NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

AI6101 Introduction to AI and AI Ethics

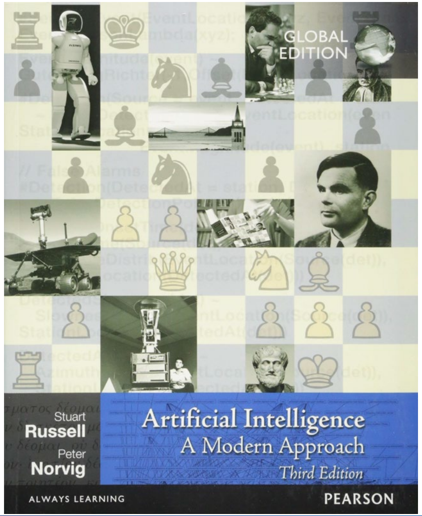
Intelligent Agents and Search

Assoc Prof Bo AN

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

Lesson Outline

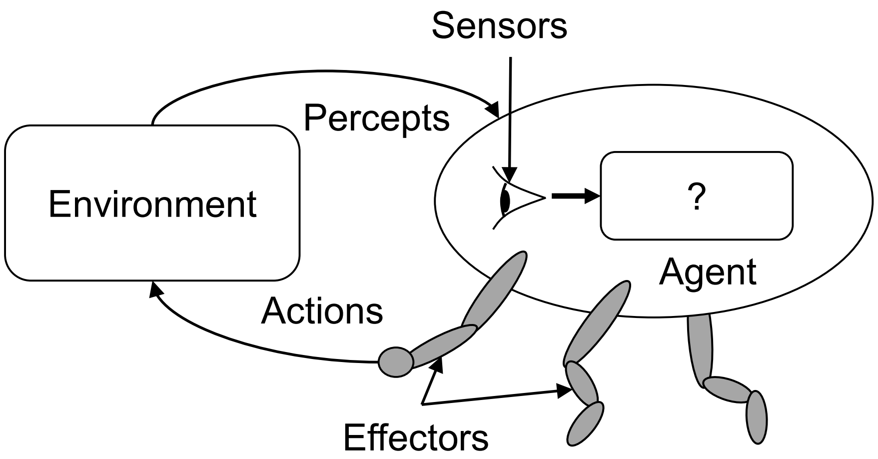
- How can one describe the task/problem for the agent?
- What are the properties of the task environment for the agent?
- Problem formulation
- Uninformed search strategies
- Informed search strategies: greedy search, A * search
- Constraint Satisfaction
- Game Playing



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, …

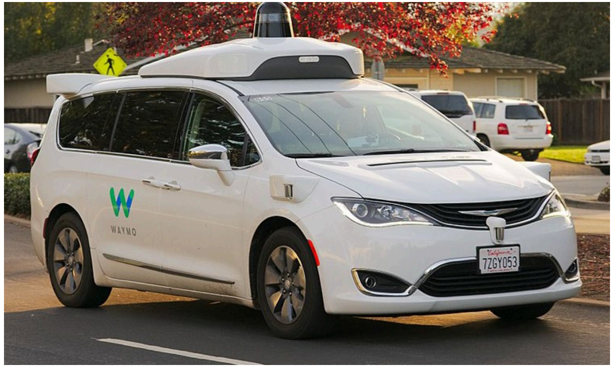


Image source: https://en.wikipedia.org/wiki/Waymo#/media/File:Waymo_Chrysler_Pacifica_in_Los_Altos,_2017.jpg

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

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)

Well-Defined Formulation

Definition of a problem	The information used by an agent to decide what to do

Specification	• Initial state
	• Action set, i.e. available actions (successor functions)
	• State space, i.e. states reachable from the initial state
	    • Solution path: sequence of actions from one state to another
	• Goal test predicate
	    • Single state, enumerated list of states, abstract properties
	• Cost function
	    • 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

