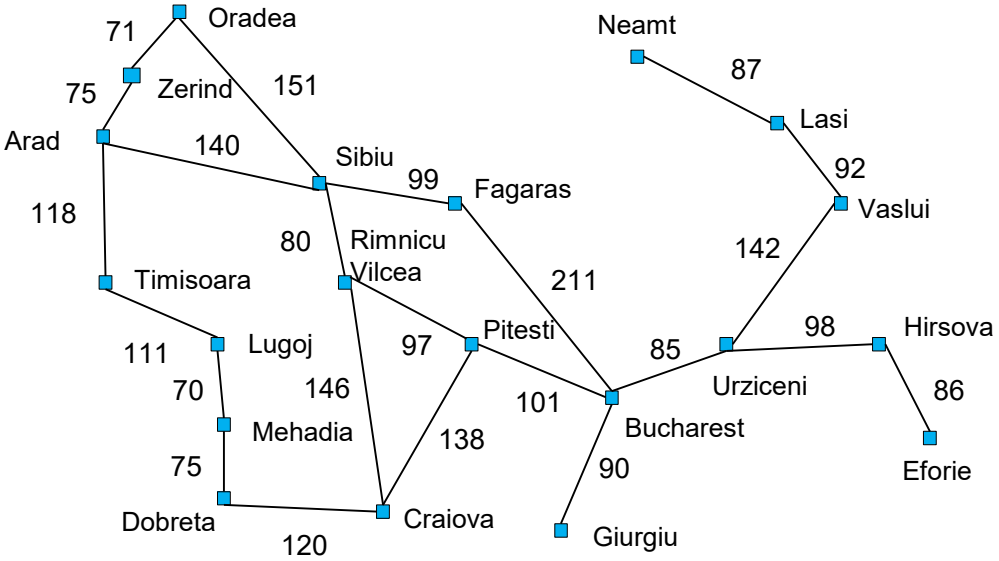
- Evaluation function $h(n)$:estimate of cost from $n$ to $goal$
- Function Greedy-Search(problem) returns solution
  - Return Best-First-Search(problem, $h$)  // $h(goal) = 0$

**Question:** How to estimation the cost from $n$ to goal?

**Answer:** Recall that we want to use problem-specific knowledge

# Example: Route-finding from Arad to Bucharest

$h(n)$ = straight-line distance from $n$ to Bucharest




- Useful but potentially fallible (heuristic)
- Heuristic functions are problem-specific

Straight-line distance to Bucharest

| | |
|---|---|
| **Arad** | 366 |
| **Bucharest** | 0 |
| **Craiova** | 160 |
| **Dobreta** | 242 |
| **Efoire** | 161 |
| **Fagaras** | 176 |
| **Giurgiu** | 77 |
| **Hirsova** | 151 |
| **Lasi** | 226 |
| **Lugoj** | 244 |
| **Mehadia** | 241 |
| **Neamt** | 234 |
| **Oradea** | 380 |
| **Pitesti** | 98 |
| **Rimnicu Vilcea** | 193 |
| **Sibiu** | 253 |
| **Timisoara** | 329 |
| **Urziceni** | 80 |
| **Vaslui** | 199 |
| **Zerind** | 374 |

# Example

a) The initial state

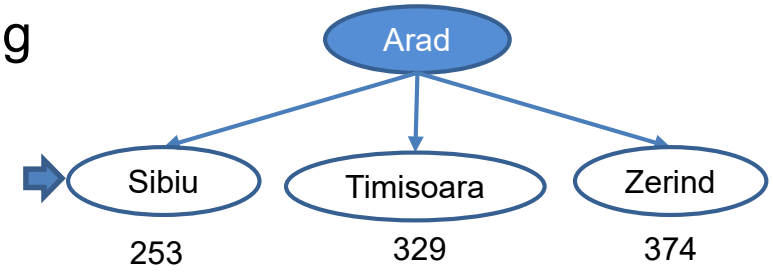

366

Straight-line distance to Bucharest

| | |
|---|---|
| **Arad** | 366 |
| **Bucharest** | 0 |
| **Craiova** | 160 |
| **Dobreta** | 242 |
| **Efoire** | 161 |
| **Fagaras** | 176 |
| **Giurgiu** | 77 |
| **Hirsova** | 151 |
| **Lasi** | 226 |
| **Lugoj** | 244 |
| **Mehadia** | 241 |
| **Neamt** | 234 |
| **Oradea** | 380 |
| **Pitesti** | 98 |
| **Rimnicu Vilcea** | 193 |
| **Sibiu** | 253 |
| **Timisoara** | 329 |
| **Urziceni** | 80 |
| **Vaslui** | 199 |
| **Zerind** | 374 |

# Example

b) After expanding Arad

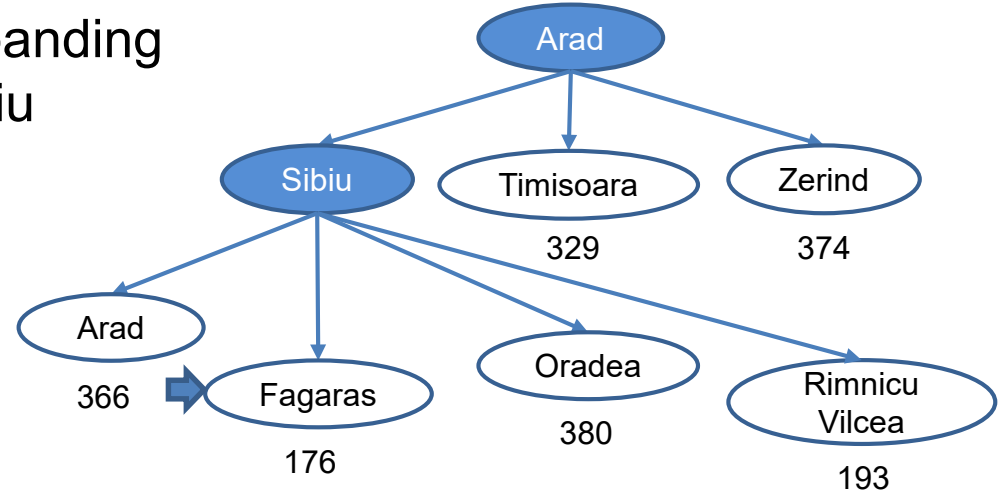



Straight-line distance to Bucharest

| | |
|---|---|
| **Arad** | 366 |
| **Bucharest** | 0 |
| **Craiova** | 160 |
| **Dobreta** | 242 |
| **Efoire** | 161 |
| **Fagaras** | 176 |
| **Giurgiu** | 77 |
| **Hirsova** | 151 |
| **Lasi** | 226 |
| **Lugoj** | 244 |
| **Mehadia** | 241 |
| **Neamt** | 234 |
| **Oradea** | 380 |
| **Pitesti** | 98 |
| **Rimnicu Vilcea** | 193 |
| **Sibiu** | 253 |
| **Timisoara** | 329 |
| **Urziceni** | 80 |
| **Vaslui** | 199 |
| **Zerind** | 374 |

# Example

c) After expanding Sibiu

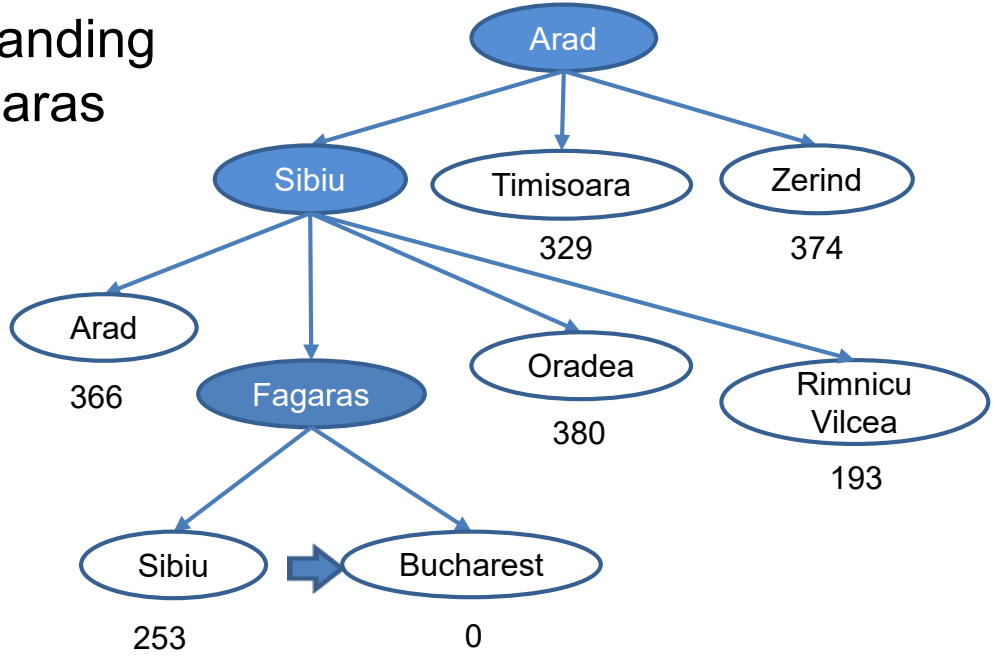



Straight-line distance to Bucharest

| | |
|---|---|
| Arad | 366 |
| Bucharest | 0 |
| Craiova | 160 |
| Dobreta | 242 |
| Efoire | 161 |
| Fagaras | 176 |
| Giurgiu | 77 |
| Hirsova | 151 |
| Lasi | 226 |
| Lugoj | 244 |
| Mehadia | 241 |
| Neamt | 234 |
| Oradea | 380 |
| Pitesti | 98 |
| Rimnicu Vilcea | 193 |
| Sibiu | 253 |
| Timisoara | 329 |
| Urziceni | 80 |
| Vaslui | 199 |
| Zerind | 374 |

