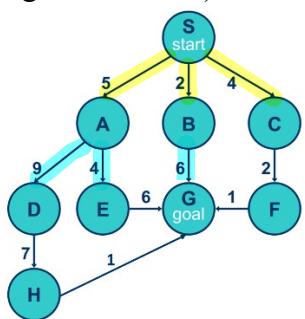
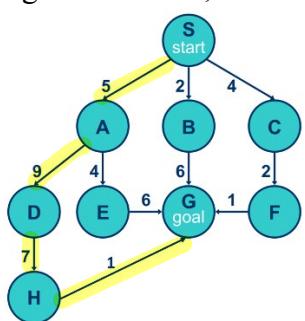


- Autonomous Agents rely entirely on built-in knowledge about the environment.  
a)True b)False
- Iterative Deepening Search is NOT complete.  
a)True b)False
- A Constraint Satisfaction Problems (CSP) consists of a set of variables to which values can be assigned such that a set of constraints over the variables is respected.  
a)True b)False
- What path would Breadth-First Search return for this search problem (initial state is S and the goal state is G)?



Breadth first search will go down ONE level  
then check if its the goal  
(cost not important)

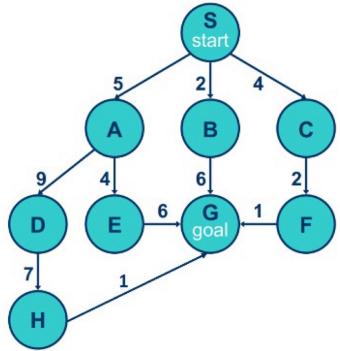
- S-A -E-G
- S-B-G
- S-C-F-G
- S-A-D-H-G
- What path would Depth-First Search return for this search problem (initial state is S and the goal state is G, searching from left to right)?



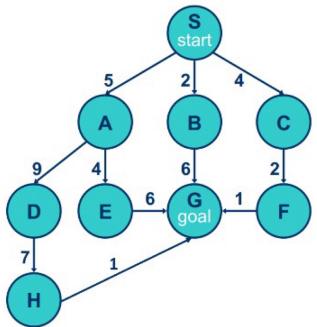
Starting from the left, it will go all the way down to the deepest node, which just so happens to be the goal

(cost not important)

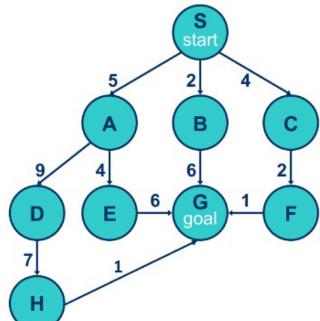
- S-A -E-G
- S-B-G
- S-C-F-G
- S-A-D-H-G



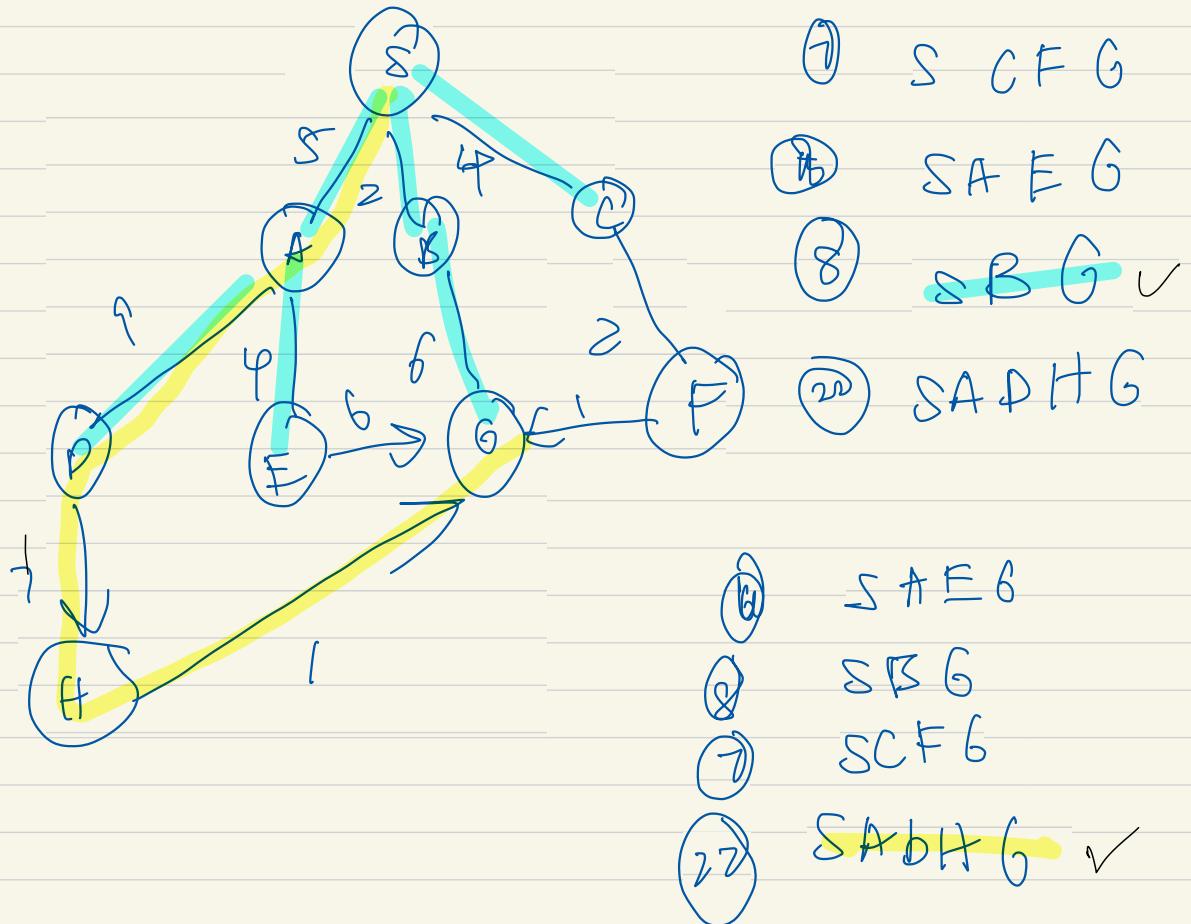
if its Uniform-Cost Search,  
the path will be  $S - B - G$   
(Cost important)



if its a Greedy search  
depends on node cost,  $h(n)$ ,  
not edge cost  
(smallest node cost)  
(Breath-first search traversal)  
CAN'T DECIDE HERE cause  
NO NODE COST



if its a A\* Search,  
Depends on both node &  
edge cost,  
 $f(n) = g(n) + h(n)$



① S CFG

② SA E G

③ S B G ✓

④ SA D H G

S A E G

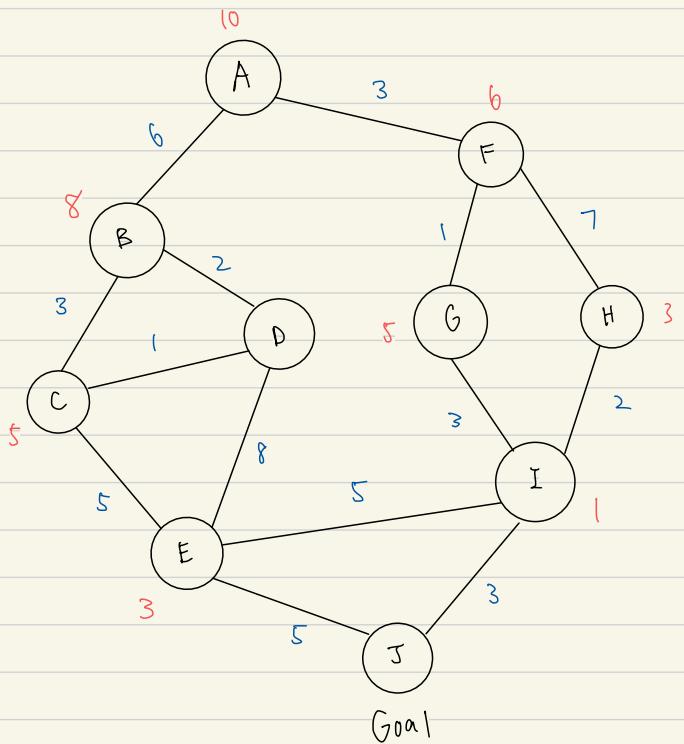
S B G

SC F G

S A D H G ✓

breadth first  
&  
depth first

A\* search : finding shortest path between initial and final state



for Greedy , sum of  $h(n)$

without  $g(n)$

$$f(n) = g(n) + h(n)$$

$n$  : previous node on the path

$g(n)$  : cost of path from start node to  $n$  (sum of visited cells)

$h(n)$  : heuristic that estimates the cost of the (actual cost cannot be calculated  
cheapest path from  $n$  to target node until final cell is reached)

Start at A ; A has nodes B & F

calculate  $F(B)$  &  $F(F)$

$$F(B) = 6 + 8 = 14$$

$$F(F) = 3 + 6 = 9$$

$F(F) < F(B)$  ,  $\therefore F$  is new start node

Start at F ; F has nodes G & H

Calculate  $F(G)$  and  $F(H)$

$$F(G) = (3+1) + 5 = 9$$

$$F(H) = (3+7) + 3 = 13$$

$F(G) < F(H)$  , so choose G as new start node

## Search Methods

### 5 Uninformed Search Methods

Breadth-first search

→ check nodes layer-by-layer  
UCS

→ consider cost of each path

Depth first search

→ As deep as possible

Depth limited }  
Iterative Deepening } Expansion of  
depth first search

### 2 Informed Search Method

Greedy Search - makes use of domain knowledge  
- expand node with smallest estimated cost