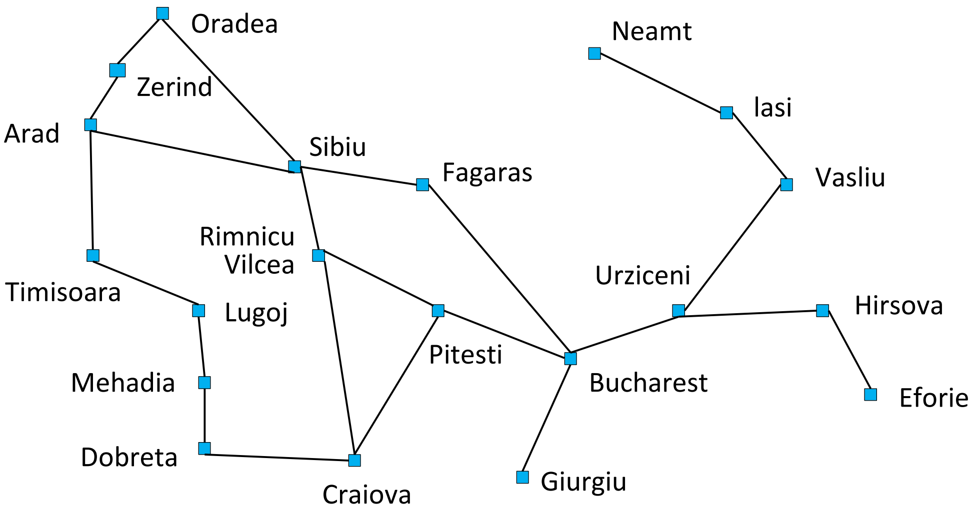
Search Cost

- 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

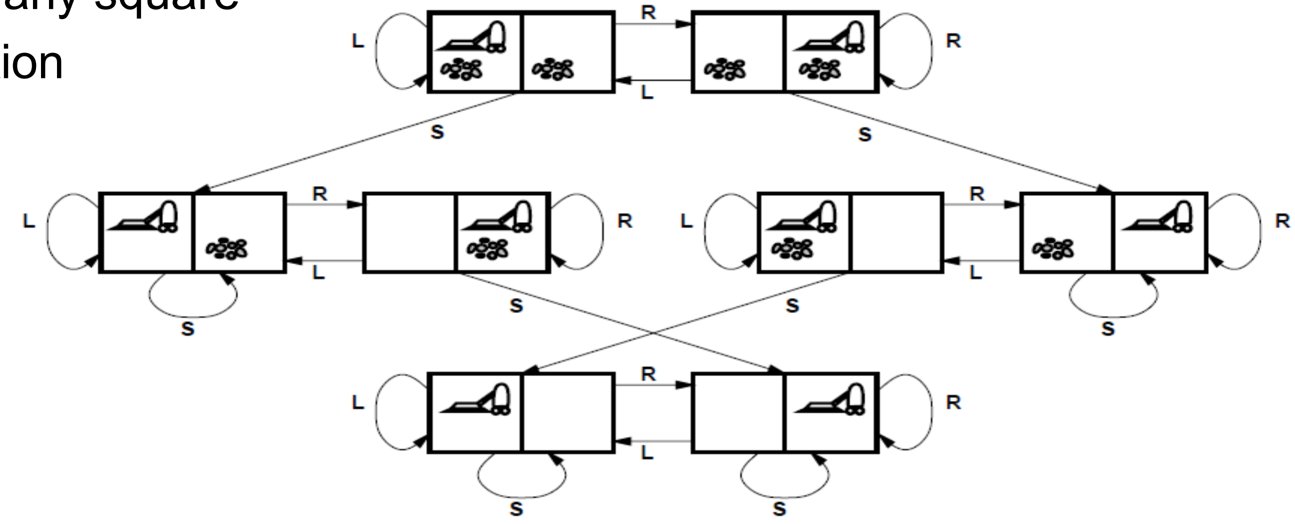
Single-State Problem Example



- 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

5	4	
6	1	8
7	3	2

Goal state

1	2	3
8		4
7	6	5

Real-World Problems

Route finding problems:

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

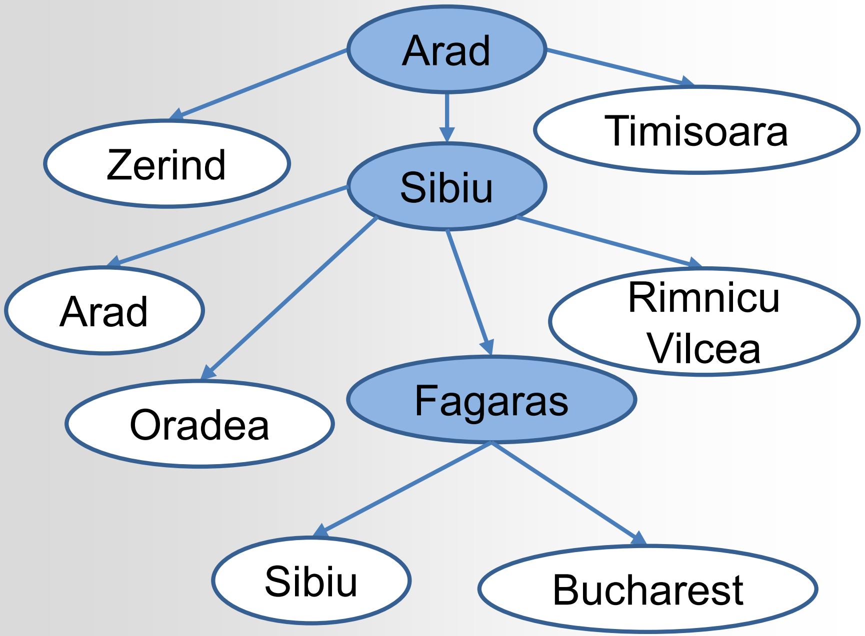
Touring problems:

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

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

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?

Uninformed vs Informed

Uninformed search strategies

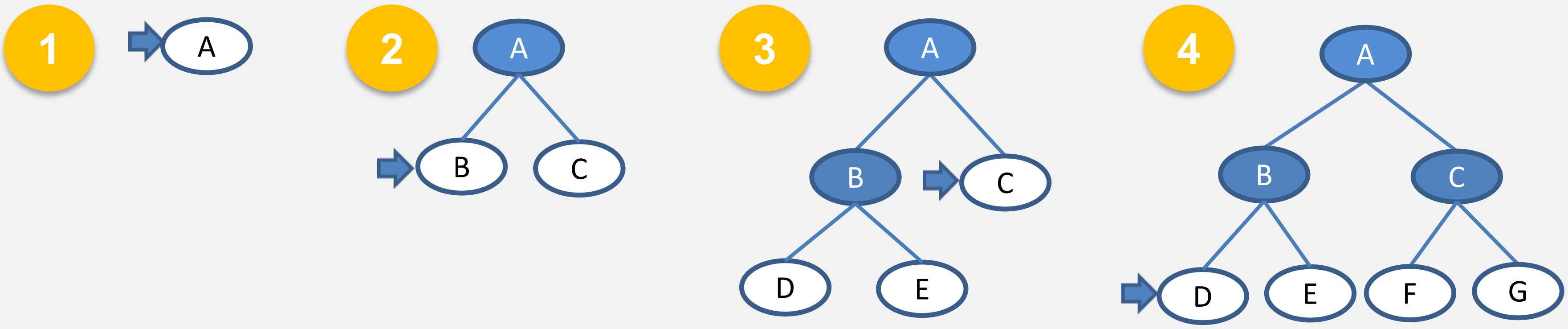
- Use only the information available in the problem definition
  1. Breadth-first search
  2. Uniform-cost search
  3. Depth-first search
  4. Depth-limited search
  5. Iterative deepening search

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

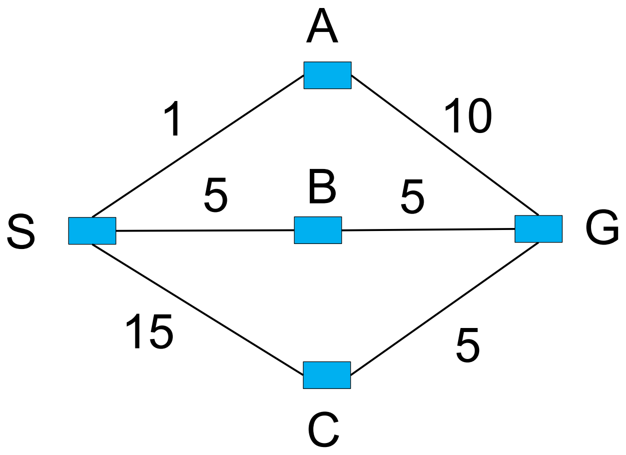
- Hypothetical state-space, where every node can be expanded into *b* new nodes, solution of path-length *d*
- Time: $1 + b + b^2 + b^3 + \ldots + b^d = O(b^d)$
- Space: (keeps every node in memory) $O(b^d)$are equal