# Example

d) After expanding Fagaras



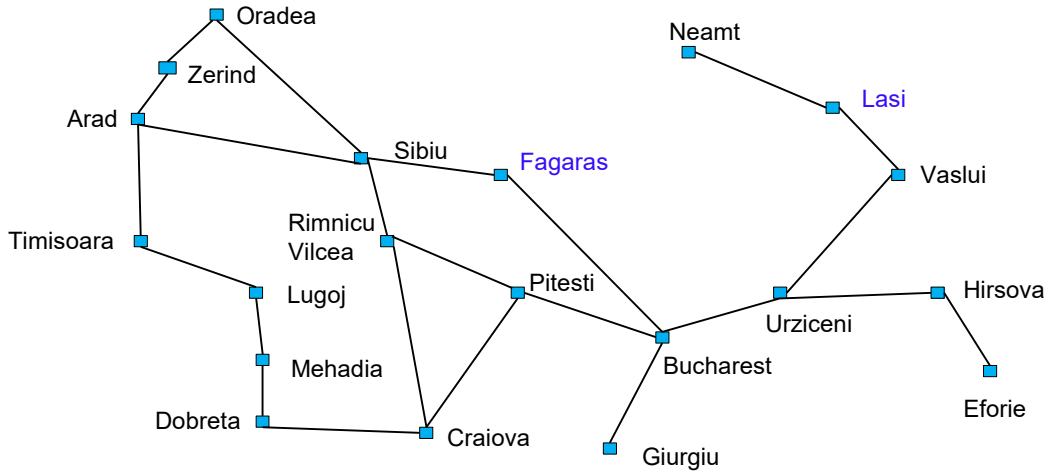

Straight-line distance to Bucharest

| | |
|---|---|
| **Arad** | 366 |
| **Bucharest** | 0 |
| **Craiova** | 160 |
| **Dobreta** | 242 |
| **Efoire** | 161 |
| **Fagaras** | 176 |
| **Giurgiu** | 77 |
| **Hirsova** | 151 |
| **Lasi** | 226 |
| **Lugoj** | 244 |
| **Mehadia** | 241 |
| **Neamt** | 234 |
| **Oradea** | 380 |
| **Pitesti** | 98 |
| **Rimnicu Vilcea** | 193 |
| **Sibiu** | 253 |
| **Timisoara** | 329 |
| **Urziceni** | 80 |
| **Vaslui** | 199 |
| **Zerind** | 374 |

# Complete?

**Question:** Is this approach complete?

**Example: Find a path from Lasi to Fagaras**




**Answer:** No

# Greedy Search...

- m: maximum depth of the search space

| | |
|---|---|
| Complete | No |
| Time | $O(b^m)$ |
| Space | $O(b^m)$(keeps all nodes in memory) |
| Optimal | No |

Question to think:

- Is it possible to combine functions g(n) and h(n) in one search strategy?

# A * Search

- Uniform-cost search
  - *g(n):* cost to reach n (Past Experience)
  - optimal and complete, but can be very inefficient
- Greedy search
  - *h(n):* cost from n to goal (Future Prediction)
  - neither optimal nor complete, but cuts search space considerably

# A * Search

Idea: Combine Greedy search with Uniform-Cost search

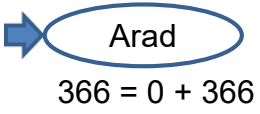
Evaluation function: $f(n) = g(n) + h(n)$

- *f (n):* estimated total cost of path through *n* to goal (Whole Life)
- If g = 0 → greedy search;  If h = 0 → uniform-cost search
- Function A* Search(problem) returns solution
  - Return Best-First-Search(problem, $g + h$)

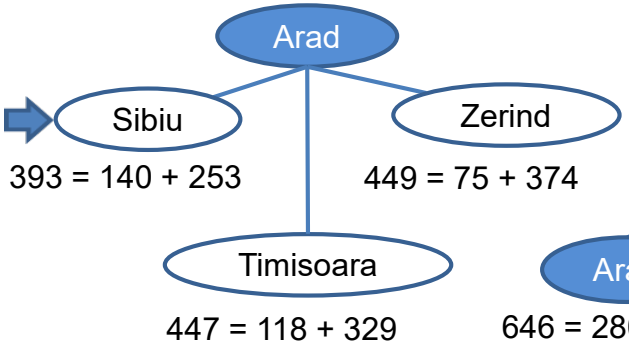
# Example: Route-finding from Arad to Bucharest