A\* - consider both the cost of initial to current state +  
estimate cost of current to goal

\* If edge function is not an overestimate of the 2  
costs, we can guarantee A\* will give optimal solution

BFS, DFS, GS  $\rightarrow$  don't care cost of edges

Based on Lab 7

### Breadth-First Search (BFS)

- Uninformed search
- Focus on graph structure
- NOT cost from initial or distance from goal
- Use FIFO Queue for frontier

### Depth-First Search (DFS)

- Uninformed search
- Focus on graph structure
- NOT cost from initial or distance from goal
- Use LIFO Queue for frontier

### Uniform-Cost Search (UCS)

- Uninformed search
- Focus on graph structure AND cost from start
- NOT distance from goal
- Priority Queue for Frontier

### Greedy Search (GS)

- informed search
- Focus on graph structure AND distance from goal [ $h(n)$ ]
- NOT cost from start
- Priority Queue for Frontier

### A\* search (A\*)

- informed search
- Focus on graph structure, cost from start AND distance from goal
- Priority Queue for Frontier

## Graph Comparison

Theory Quiz FT

	BFS	UCS	DFS	Greedy	A*	Depth limited	Iterative deepening
Complete	Yes	Yes	No	No	Yes	Yes, if $l \geq d$ else no	Yes
Optimal	Yes	Yes	No	No	Yes	No	Yes
Time	$L^d$	$b^d$	$b^m$	$O(b^m)$	Exponential	$b^L$	$b^d$
Space	$b^L$	$b^d$	$b^m$	$O(b^m)$	Exponential	$bL$	$bd$

Accessible : all information about environment is given

Inaccessible : Not all information about environment is given

Deterministic : outcome of movement can be determined (not randomly assigned, etc.)

non deterministic : outcome of movement cannot be determined

Episodic : current action not based on previous action

sequential : current action based on previous action

Static : environment does not change while agent is deliberating

Dynamic : environment changes while agent is deliberating

Discrete : finite number of percepts & actions

continuous : infinite number of percepts & actions

## Properties of greedy best-first search

- Complete? **No** – can get stuck in loops, e.g., Iasi → Neamt → Iasi
- Time?  $O(b^m)$ , (in worst case) but a good heuristic can give dramatic improvement ( $m$  is max depth of search space).
- Space?  $O(b^m)$  -- keeps all nodes in memory.
- Optimal? **No** (not guaranteed to render lowest cost solution).

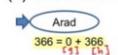
## Properties of A\*

- Complete: **Yes** (unless there are infinitely many nodes with  $f \leq f(G)$ ).
- Time: Exponential.
- Space: Keeps all nodes in memory, so also exponential.
- Optimal: **Yes** (provided  $h$  admissible or consistent).
- Optimally Efficient: **Yes** (no algorithm with the same heuristic is guaranteed to expand fewer nodes).

### Example: Route-finding from Arad to Bucharest

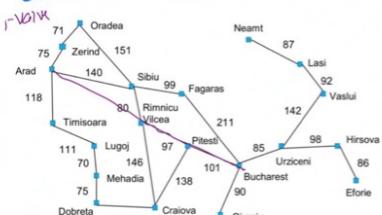
Best-first-search with evaluation function  $g + h$

(a) The initial state

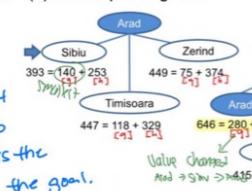


*g is the cost of location to location. h is the distance to the goal.*

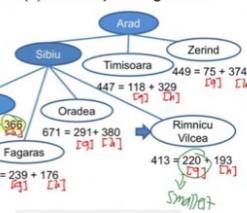
*Value of h*



(b) After expanding Arad



(c) After expanding Sibiu



Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Giurgiu	242
Eforie	161
Fagaras	176
Glurjui	77
Hirsova	151
Lasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	360
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

## 1 Directed Questions

- What is meant by search algorithm *completeness*? **Answer:** If an algorithm is complete, it means that if at least one solution exists then the algorithm is guaranteed find a solution in a finite amount of time.
- What is meant by search algorithm *optimality*? **Answer:** If a search algorithm is optimal, then when it finds a solution it finds the *best* solution.
- What are the advantages of breadth-first search (BFS) over depth-first search (DFS)?  
**Answer:** BFS is complete and optimal, while DFS is not guaranteed to halt when there are loops.
- What is the advantage of DFS over BFS? **Answer:** If  $m$  is the maximum path length and  $b$  is the branching factor, the space complexity for DFS is  $mb$  while for BFS it is  $b^m$ .

3. (10 pts) In general, which is the preferred uninformed search method when there is a large search space and the depth of the solution is unknown? (Mark one blank with "X")

depth-first search     breadth-first search     uniform-cost search  
 depth-limited search     iterative-deepening search



## Introduction to Data Science and Artificial Intelligence

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

Assoc Prof Bo AN

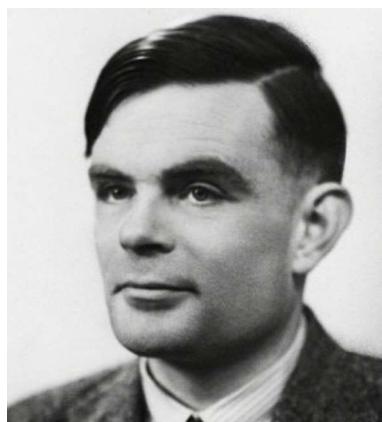
Research area: artificial intelligence,  
computational game theory, optimization  
[www.ntu.edu.sg/home/boan](http://www.ntu.edu.sg/home/boan)  
Email: [boan@ntu.edu.sg](mailto:boan@ntu.edu.sg)  
Office: N4-02b-55



## 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



Turing  
Talk



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

## What is Artificial Intelligence?

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



## Founding Fathers of AI

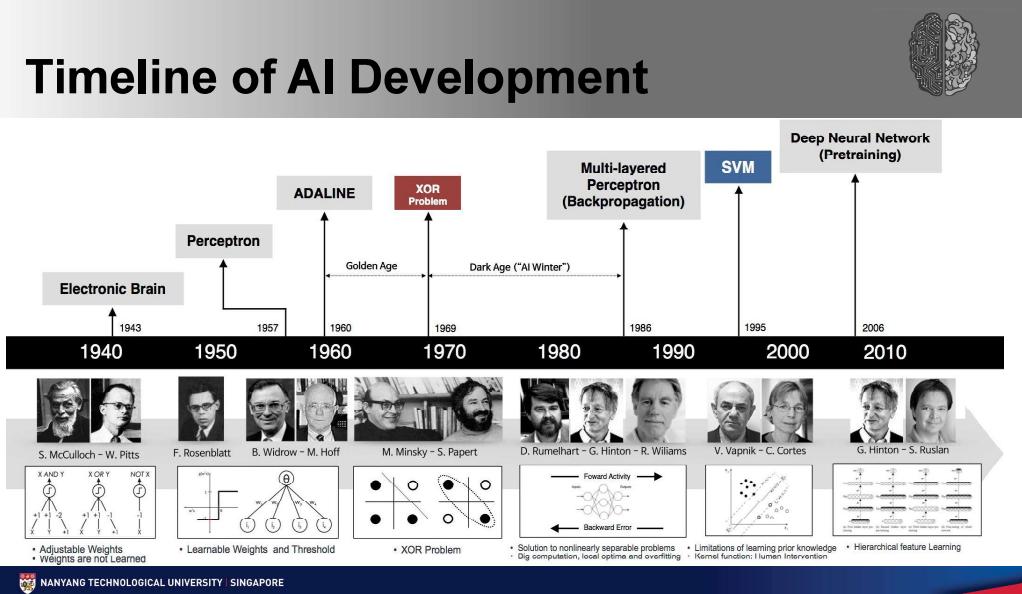
1956 Dartmouth Conference



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



# Timeline of AI Development



# Views of AI

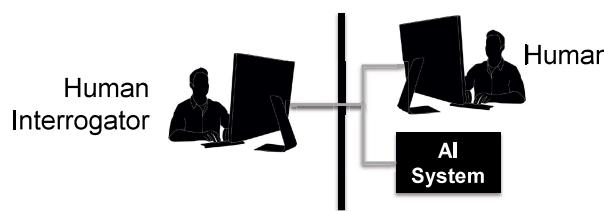
Views of AI fall into four categories:

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

↗ thinking this in computer science

# 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>

# 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

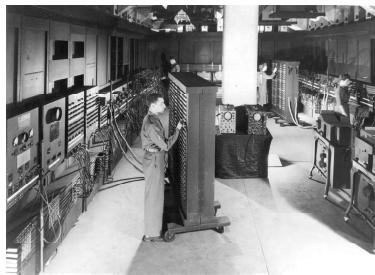


## Thinking Rationally: "Laws of Thought"

- 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?