Depth	Nodes	Time		Memory	
0	1	1	millisecond	100	bytes
2	111	0.1	seconds	1	kilobytes
4	11111	11	seconds	11	kilobytes
6	10^6	18	minutes	111	megabyt e
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

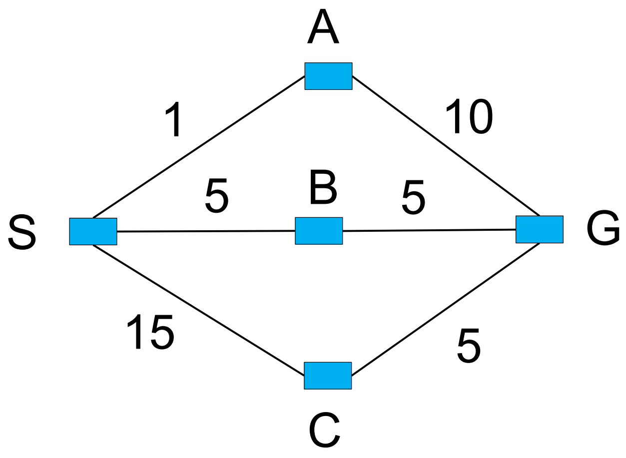
Uniform-Cost Search

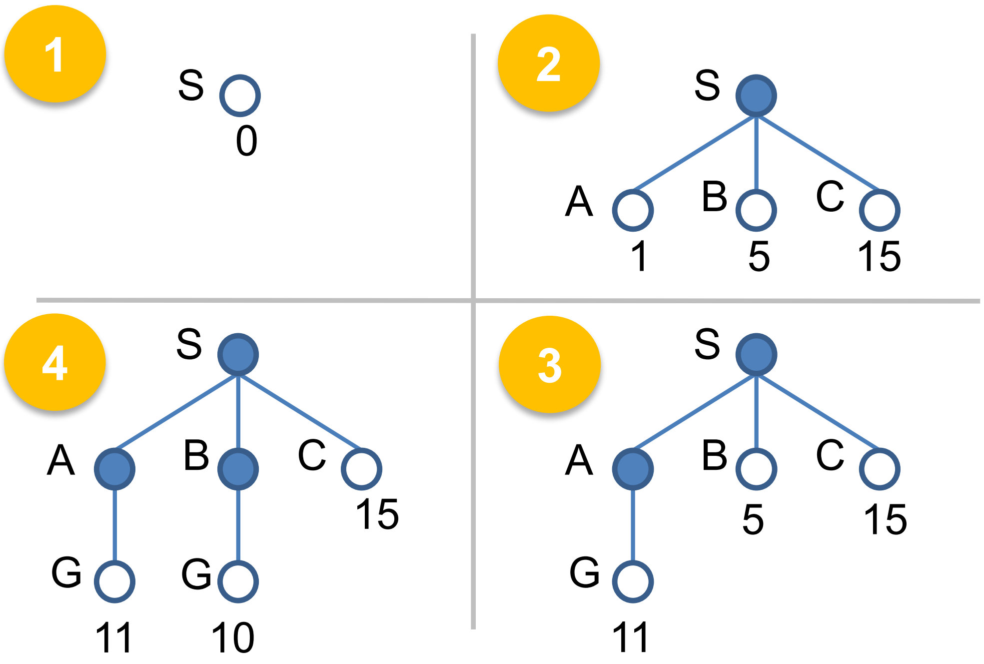
To consider edge costs, expand unexpanded node with the least path cost *g*