Best-first-search with evaluation function $g + h$

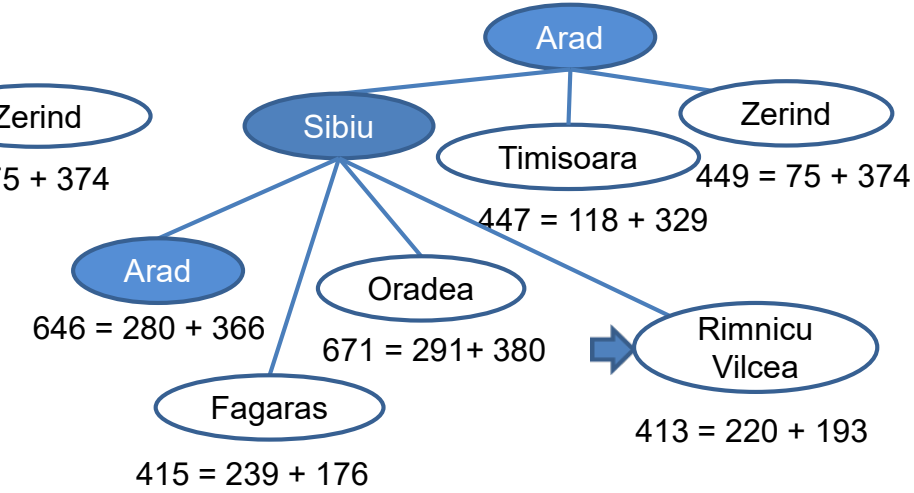
(a) The initial state




(b) After expanding Arad



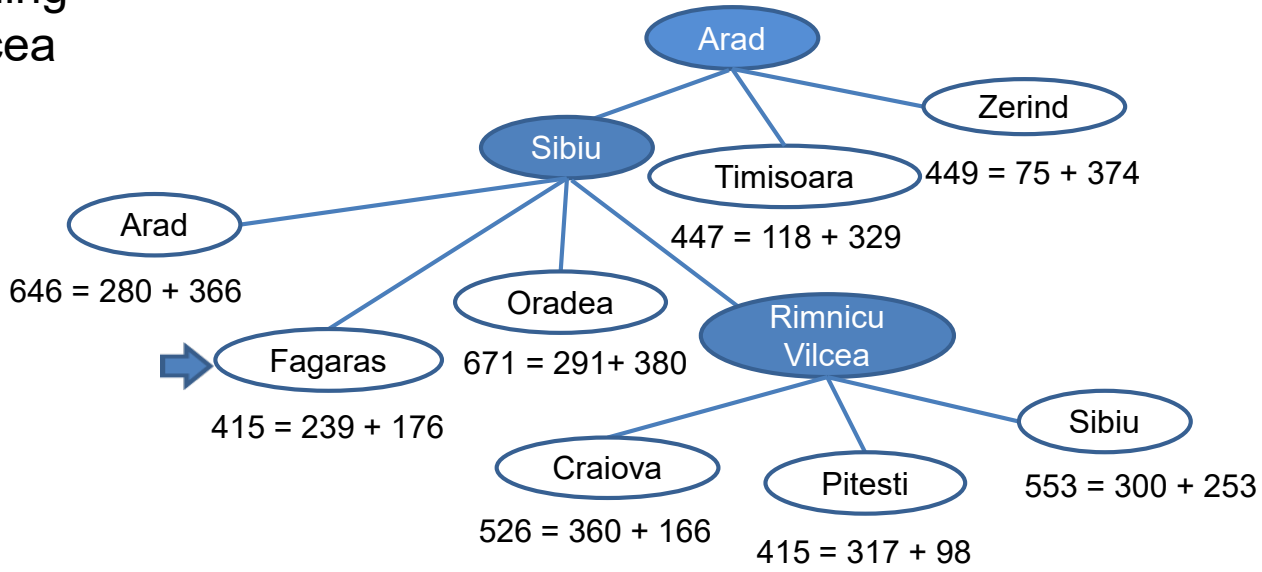

(c) After expanding Sibiu



# Example: Route-finding from Arad to Bucharest

Best-first-search with evaluation function $g + h$

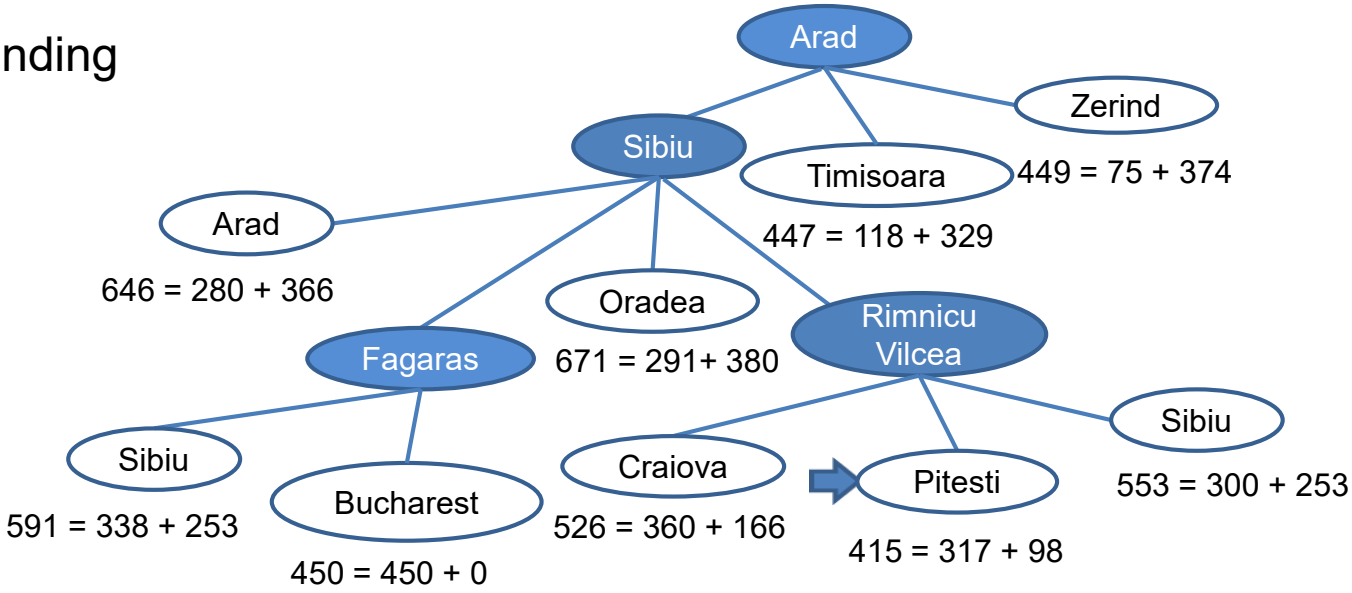
(d) After expanding
Rimnicu Vilcea



# Example: Route-finding from Arad to Bucharest

Best-first-search with evaluation function $g + h$

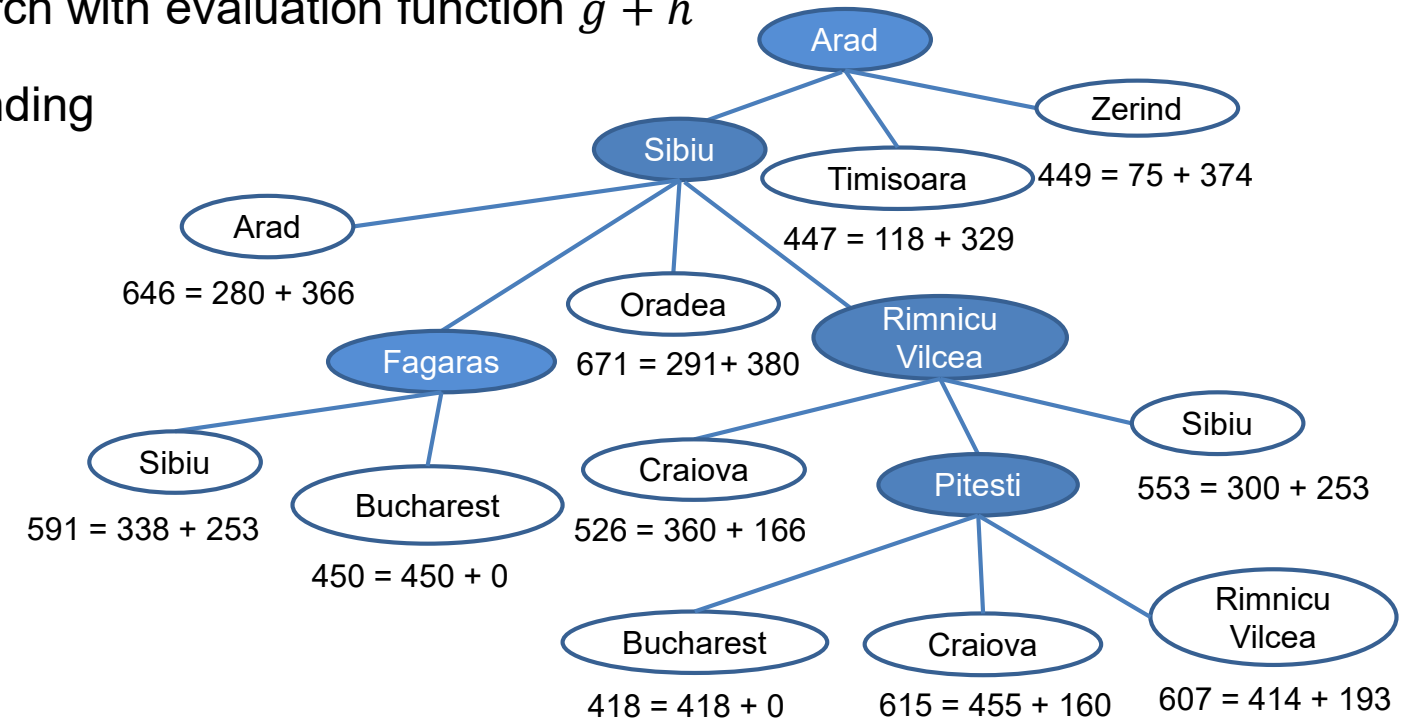
(e) After expanding
Fagaras



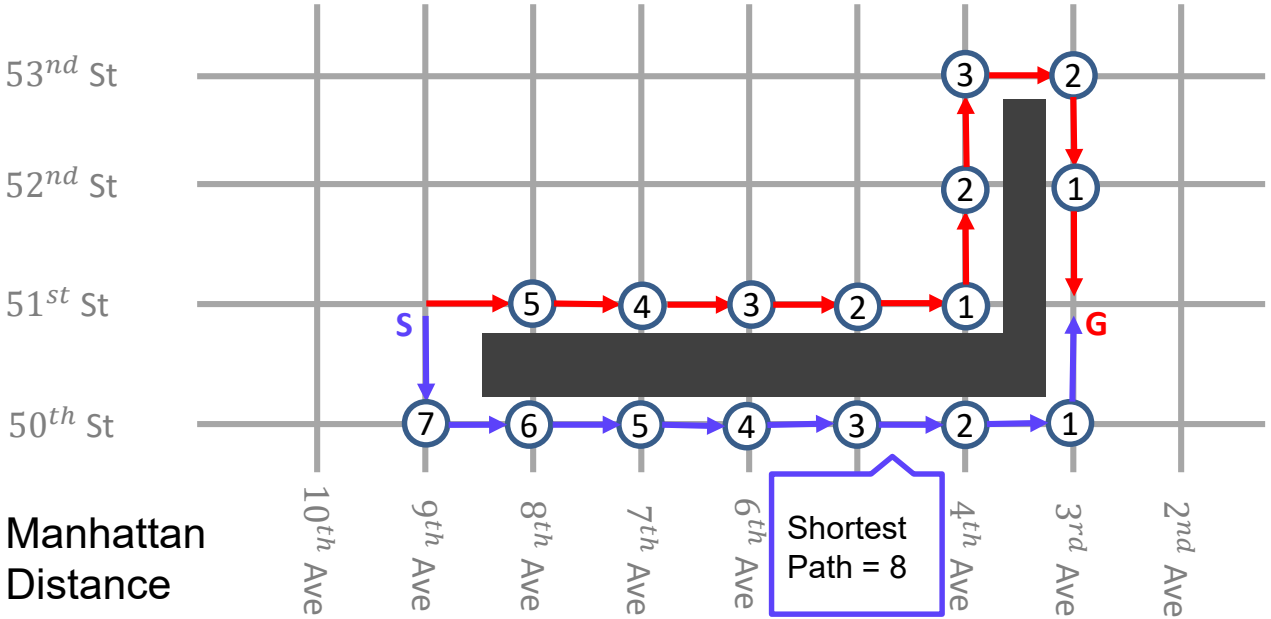
# Example: Route-finding from Arad to Bucharest