## Computer Chess

Garry Kasparov VS Deep Blue



ENIAC 1946

VS



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](https://en.wikipedia.org/wiki/Deep_Blue_(chess_computer)#/media/File:Deep_Blue.jpg)



## Acting Rationally: Rational Agents

purpose of  
this course

- **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



## 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)

complete  
information  
game

(based on deep  
learning)

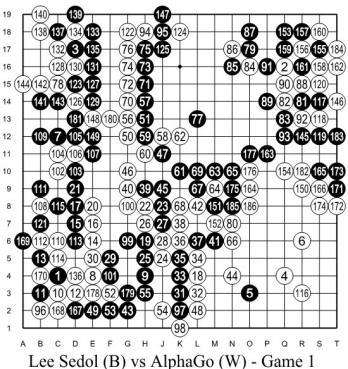


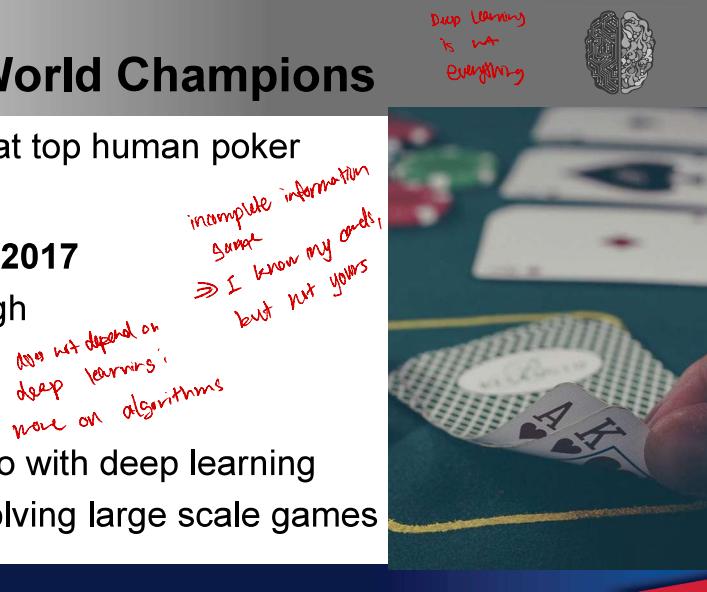
Image source: [https://commons.wikimedia.org/wiki/File:Lee\\_Sedol\\_\(B\)\\_vs\\_AlphaGo\\_\(W\)\\_-\\_Game\\_1\\_-\\_BW.jpg](https://commons.wikimedia.org/wiki/File:Lee_Sedol_(B)_vs_AlphaGo_(W)_-_Game_1_-_BW.jpg)

## 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



NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

## Natural Language Processing

Mi Box Voice Assistant



<https://youtu.be/S38ZjELrMmQ?t=16>

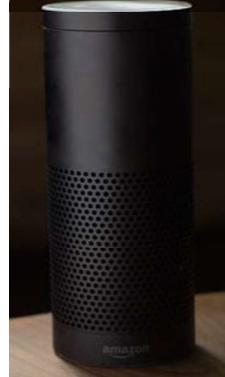
NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Google Assistant



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

Alexa Amazon



## Manufacturing

Tesla

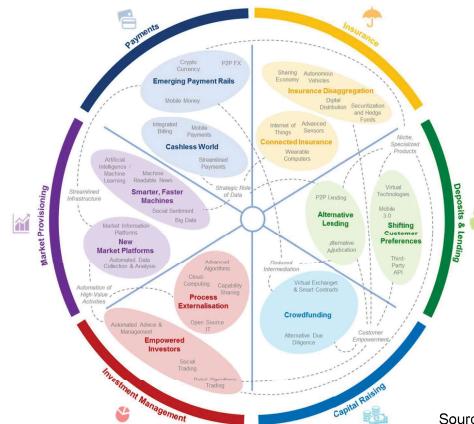
[https://www.youtube.com/watch?v=8\\_lfxPI5ObM](https://www.youtube.com/watch?v=8_lfxPI5ObM)

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



NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

## Artificial Intelligence for Finance



Source: World Economic Forum, 2015

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

# AI for Complex Interactions

Recent AI breakthrough

IMAGENET



NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE



# What's Next?

For AI Complex Interactions



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

Smart City AI



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](http://www.ntu.edu.sg/home/boan)  
Email: boan@ntu.edu.sg  
Office: N4-02b-55

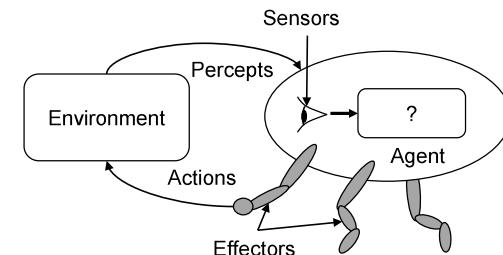


# Agent

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)

observe the environment  
to make changes



NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE





## 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](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

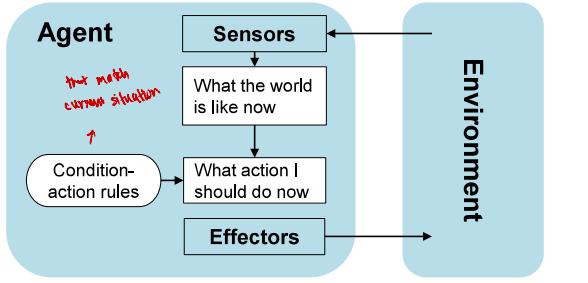
# Simple Reflex Agents



## Example

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

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



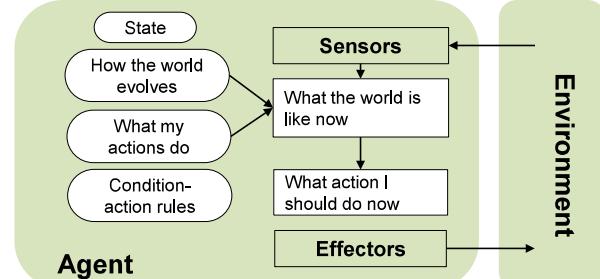
# Reflex Agents with State



## Example

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

- from "humming"*
- Find the rule whose condition matches the current situation (as defined by the percept and the stored internal state)
  - Perform the action associated with that rule



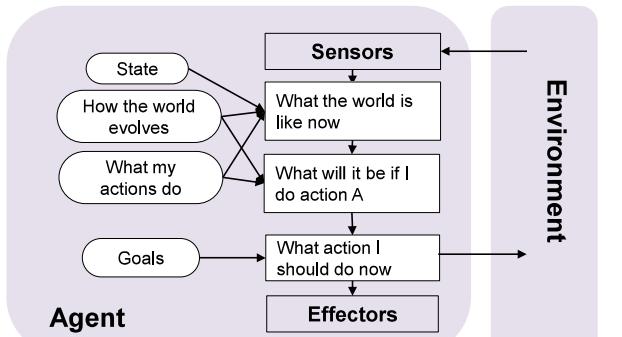
# Goal-Based Agents



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

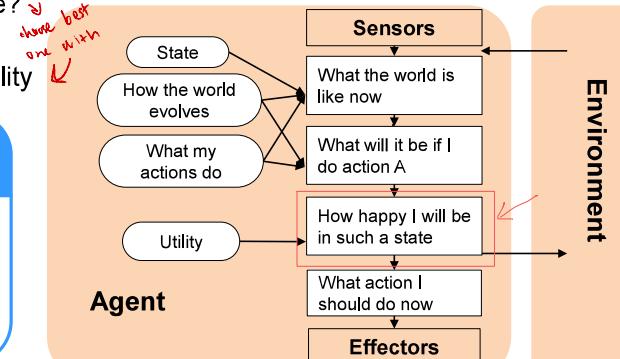


*/ extension of  
goal based agents*

- 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?

## 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

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: 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



Image source: [https://commons.wikimedia.org/wiki/File:Las\\_Vegas\\_slot\\_machines.jpg](https://commons.wikimedia.org/wiki/File:Las_Vegas_slot_machines.jpg)

Accessible?	No
Deterministic?	No
Episodic?	Yes
Static?	Yes

Important to analyse the environment  
whether or not you won previous round, nothing to do with last one

whether or not you won previous round, nothing to do with last one

Question to think:

- Can you find another scenario whose environment is episodic?

Image analysis,  
Robot part picking



## 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



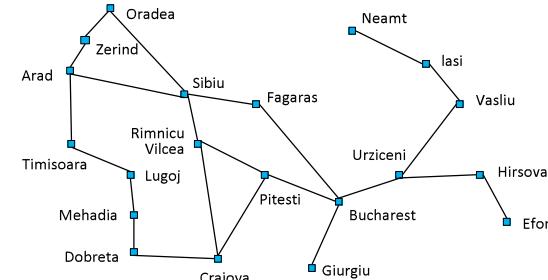
### Steps

1. Goal formulation *or utility*
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](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.