Best-first-search with evaluation function $g + h$

(f) After expanding
Pitesti

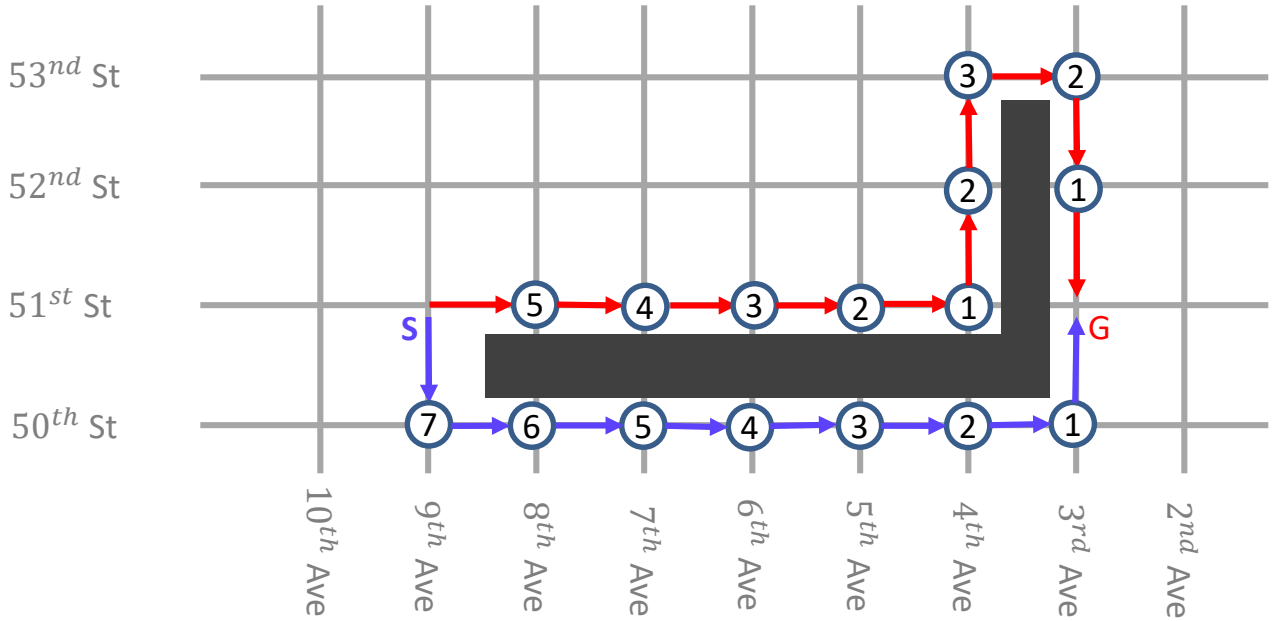


# Example: Route-finding in Manhattan

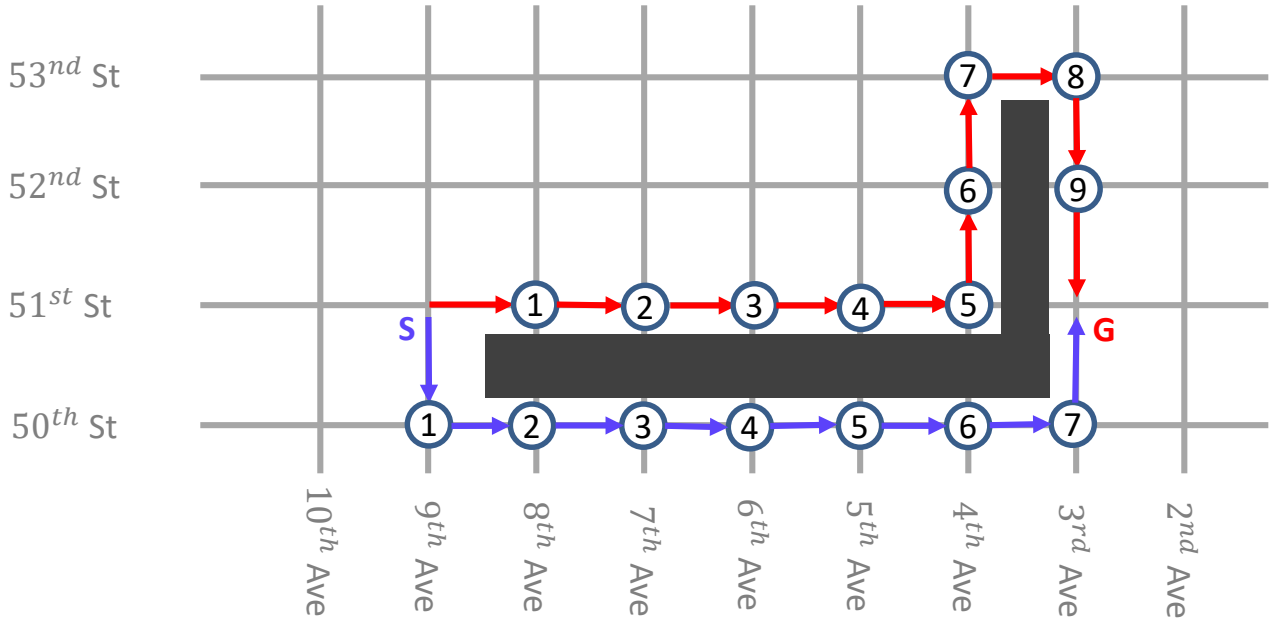



Manhattan Distance Heuristic

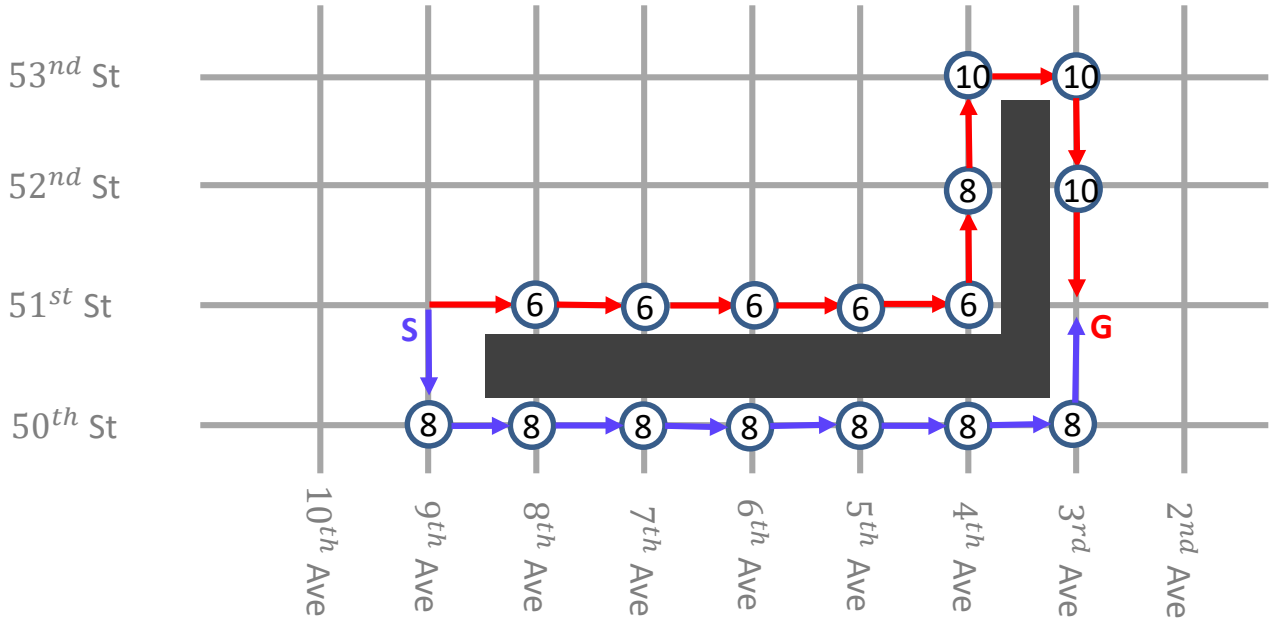
# Example: Route-finding in Manhattan (Greedy)



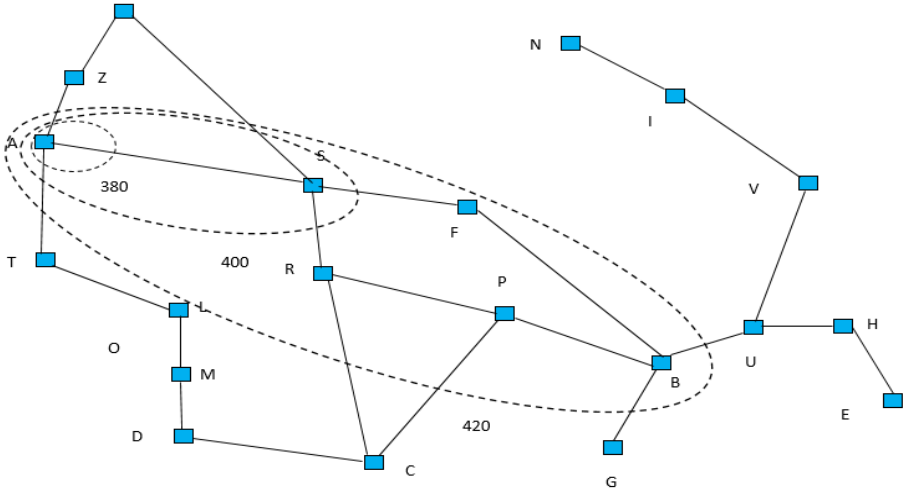
# Example: Route-finding in Manhattan (UCS)



# Example: Route-finding in Manhattan (A*)



# Complexity of A*




| | |
|---|---|
| Time | Exponential in length of solution |
| Space | (all generated nodes are kept in memory) Exponential in length of solution |

With a good heuristic, significant savings are still possible compared to uninformed search methods

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

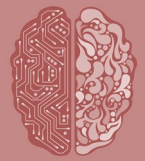
# Introduction to Data Science and Artificial Intelligence

## Constraint Satisfaction and Game Playing

Assoc Prof Bo AN

*Research area*: artificial intelligence, computational game theory, optimization
www.ntu.edu.sg/home/boan
*Email*: boan@ntu.edu.sg
*Office*: N4-02b-55

# Lesson Outline

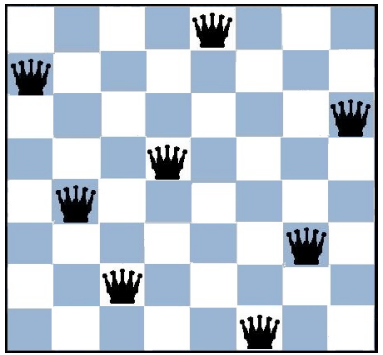


- Constraint Satisfaction
  - Backtracking search
  - Forward checking and constraint propagation
- Game Playing
  - Games as search problems
  - Minimax search strategy
  - Evaluation functions

# Constraint Satisfaction Problem (CSP)

Goal: discover some state that satisfies a given set of constraints

**Example: 8-Queens Problem**



**Example: Cryptarithmetic Puzzle**

$$\begin{array}{r} \text{S E N D} \\ +\,\text{M O R E} \\ \hline \text{M O N E Y} \end{array}$$

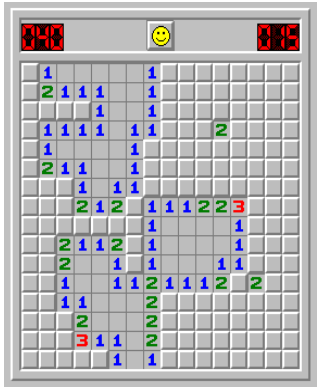
# Constraint Satisfaction Problem (CSP)

Goal: discover some state that satisfies a given set of constraints

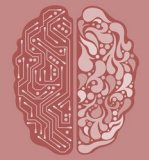
## Example: Sudoku

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

## Example: Minesweeper



# Examples: Real-world CSPs



- Assignment problems
  - e.g. who teaches what class
- Timetabling problems
  - e.g. which class is offered when and where?
- Hardware configuration
- Transportation scheduling
- Factory scheduling
- Floor-planning

# CSP

State

- defined by variables $V_i$ with values from domain $D_i$

**Example: 8-queens**

- Variables: locations of each of the eight queens
- Values: squares on the board

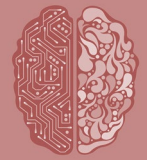
Goal test

- a set of constraints specifying allowable combinations of values for subsets of variables

**Example: 8-queens**

- Goal test: No two queens in the same row, column or diagonal

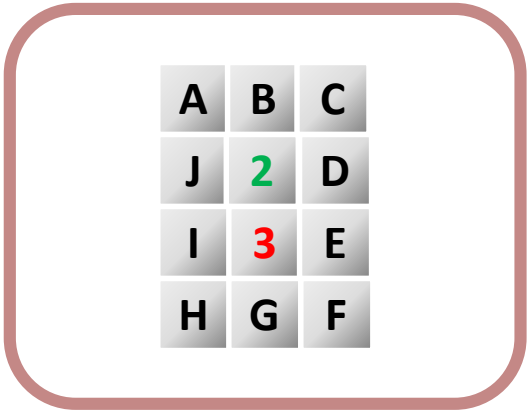
# Example: Cryptarithmetic Puzzle



```
  S E N D
+ M O R E
---------
M O N E Y
```

- Variables: D, E, M, N, O, R, S, Y
- Domains: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
- Constraints
  - $Y = D + E$ *or* $Y = D + E - 10$, etc.
  - D ≠ E, D ≠ M, D ≠ N, etc.
  - M ≠ 0, S ≠ 0 (unary constraints: concern the value of a single variable)

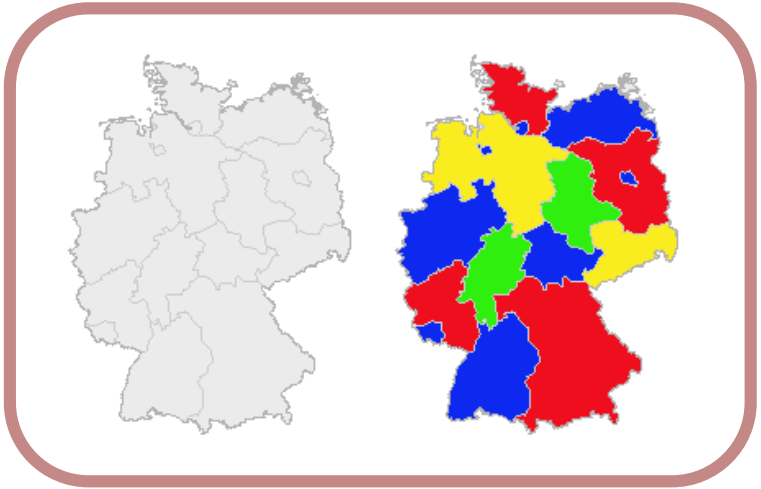
# Example: Minesweeper




- Variables: The cells
- Domains: {0; 1} representing {safe, mined}
- Constraints: Each cell has a number $m \in \{ 1, \ldots , 8\}$ indicating the number of mines nearby, so $m$ is equal to sum of value of neighbour cells

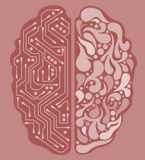
# Example: Map Colouring

Colour a map so that no adjacent parts have the same colour



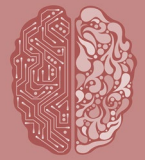
- Variables: Countries *Ci*
- Domains: {*Red, Blue, Green*}
- Contraints: C1 ≠ C2, C1 ≠ C5, etc.
  - binary constraints

# Some Definitions



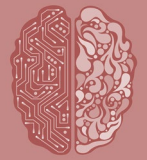
- A state of the problem is defined by an **assignment** of values to some or all of the variables.
- An assignment that does not violate any constraints is called a **consistent** or **legal** assignment.
- A **solution** to a CSP is an assignment with every variable given a value (**complete**) and the assignment satisfies all the constraints.

# Applying Standard Search



- States: defined by the values assigned so far
- Initial state: all variables unassigned
- Actions: assign a value to an unassigned variable
- Goal test: all variables assigned, no constraints violated

# Applying Standard Search



**Question:** How to represent constraints?

**Answer:** Explicitly (e.g., D ≠ E)

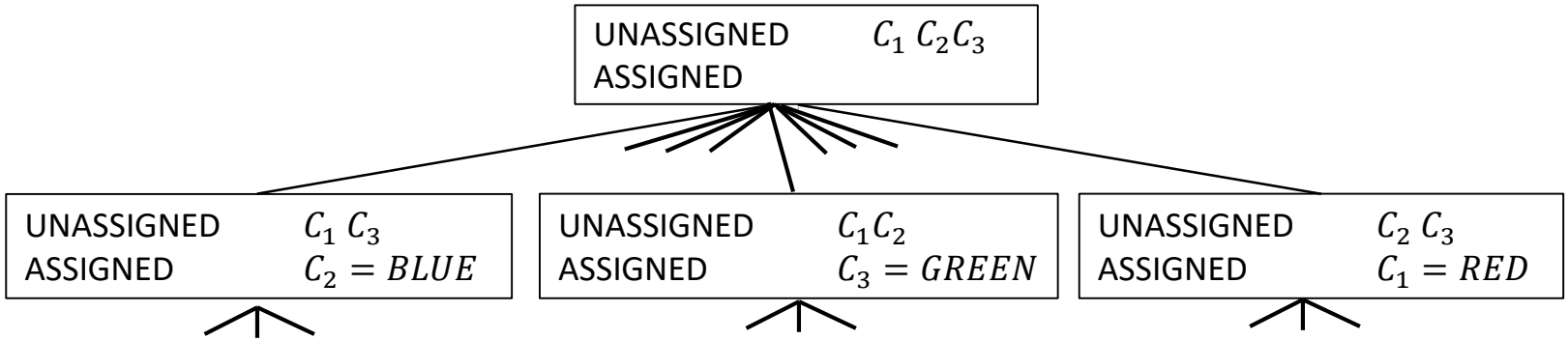
**Example**

- Row the 1st queen occupies: $V_1 \in \{1, 2, 3, 4, 5, 6, 7, 8\}$
  (similarly, for $V_2$)
- No-attack constraint for $V_1$ and $V_2$:
  { <1, 3> , <1, 4>, <1, 5>, … , <2, 4>, <2, 5>, …}

Implicitly: use a function to test for constraint satisfaction

# Applying Standard Search...

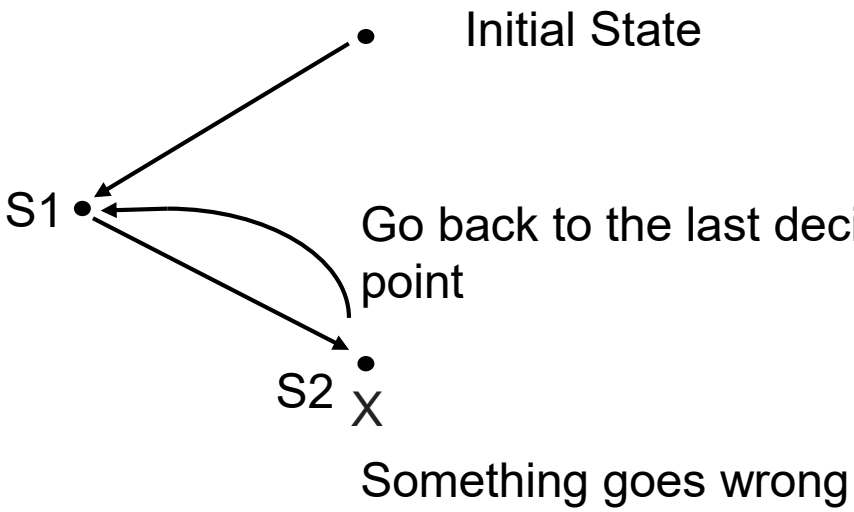
## Example: map colouring




- Number of variables: $n$
- Max. depth of space: $n$
- Depth of solution state: $n$ (all variables assigned)
- Search algorithm: depth-first search

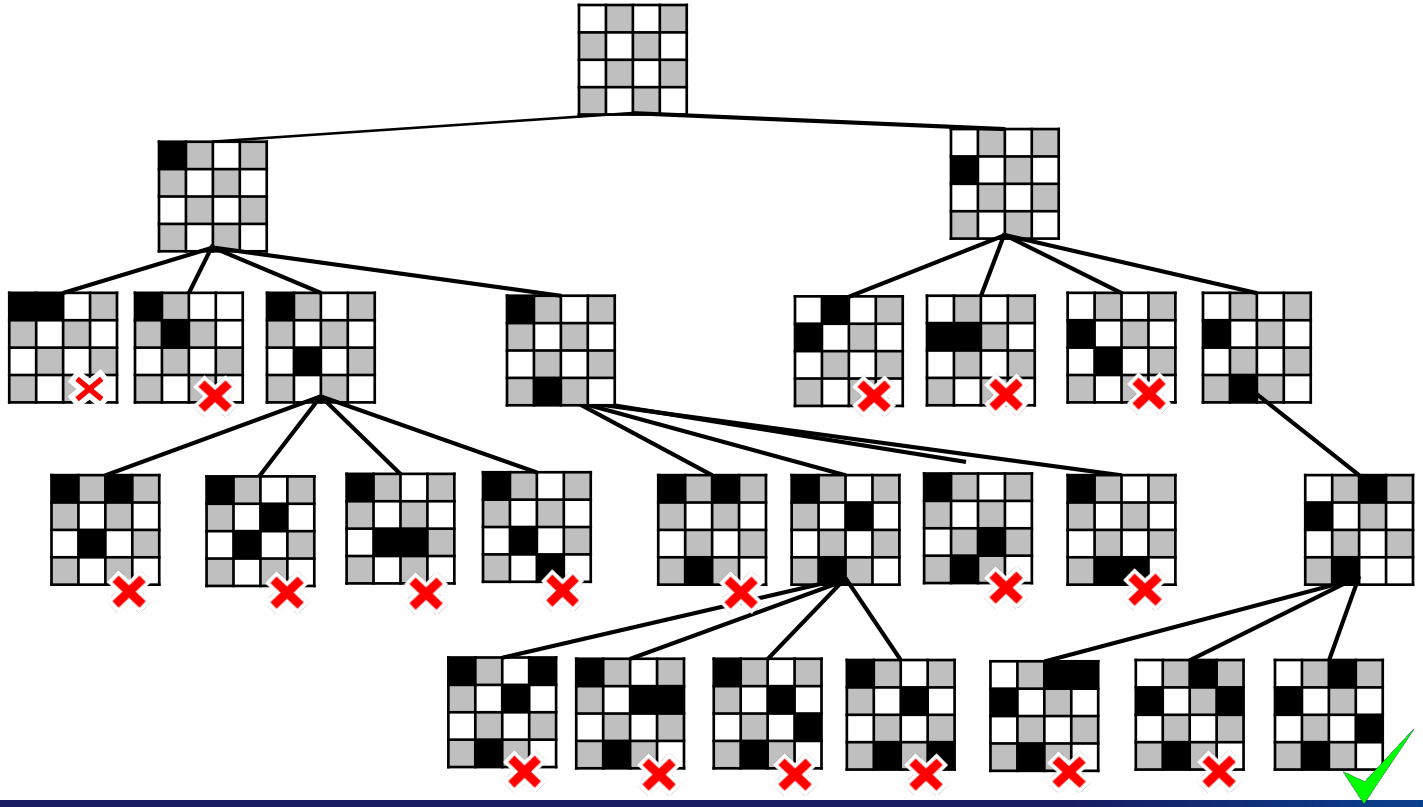
# Backtracking Search

Backtracking search: Do not waste time searching when constraints have already been violated




- Before generating successors, check for constraint violations
- If yes, backtrack to try something else

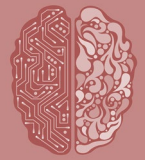
# Example (4-Queens)



Question to think:

- Please think about whether there are ways for further improving search efficiency.

# Heuristics for CSPs



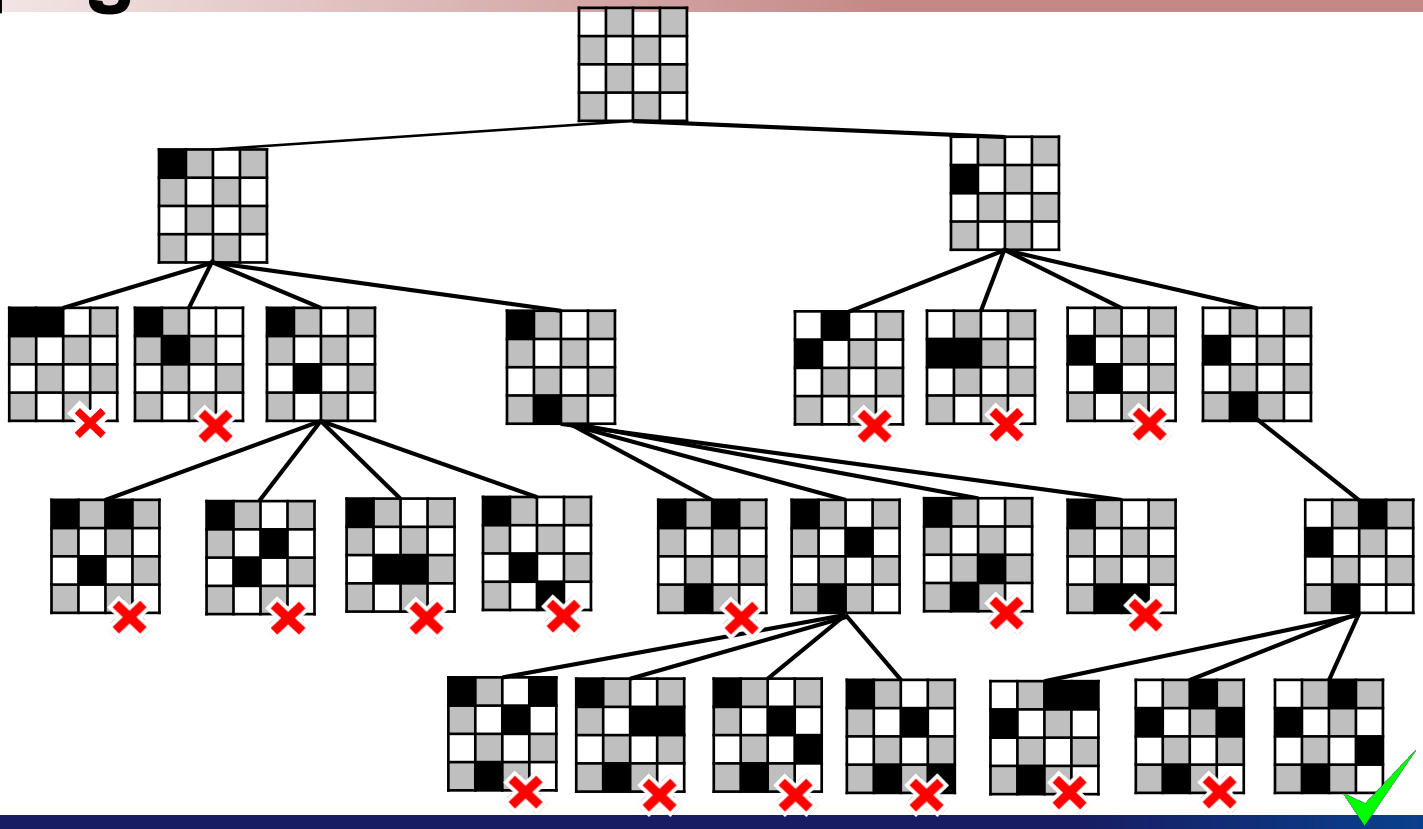
Plain backtracking is an uninformed algorithm!!