1		
2		
3		
4		
5		
6		
7		
8		

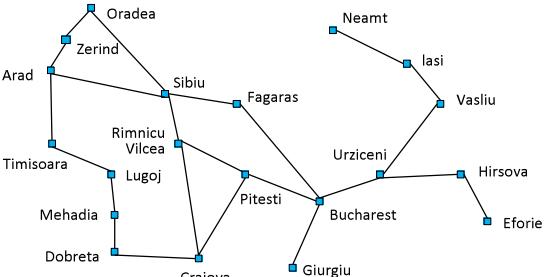
- 8 possible world states
- Possible actions: **left**, **right**, and **suck**
- Goal: clean up all dirt → Two goal states, i.e. {7, 8}

## Well-Defined Formulation



Definition of a problem	The information used by an agent to decide what to do
Specification	<ul style="list-style-type: none"> <li>Initial state</li> <li>Action set, i.e. available actions (successor functions)</li> <li>State space, i.e. states reachable from the initial state           <ul style="list-style-type: none"> <li>Solution path: sequence of actions from one state to another</li> </ul> </li> <li>Goal test predicate           <ul style="list-style-type: none"> <li>Single state, enumerated list of states, abstract properties</li> </ul> </li> <li>Cost function → <small>take action leads cost</small> <ul style="list-style-type: none"> <li>Path cost <math>g(n)</math>, sum of all (action) step costs along the path</li> </ul> </li> </ul>
Solution	A path (a sequence of operators leading) from the Initial-State to a state that satisfies the Goal-Test

## 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.,  $x = \text{"at Bucharest"}$ )
- 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

## Measuring Problem-Solving Performance



### Search Cost

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

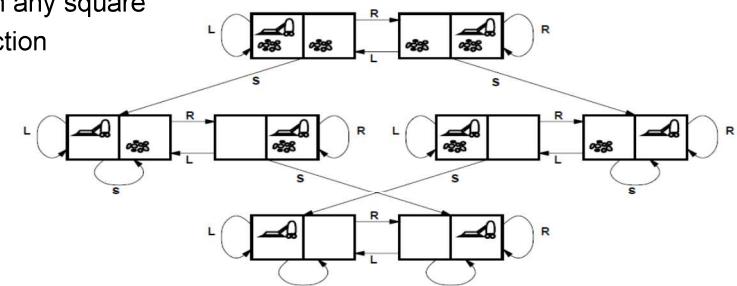
Total cost of problem-solving *how much time you need to find out a solution*

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

## Example: Vacuum World (Single-state Version)



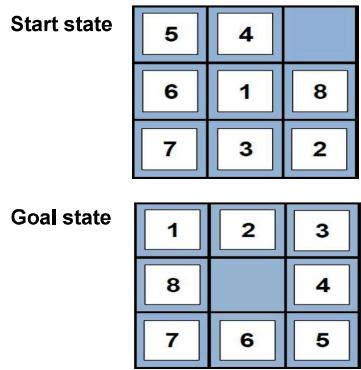
- Initial state:** one of the eight states shown previously
- Actions:** left, right, suck
- Goal test:** no dirt in any square
- Path cost:** 1 per action





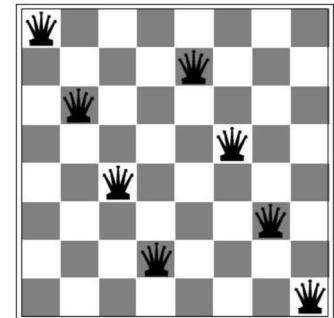
## Example: 8-puzzle

- **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



## 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](https://commons.wikimedia.org/wiki/File:Font_Awesome_5_solid_route.svg)



You can think any AI problem as a search problem

**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](http://www.ntu.edu.sg/home/boan)  
 Email: [boan@ntu.edu.sg](mailto:boan@ntu.edu.sg)  
 Office: N4-02b-55



# Lesson Outline



- Uninformed search strategies → does not make use of domain knowledge
- Informed search strategies → makes use of domain knowledge
  - Greedy search
  - A\* search

# Review: Well-Defined Formulation



Definition of a problem	The information used by an agent to decide what to do
Specification	<ul style="list-style-type: none"><li>• Initial state</li><li>• Action set, i.e. available actions (successor functions)</li><li>• State space, i.e. states reachable from the initial state<ul style="list-style-type: none"><li>• Solution <b>path</b>: sequence of actions from one state to another</li></ul></li><li>• Goal test predicate<ul style="list-style-type: none"><li>• Single state, enumerated list of states, abstract properties</li></ul></li><li>• Cost function<ul style="list-style-type: none"><li>• Path cost <math>g(n)</math>, sum of all (action) step costs along the path</li></ul></li></ul>
Solution	A path (a sequence of operators leading) from the <b>Initial-State</b> to a state that satisfies the <b>Goal-Test</b>

# Search Algorithms



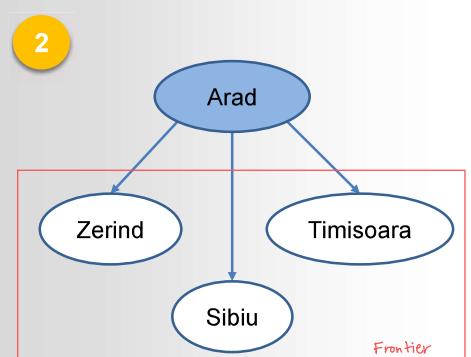
- Exploration of state space by generating successors of already-explored states
- Frontier: candidate nodes for expansion
- Explored set → stores the states that have been expanded



# Search Algorithms

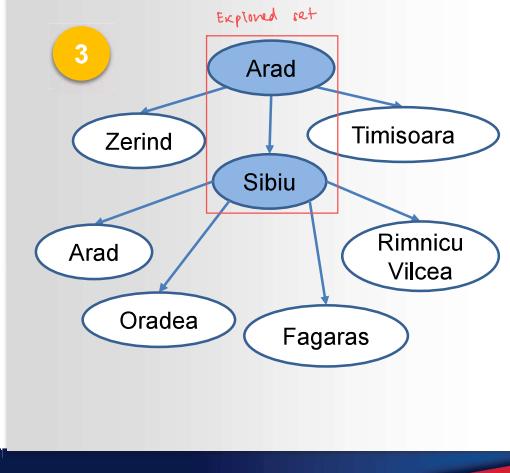


- Exploration of state space by generating successors of already-explored states
- Frontier: candidate nodes for expansion
- Explored set



## Search Algorithms

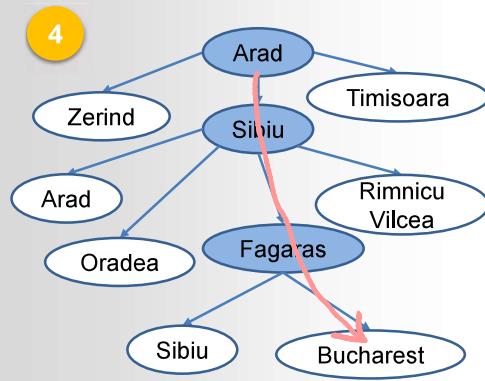
- Exploration of state space by generating successors of already-explored states
  - Frontier: candidate nodes for expansion
  - Explored set



## Search Algorithms

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

*Tree Property:* for any 2 nodes, there is only 1 path



## Search Strategies

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

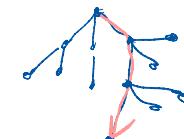
Completeness	Does it always find a solution if one exists?
Time Complexity	How long does it take to find a solution: the number of nodes generated
Space Complexity	Maximum number of nodes in memory
Optimality	Does it always find the best (least-cost) solution?

Search costs

## Search Strategies

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

*Up in this case*  
computed by number  
of nodes in tree  
divide by the no. of  
branches / edges





## 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

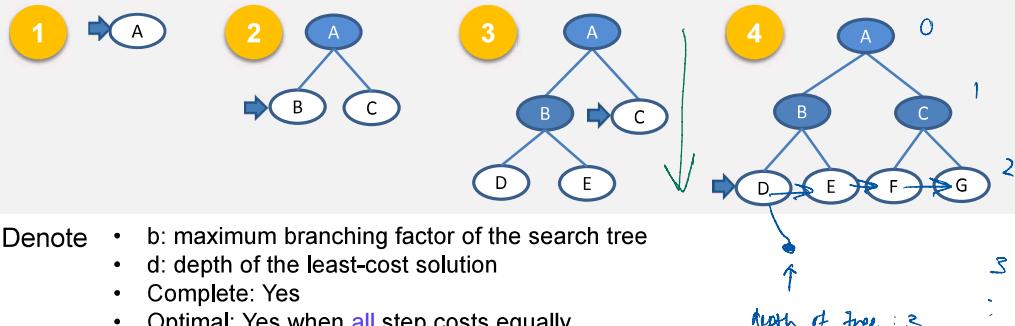
*Greedy*

*A\**



## Breadth-First Search

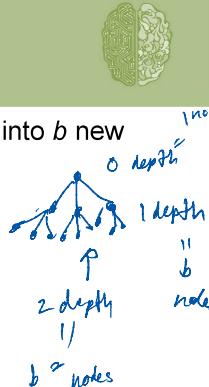
Expand **shallowest** unexpanded node which can be implemented by a First-In-First-Out (FIFO) queue



## Complexity of BFS

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

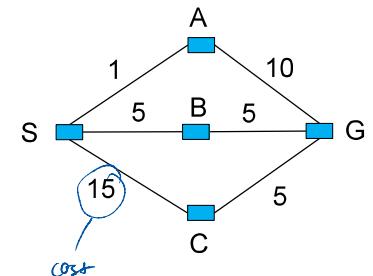
Depth	Nodes	Time	Memory
0	1	1 millisecond	100 bytes
2	111	0.1 seconds	11 kilobytes
4	11111	11 seconds	1 kilobytes
6	$10^6$	18 minutes	111 megabytes
8	$10^8$	31 hours	11 gigabytes
10	$10^{10}$	128 days	1 terabyte
12	$10^{12}$	35 years	111 terabytes
14	$10^{14}$	3500 years	11111 terabytes



## Uniform-Cost Search

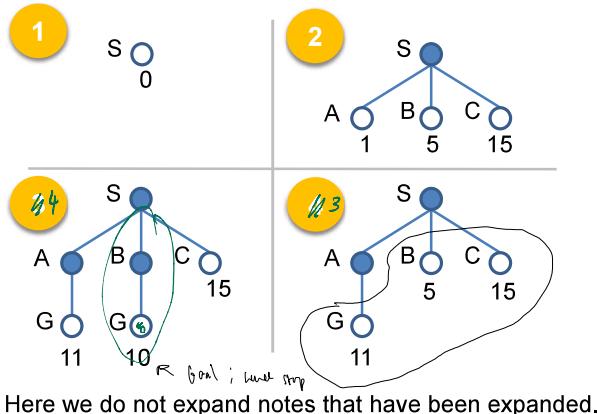
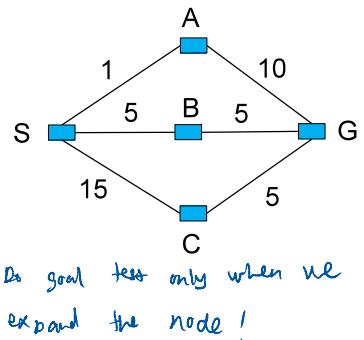
To consider **edge costs**, expand unexpanded node with the **least** path cost  $g$

- 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  $g(n) = \text{Depth}(n)$





## Uniform-Cost Search

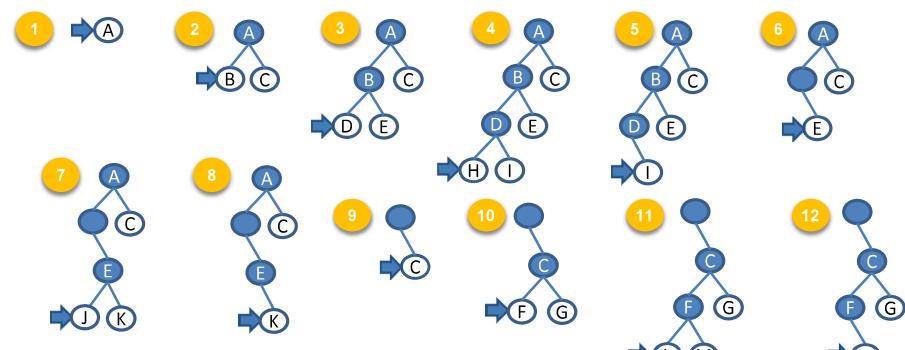


## Uniform-Cost Search

Complete	Yes
Time	# of nodes with path cost $g \leq$ cost of optimal solution (eqv. # of nodes pop out from the priority queue)
Space	# of nodes with path cost $g \leq$ 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



more memory-efficient than BFS

loop minor  
cycle issues

## Depth-First Search

Denote

- m: maximum depth of the state space

Complete	<ul style="list-style-type: none"> <li>infinite-depth spaces: No</li> <li>finite-depth spaces with loops: No           <ul style="list-style-type: none"> <li>with repeated-state checking: Yes</li> </ul> </li> <li>finite-depth spaces without loops: Yes</li> </ul>
Time	$O(b^m)$ If solutions are dense, may be much faster than breadth-first
Space	$O(bm) \rightarrow b \times m$
Optimal	No

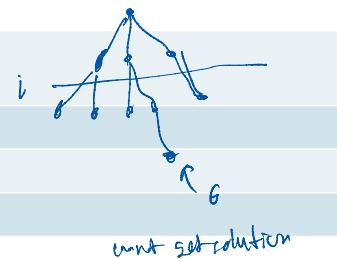


## Depth-Limited Search

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

but we don't know  
the best limit  $\rightarrow$  hence use iterative deepening



## Iterative Deepening Search

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

Limit = 0



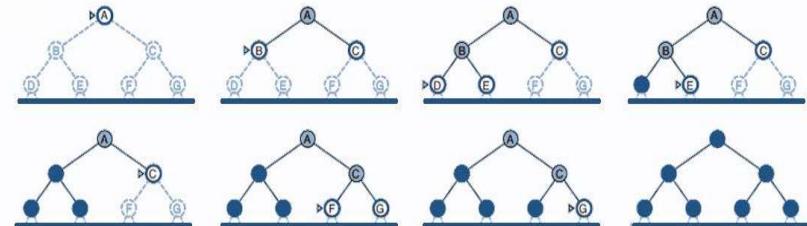
Limit = 1



## Iterative Deepening Search

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

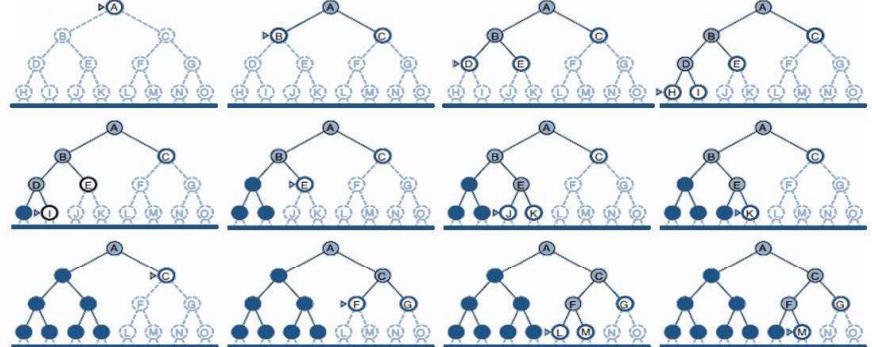
Limit = 2



## 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  $\infty$  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? YES

complete  $\rightarrow$  guaranteed or not?  
 optimal  $\rightarrow$  best solution found or not?  
 $\therefore$  if optimal, it is also complete



## General Search

Uninformed search strategies

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

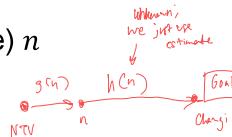
Informed search strategies

- 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)

$A^*$  = Greedy + Uniform Cost Search



## Greedy Search

Expands the node that appears to be closest to goal

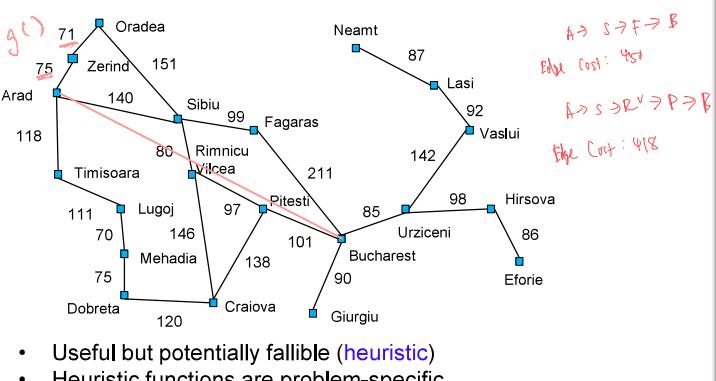
- Evaluation function  $h(n)$ : estimate of cost from  $n$  to *goal*
- Function Greedy-Search(problem) returns solution
  - Return Best-First-Search(problem,  $h$ ) //  $h(goal) = 0$

**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



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

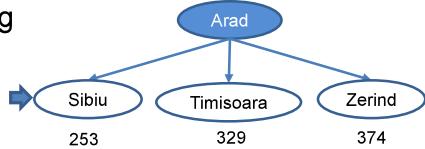


Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Efoire	161
Fagaras	176
Giurgiu	77
Hirsova	151
Lasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



## Example

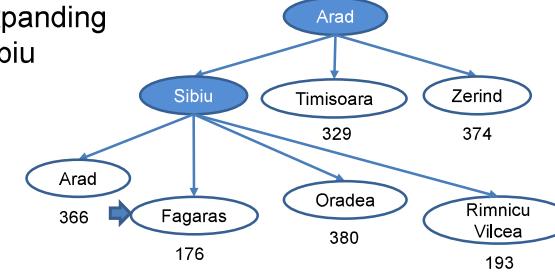
- b) After expanding Arad



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

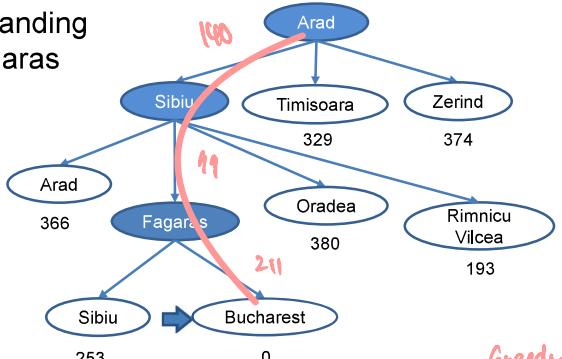


Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Efoire	161
Fagaras	176
Giurgiu	77
Hirsova	151
Lasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

## Example

Greedy

- d) After expanding Fagaras

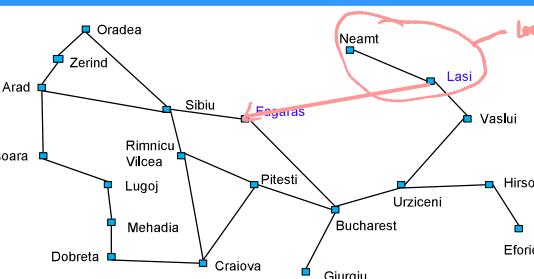


Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Efoire	161
Fagaras	176
Giurgiu	77
Hirsova	151
Lasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

## 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

↓  
optimal locally, not globally

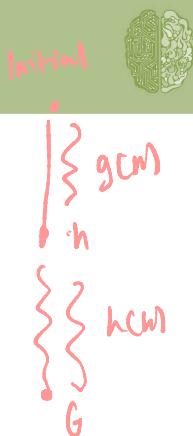
look at worst case

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

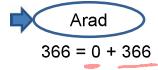
- $f(n)$ : estimated **total** cost of path through  $n$  to goal (**Whole Life**)
  - If  $g = 0 \rightarrow$  greedy search;     If  $h = 0 \rightarrow$  uniform-cost search
- Function  $A^*$  Search(problem) returns solution
  - Return  $\text{Best-First-Search}(\text{problem}, f)$