More intelligent search that takes into consideration

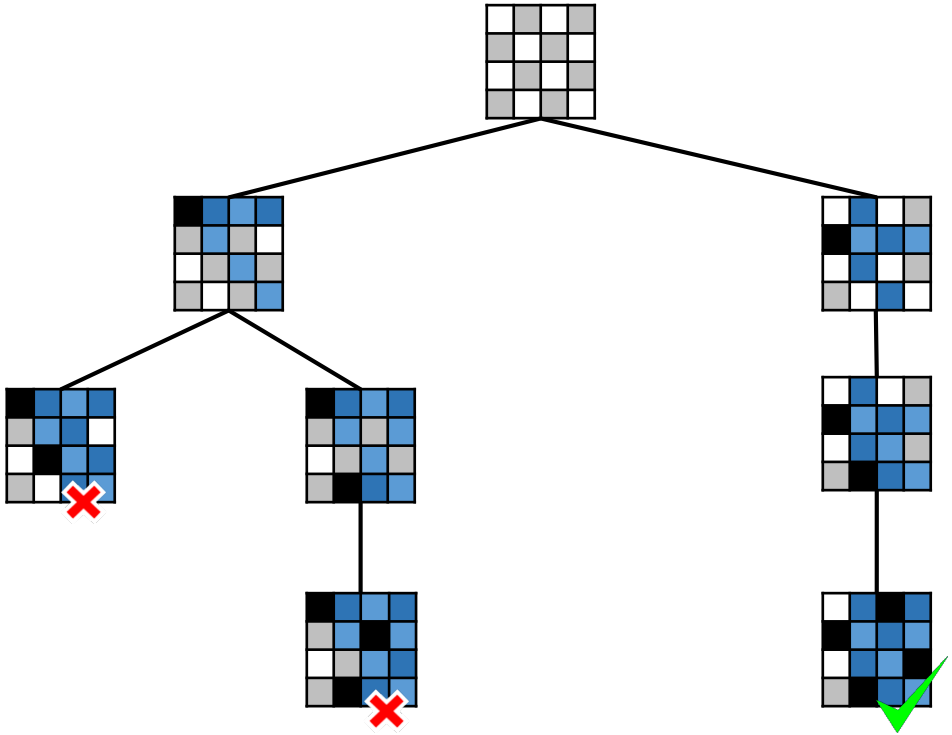
- Which variable to assign next
- What order of the values to try for each variable
- Implications of current variable assignments for the other unassigned variables
  - forward checking and **constraint propagation**

**Constraint propagation:** propagating the implications of a constraint on one variable onto other variables

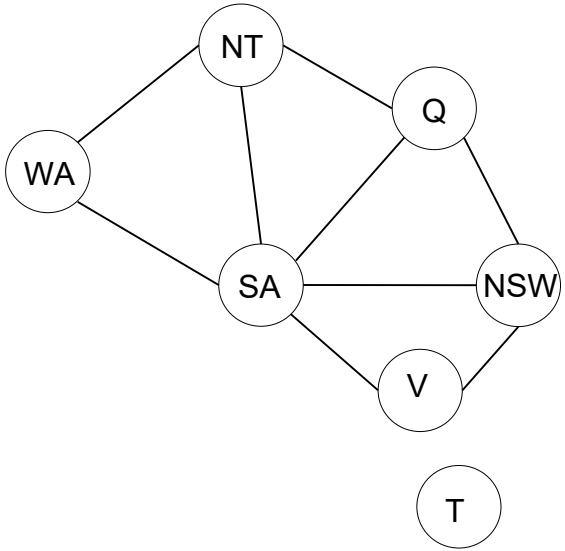
# Example (4-Queens) without Constraint Propagation



# Search Tree of 4-Queens with Constraint Propagation

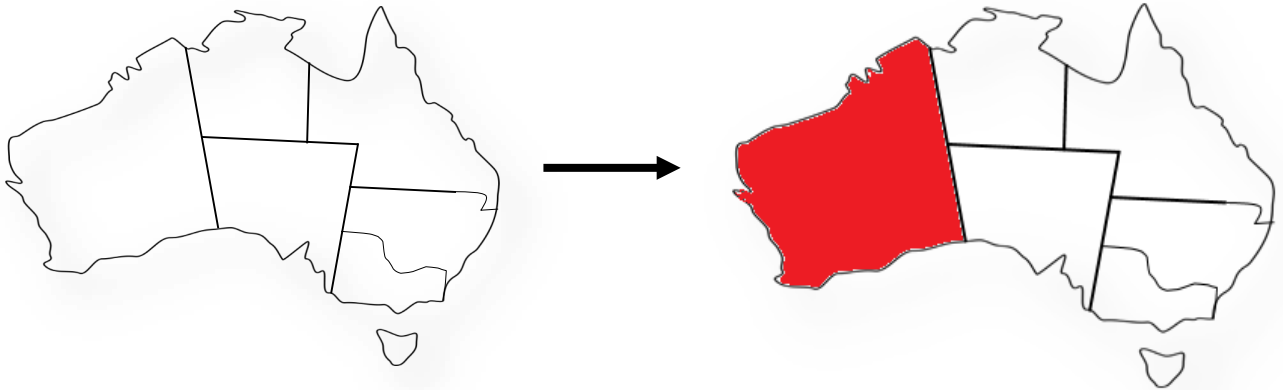


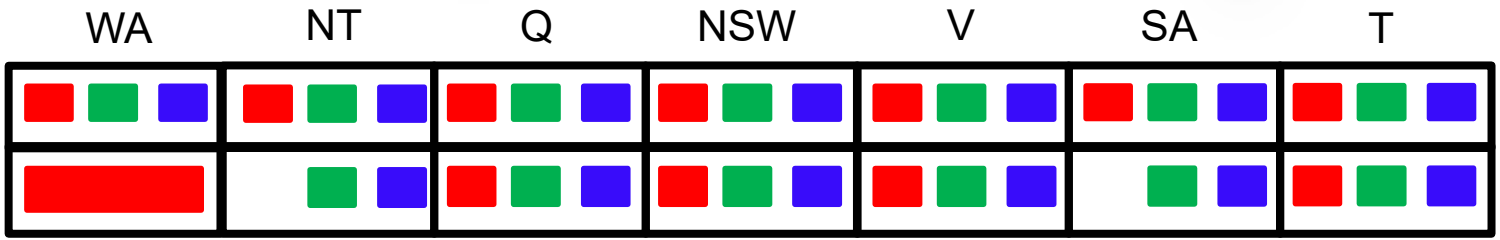
# Example (Map Colouring)

# Example (Map Colouring)...

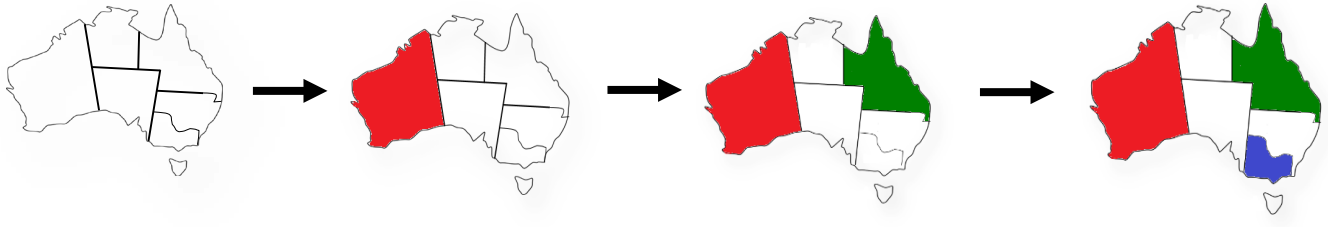

# Example (Map Colouring)...



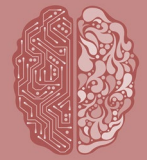


# Example (Map Colouring)...

# Example (Map Colouring)...

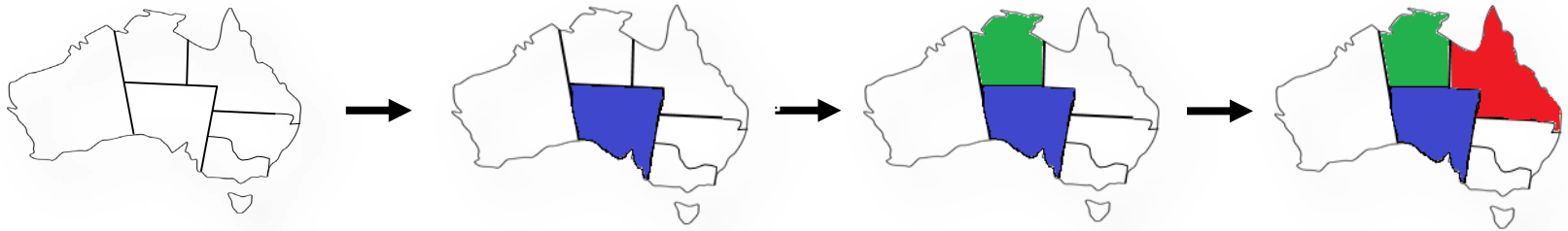


# Most Constrained Variable



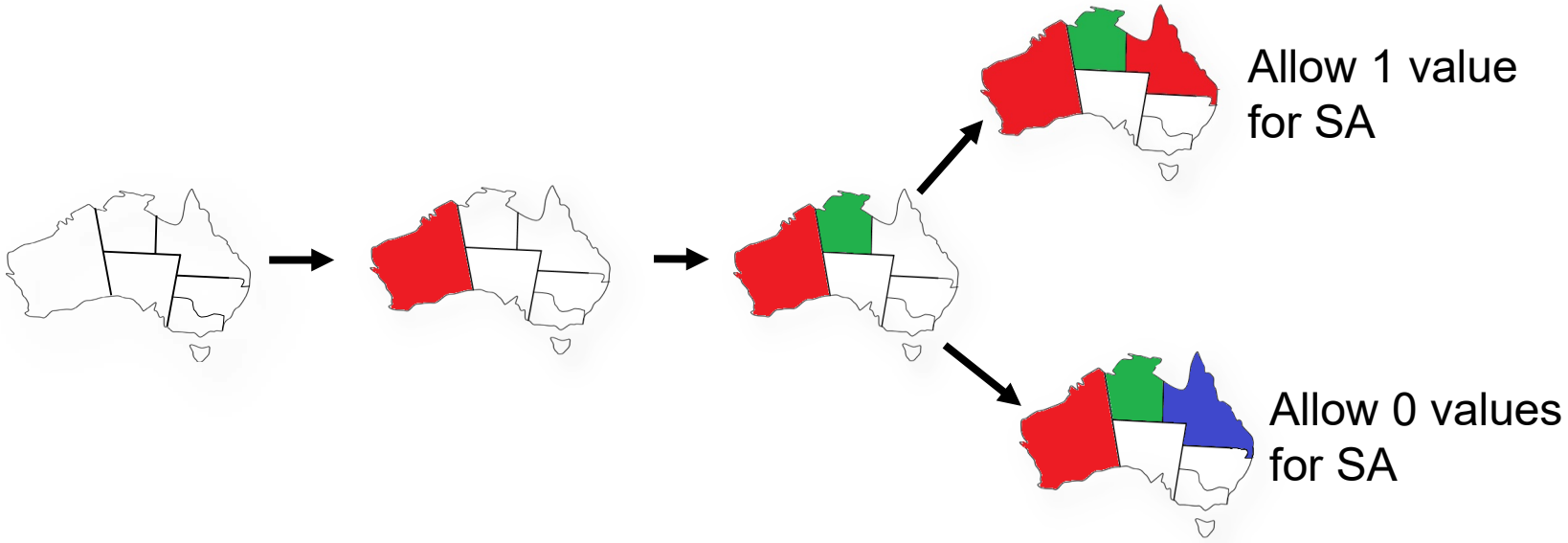
Or minimum remaining values (MRV) heuristic

**Example: map colouring**



To reduce the branching factor on future choices by selecting the variable that is involved in the **largest number of constraints** on unassigned variables.