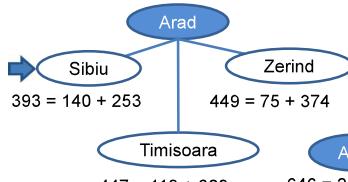
## Example: Route-finding from Arad to Bucharest

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

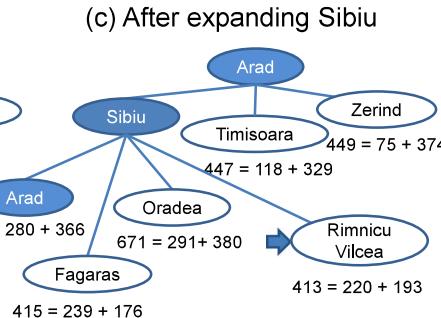
(a) The initial state



(b) After expanding Arad



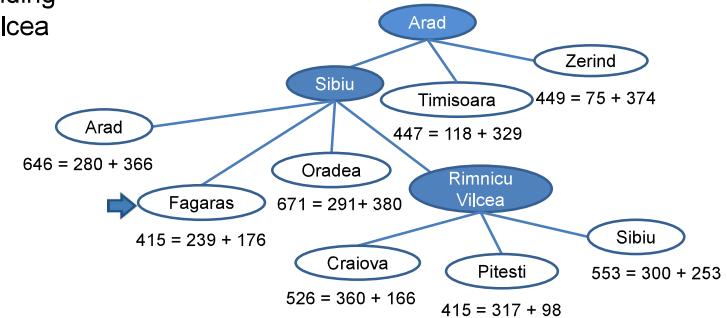
(c) After expanding Sibiu



## Example: Route-finding from Arad to Bucharest

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

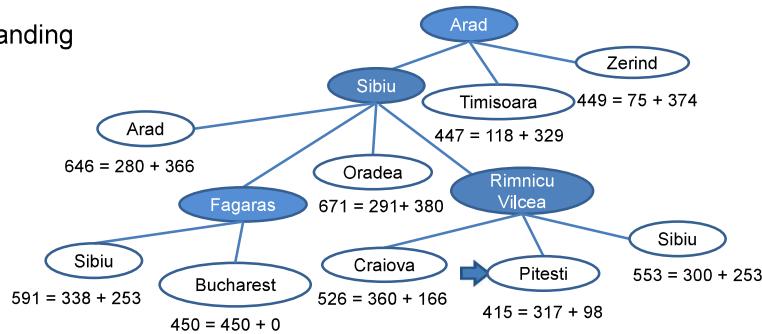
(d) After expanding Rimnicu Vilcea



## Example: Route-finding from Arad to Bucharest

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

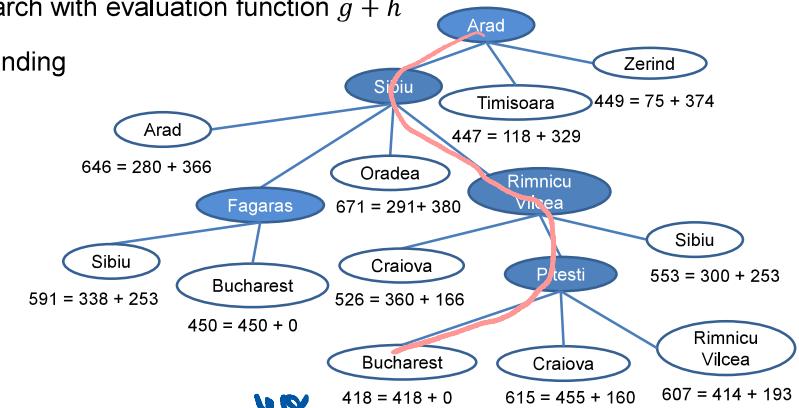
(e) After expanding Fagaras



## Example: Route-finding from Arad to Bucharest

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

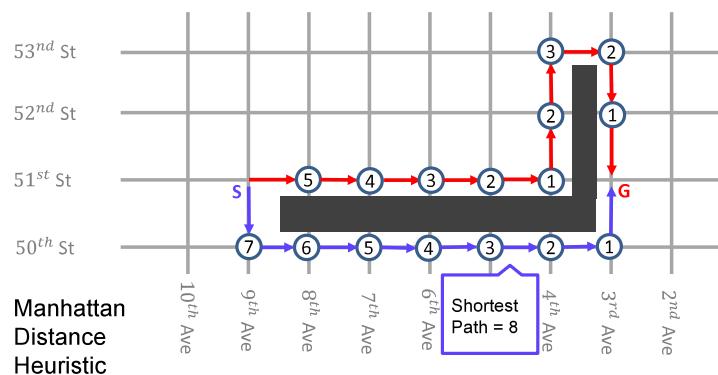
(f) After expanding Pitesti



Greedy

YD

## Example: Route-finding in Manhattan



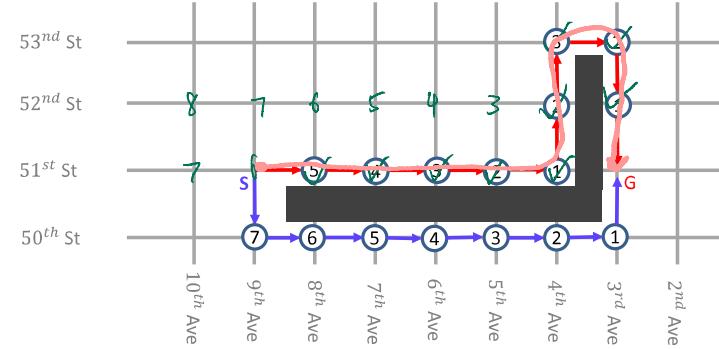
Manhattan Distance Heuristic

## Example: Route-finding in Manhattan (Greedy)

stuck though

cost: 10

Not optimal

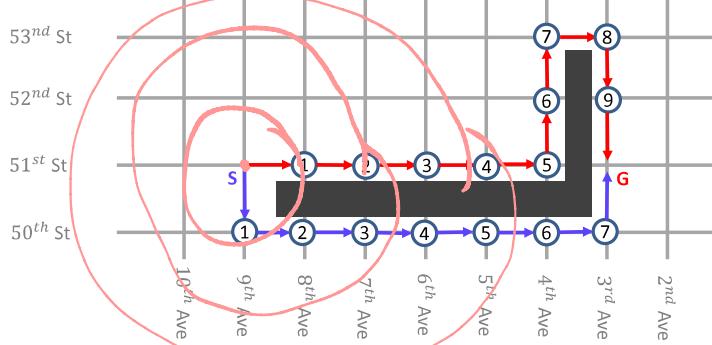


## Example: Route-finding in Manhattan (UCS)



Cost: 8

but inefficient  
as it schedules  
a lot of  
nodes;  
waste memory

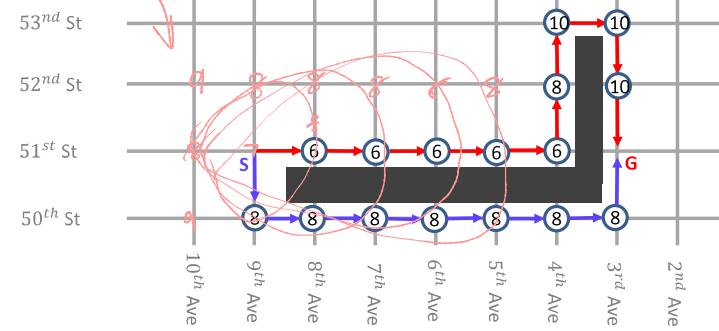


## Example: Route-finding in Manhattan (A\*)

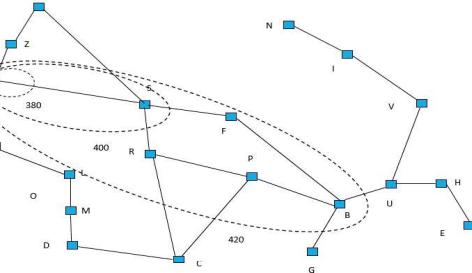


Does not check  
the nodes here like UCS ; only the ones closer to  
goal

smaller  
time &  
space  
complexity



# 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

## 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](http://www.ntu.edu.sg/home/boan)  
Email: [boan@ntu.edu.sg](mailto:boan@ntu.edu.sg)  
Office: N4-02b-55



## Lesson Outline



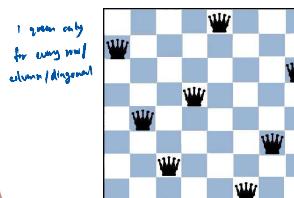
- Constraint Satisfaction
  - Backtracking search
  - Forward checking and constraint propagation
- Game Playing
  - Games as search problems
  - Minimax search strategy
  - Evaluation functions

## 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 → *mother board*
- 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

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

- 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.
  - $D \neq E$ ,  $D \neq M$ ,  $D \neq N$ , etc.
  - $M \neq 0$ ,  $S \neq 0$  (**unary** constraints: concern the value of a single variable)



## Example: Minesweeper



A	B	C
J	2	D
I	3	E
H	G	F

- Variables: The cells
- Domains:  $\{0; 1\}$  representing {safe, mined}
- Constraints: Each cell has a number  $m \in \{1, \dots, 8\}$  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  $C_i$
- Domains: {Red, Blue, Green}
- Constraints:  $C_1 \neq C_2, C_1 \neq C_5$ , 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

e.g. 5 states (water)  
 $(x_1, x_2, x_3, x_4, x_5)$



# Applying Standard Search



**Question:** How to represent constraints?

**Answer:** Explicitly (e.g.,  $D \neq E$ )

## Example

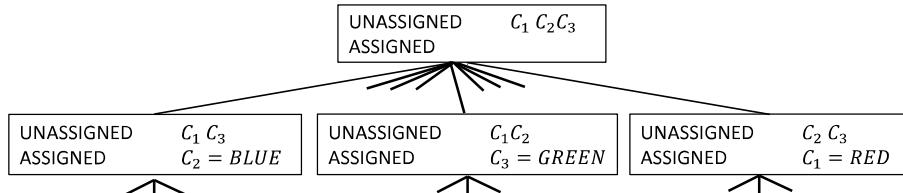
- Row the 1<sup>st</sup> queen occupies:  $V_1 \in \{1, 2, 3, 4, 5, 6, 7, 8\}$   
(similarly, for  $V_2$ )
- No-attack constraint for  $V_1$  and  $V_2$ :  
 $\{\langle 1, 3 \rangle, \langle 1, 4 \rangle, \langle 1, 5 \rangle, \dots, \langle 2, 4 \rangle, \langle 2, 5 \rangle, \dots\}$

Implicitly: use a function to test for constraint satisfaction

# Applying Standard Search...



## Example: map colouring

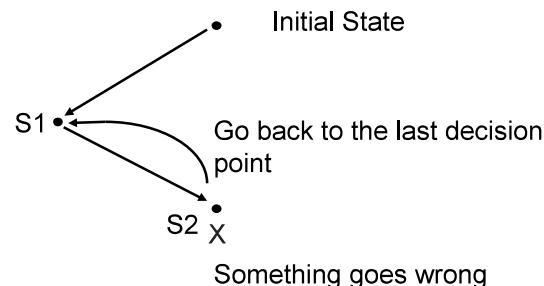


- Number of variables:  $n$
- Max. depth of space:  $n$
- Depth of solution state:  $n$  (all variables assigned)
- Search algorithm: depth-first search

# Backtracking Search



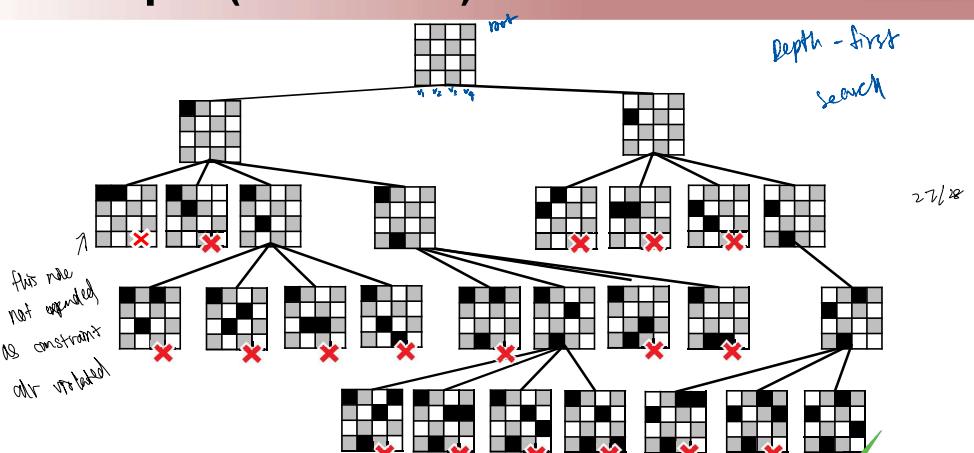
**Backtracking search:** Do not waste time searching when constraints have already been violated



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

if some constraints already violated,  
don't continue to expand ; not the  
solution we need ; print-if backtrack

# Example (4-Queens)





## Heuristics for CSPs

Plain backtracking is an uninformed algorithm!!

More intelligent search that takes into consideration

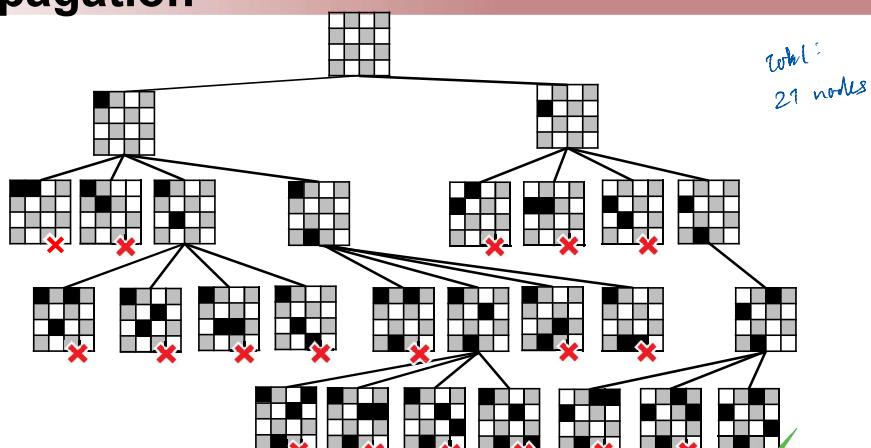
- Which variable to assign next
- What order of the values to try for each variable
- Implications of current variable assignments for the other unassigned variables
  - forward checking and **constraint propagation**

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

Question to think:

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

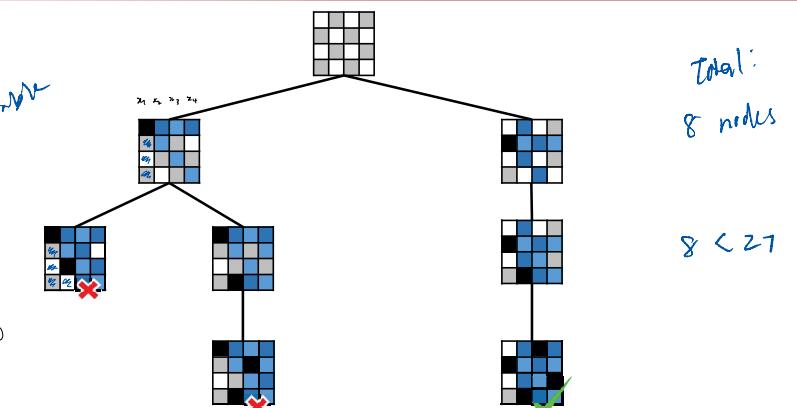
## Example (4-Queens) without Constraint Propagation



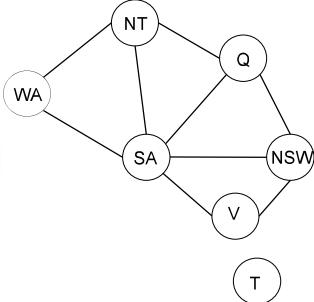
## Search Tree of 4-Queens with Constraint Propagation



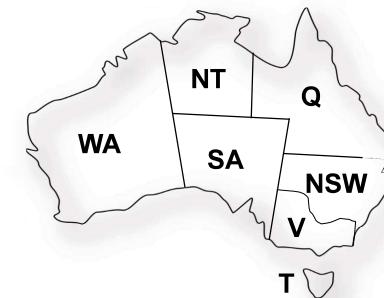
reduced  
no. of children  
for each variable  
if goal state  
→ no more branching



## Example (Map Colouring)



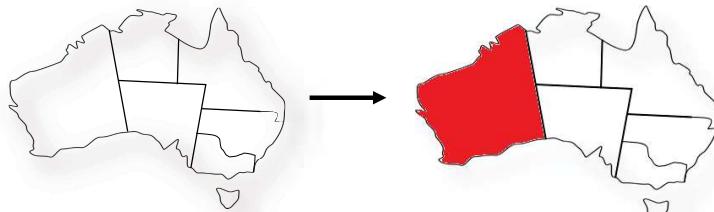
## Example (Map Colouring)...



WA	NT	Q	NSW	V	SA	T
■ Red	■ Green	■ Blue	■ Red	■ Green	■ Blue	■ Red



## Example (Map Colouring)...



WA	NT	Q	NSW	V	SA	T
■ Red	■ Green	■ Blue	■ Red	■ Green	■ Blue	■ Red
■ Red	■ Green	■ Blue	■ Red	■ Green	■ Blue	■ Red



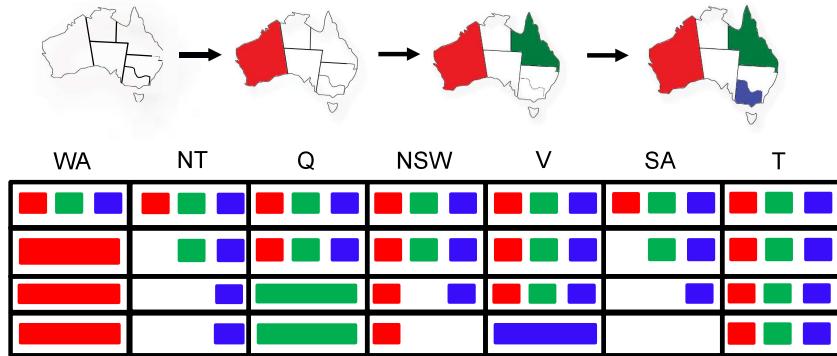
## Example (Map Colouring)...