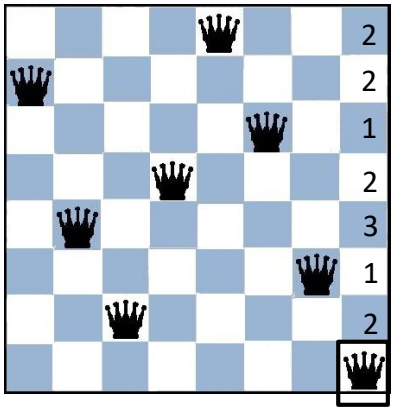
# Least Constraining Value

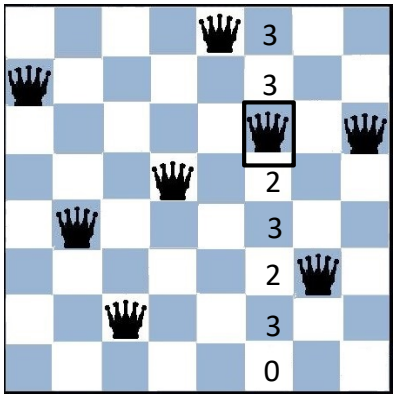


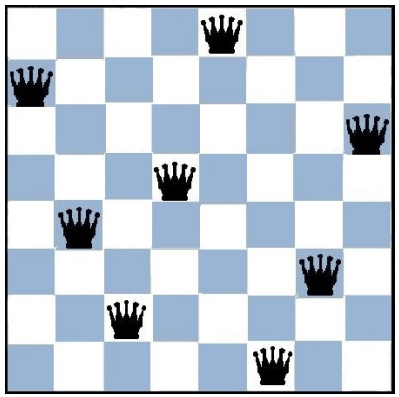

Choose the value that  leaves maximum flexibility for subsequent variable assignments

# Min-Conflicts Heuristic (8-queens)

- A local heuristic search method for solving CSPs
- Given an initial assignment, selects a variable in the scope of a violated constraint and assigns it to the value that minimises the number of violated constraints



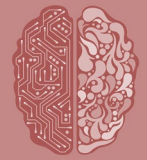



Question to think:

- Please think the intuitions for why the above ideas can improve search efficiency.

# Games as Search Problems



Abstraction

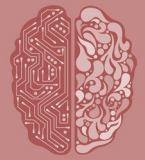
- Ideal representation of real world problems
  - e.g. board games, chess, go, etc. as an abstraction of war games
  - Perfect information, i.e. fully observable
- Accurate formulation: state space representation

Uncertainty

- Account for the existence of hostile agents (players)
  - Other agents acting so as to diminish the agent's well-being
  - Uncertainty (about other agents' actions):
    - not due to the effect of non-deterministic actions
    - not due to randomness

    → Contingency problem

# Games as Search Problems...



Complexity

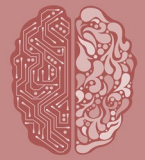
- Games are abstract but not simple
  - e.g. chess: average branching factor = 35, game length > 50

  → complexity = $35^{50}$ (only $10^{40}$ for legal moves)

- Games are usually time limited
  - Complete search (for the optimal solution) not possible

  → uncertainty on actions desirability
  - Search efficiency is crucial

# Types of Games

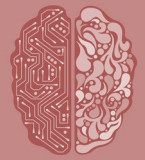


| | Deterministic | Chance |
|---|---|---|
| **Perfect information** | Chess, Checkers, Go, Othello | Backgammon, Monopoly |
| **Imperfect information** | | Bridge, Poker, Scrabble, Nuclear war |

Perfect information

- each player has complete information about his opponent's position and about the choices available to him

# Game as a Search Problem

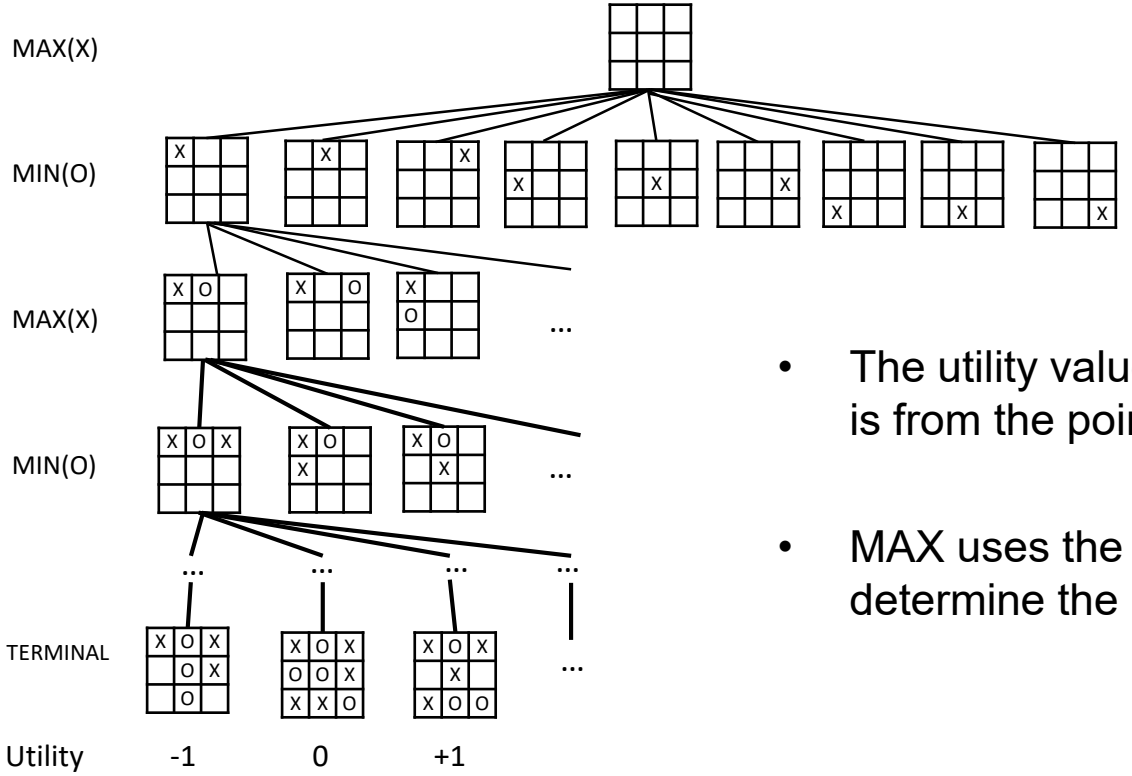


- Initial state: initial board configuration and indication of who makes the first move
- Operators: legal moves
- Terminal test: determines when the game is over
  - states where the game has ended: terminal states
- Utility function (payoff function): returns a numeric score to quantify the outcome of a game

**Example: Chess**

Win (+1), loss(-1) or draw (0)

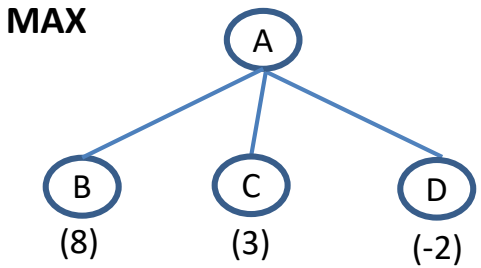
# Game Tree for Tic-Tac-Toe



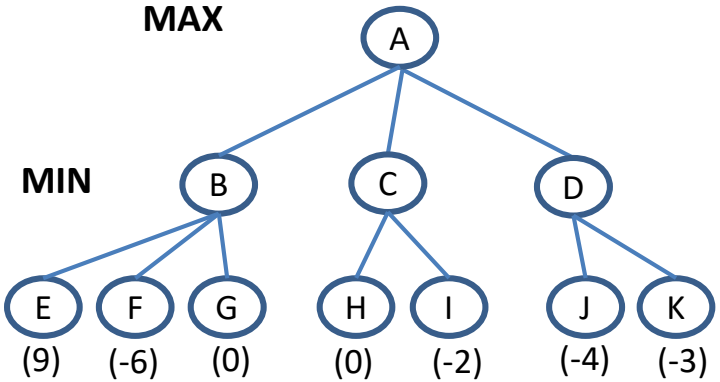

- The utility value of the terminal state is from the point of view of MAX
- MAX uses the search tree to determine the best move

# What Search Strategy?

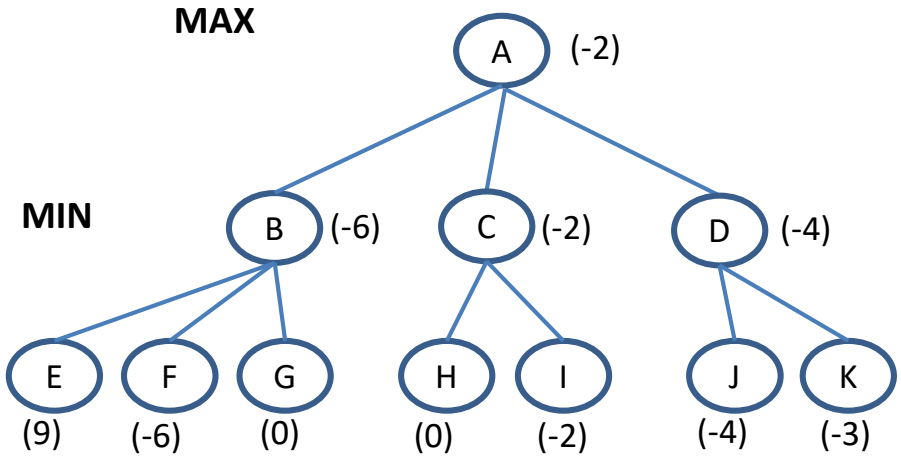
One-play



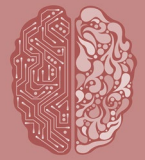

Two-play



# What Search Strategy?...



# Minimax Search Strategy



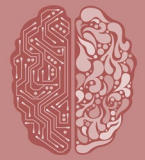
Search strategy

- Find a sequence of moves that leads to a terminal state (goal)