WA	NT	Q	NSW	V	SA	T
■ Red	■ Green	■ Blue	■ Red	■ Green	■ Blue	■ Red
■ Red	■ Green	■ Blue	■ Red	■ Green	■ Blue	■ Red
■ Red	■ Green	■ Blue	■ Red	■ Green	■ Blue	■ Red



## Example (Map Colouring)...



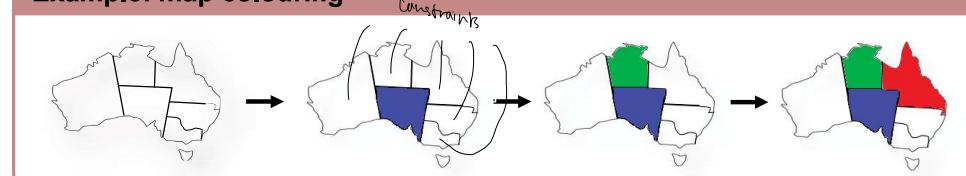
NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE



## Most Constrained Variable

Or minimum remaining values (MRV) heuristic

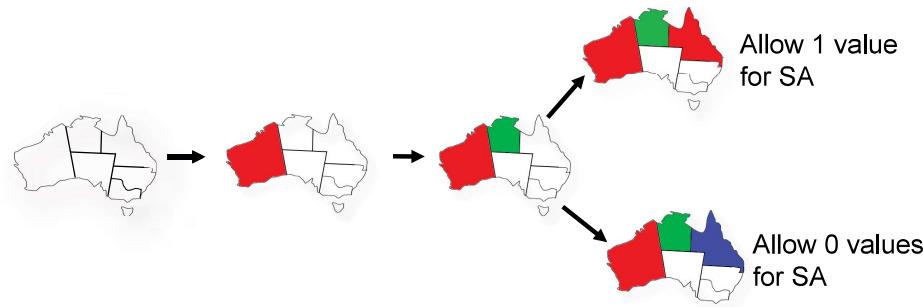
### Example: map colouring



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

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

## Least Constraining Value



Choose the value that leaves maximum flexibility for subsequent variable assignments

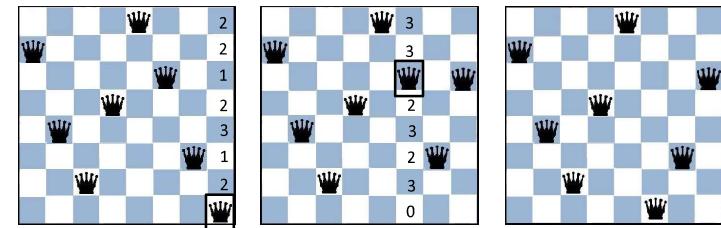
NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE



## 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

*Local search*



NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE





## 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

Question to think:

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

## 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

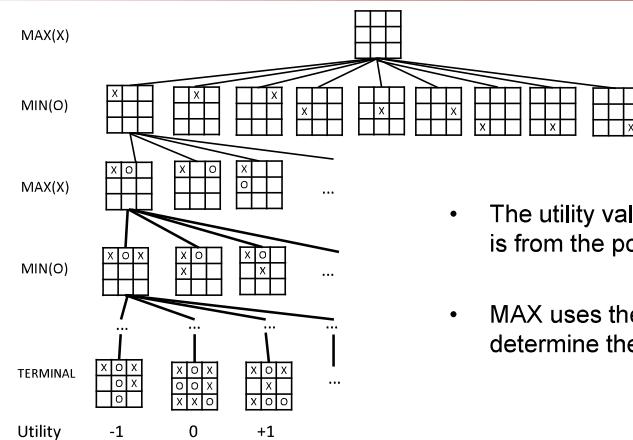


- Initial state: initial board configuration and indication of who makes the first move
- Operators: legal moves
- Terminal test: determines when the game is over
  - states where the game has ended: **terminal states**
- Utility function (payoff function): returns a numeric score to **quantify** the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

### Example: Chess

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

## Game Tree for Tic-Tac-Toe

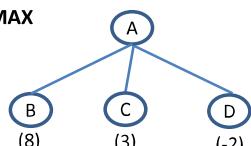


- The utility value of the terminal state is from the point of view of MAX
- MAX uses the search tree to determine the best move

## What Search Strategy?

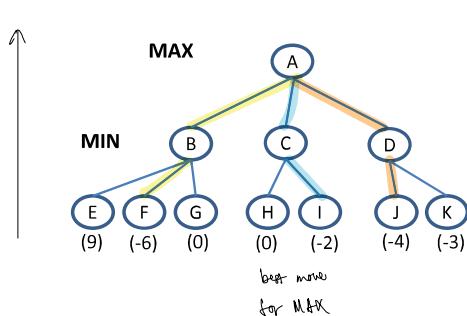


### One-play



min-max search

### Two-play



best move  
for MIN

## What Search Strategy?...



MAX

MIN

Min max Search



# 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

- Generate the entire game tree down to terminal states
- Calculate utility
  - Assess the utility of each terminal state
  - Determine the best utility of the parents of the terminal state
  - Repeat the process for their parents until the root is reached
- 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

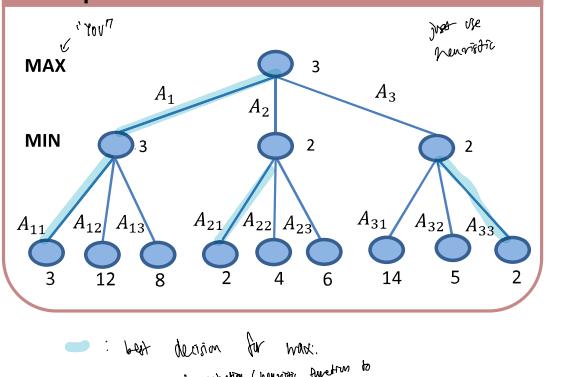
- generate the complete search tree

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

may not work for games like Go,  
too many depth! end up building  
depth-limited search | but still makes it terminate nodes → make use of evaluation / heuristic function to  
estimate the value of last node in depth-limited

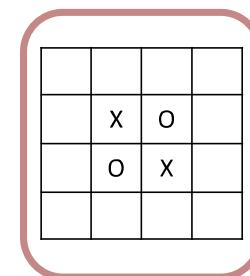
## Example



# Othello 4

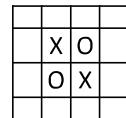


- A player can place a new piece in a position if there exists at least one straight (horizontal, vertical, or diagonal) occupied line between the new piece and another piece of the same kind, with one or more contiguous pieces from the opponent player between them
- After placing the new piece, the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

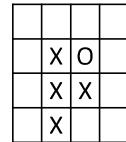


## 'X' plays first

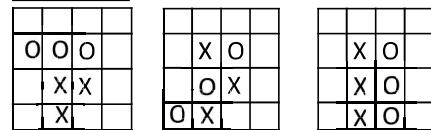
X considers the game now



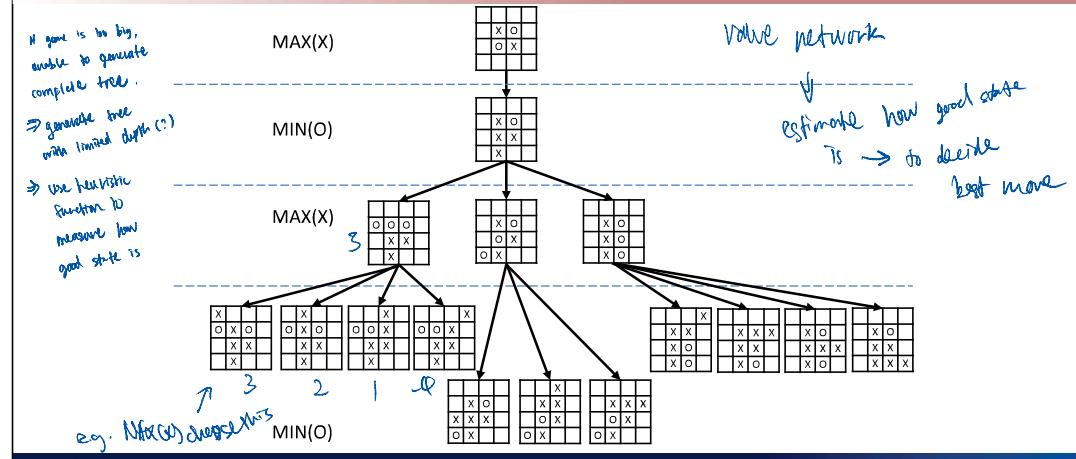
O considers the game now



X considers the game now



## Game Tree Othello 4



## Imperfect Decisions



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

Time/space requirements  $\rightarrow$  complete game tree search is intractable  $\rightarrow$  **impractical** to make perfect decisions

**Modifications** to minimax algorithm

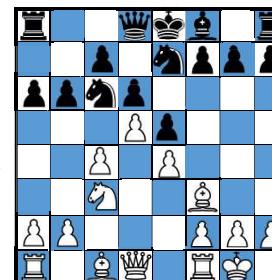
1. replace utility function by an **estimated** desirability of the position
  - **Evaluation function**
2. **partial tree search**
  - E.g., depth limit
  - Replace terminal test by a **cut-off** test

## Evaluation Functions

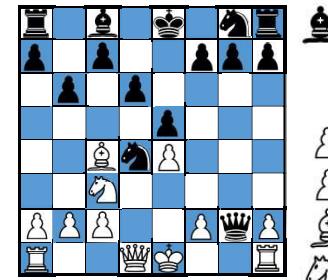


Returns an **estimate** of the expected utility of the game from a given position

make assumption of opponent  
→ assume opponents use best strategy as well  
use minimax search to find best plan



Black: to move  
White: slightly better



White: to move  
Black: winning