Minimax search strategy

- Maximise one's own utility and minimise the opponent's
  - Assumption is that the opponent does the same

# Minimax Search Strategy



**3-step process**

1. Generate the entire game tree down to terminal states

2. Calculate utility
   a) Assess the utility of each terminal state
   b) Determine the best utility of the parents of the terminal state
   c) Repeat the process for their parents until the root is reached

3. Select the best move (i.e. the move with the highest utility value)

# Perfect Decisions by Minimax Algorithm

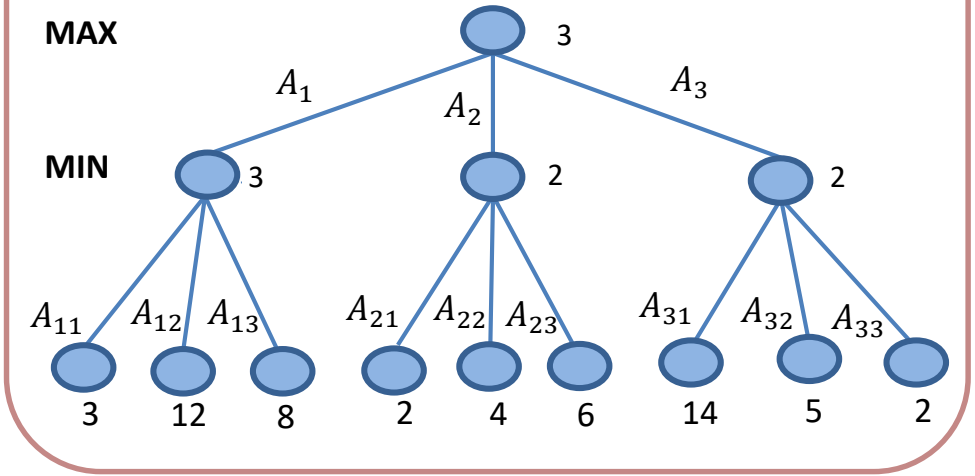
Perfect decisions: no time limit is imposed

- generate the complete search tree

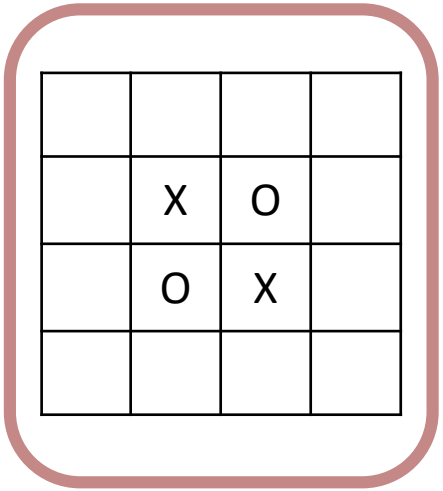
Two players: MAX and MIN

- Choose move with best achievable payoff against best play
- MAX tries to max the utility, assuming that MIN will try to min it

**Example**



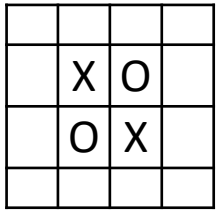
# Othello 4



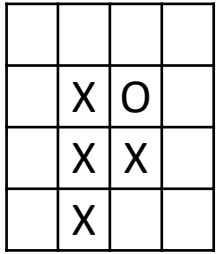

- A player can place a new piece in a position if there exists at least one straight (horizontal, vertical, or diagonal) occupied line between the new piece and another piece of the same kind, with one or more contiguous pieces from the opponent player between them
- After placing the new piece, the pieces from the opponent player will be captured and become the pieces from the same Player
- The player with the most pieces on the board wins

# `X' plays first

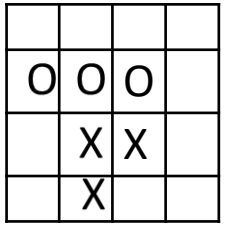
X considers the game now

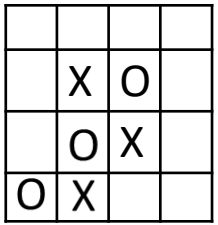


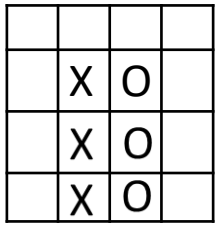

O considers the game now



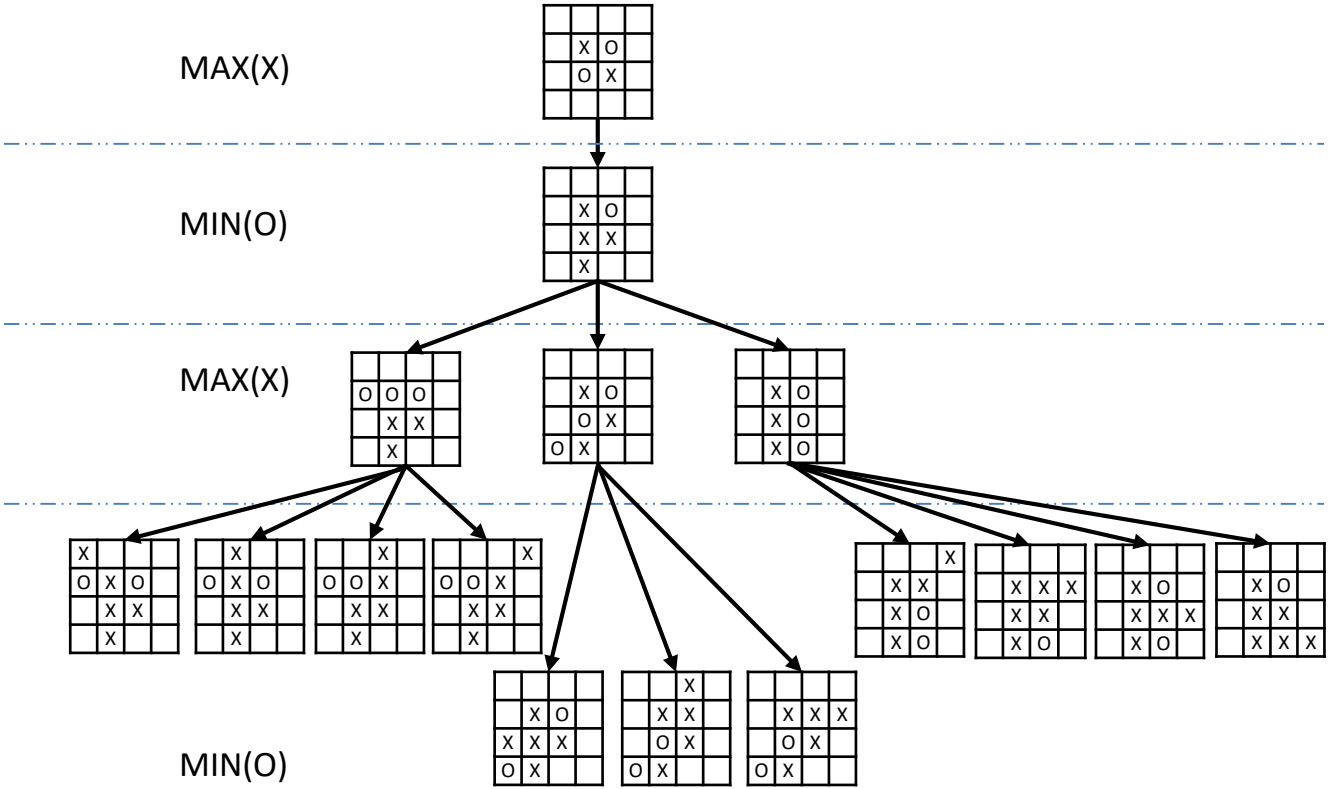

X considers the game now



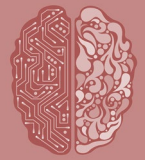



# Game Tree Othello 4



# Imperfect Decisions



For chess, branching factor ≈ 35, each player typically makes 50 moves → for the complete game tree, need to examine $35^{100}$positions

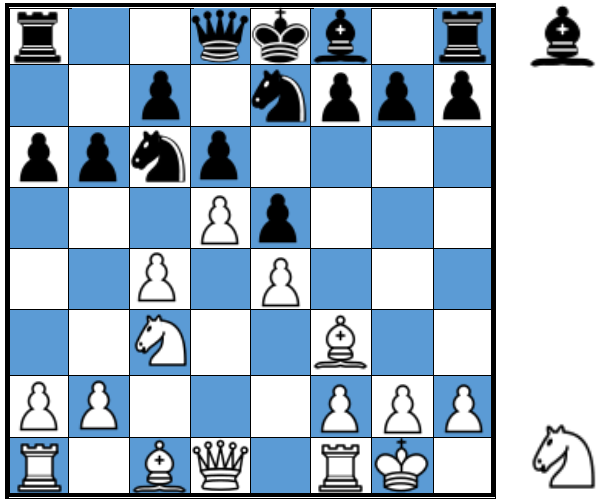
Time/space requirements → complete game tree search is intractable → impractical to make perfect decisions

Modifications to minimax algorithm

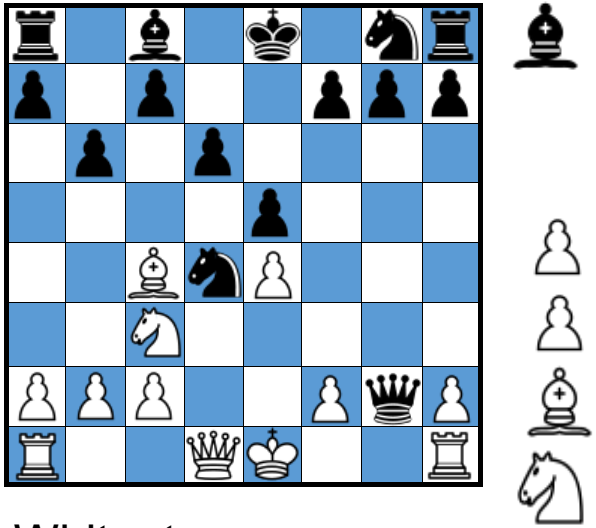
1. replace utility function by an estimated desirability of the position
    - Evaluation function
2. partial tree search
    - E.g., depth limit
    - Replace terminal test by a cut-off test

# Evaluation Functions

Returns an estimate of the expected utility of the game from a given position



Black: to move
White: slightly better



White: to move
Black: winning