- 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





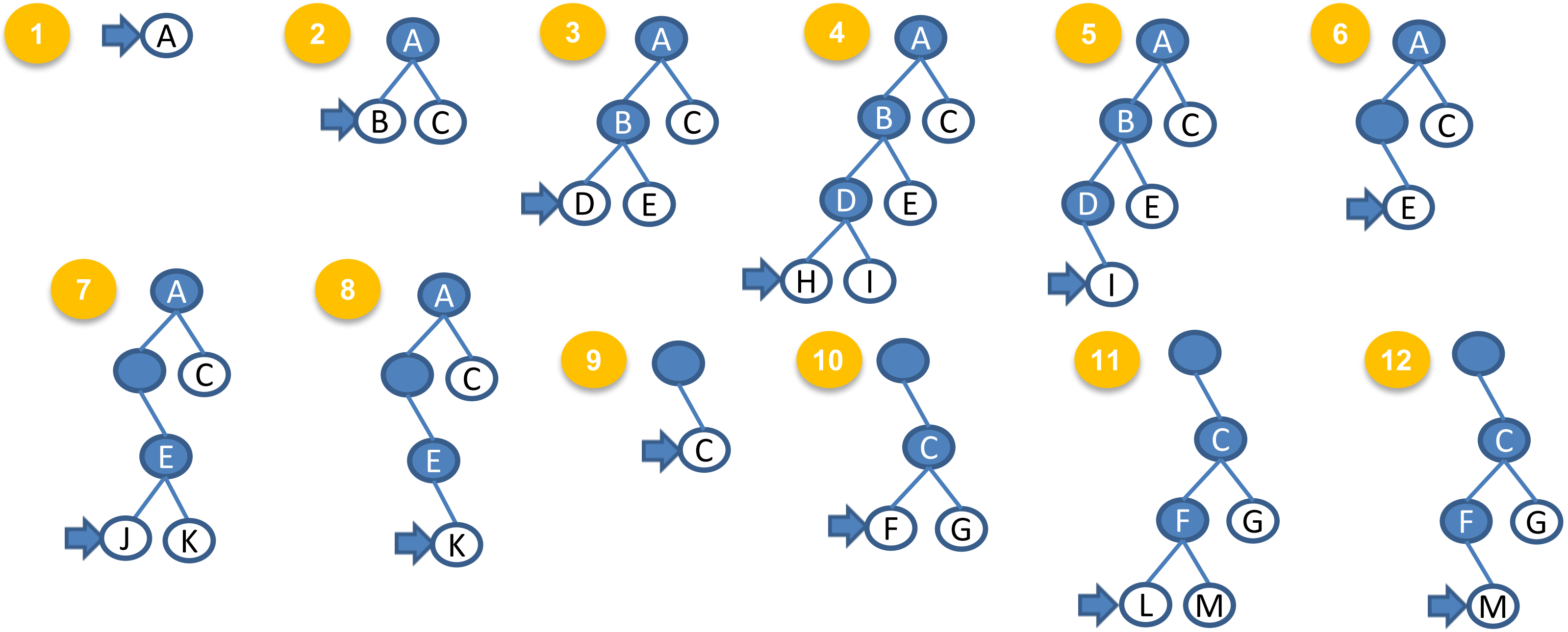
Here we do not expand notes that have been expanded.

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 • finite-depth spaces with loops: No • with repeated-state checking: Yes • finite-depth spaces without loops: Yes
Time	$O(b^m)$ 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 l of a path

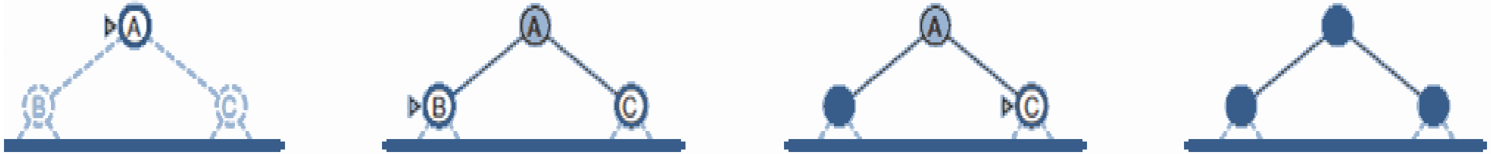
Complete	Yes, if $l \geq d$
Time	$O(b^l)$
Space	$O(bl)$
Optimal	No

Iterative Deepening Search

Iteratively estimate the max depth *l* of DLS one-by-one

Limit = 0

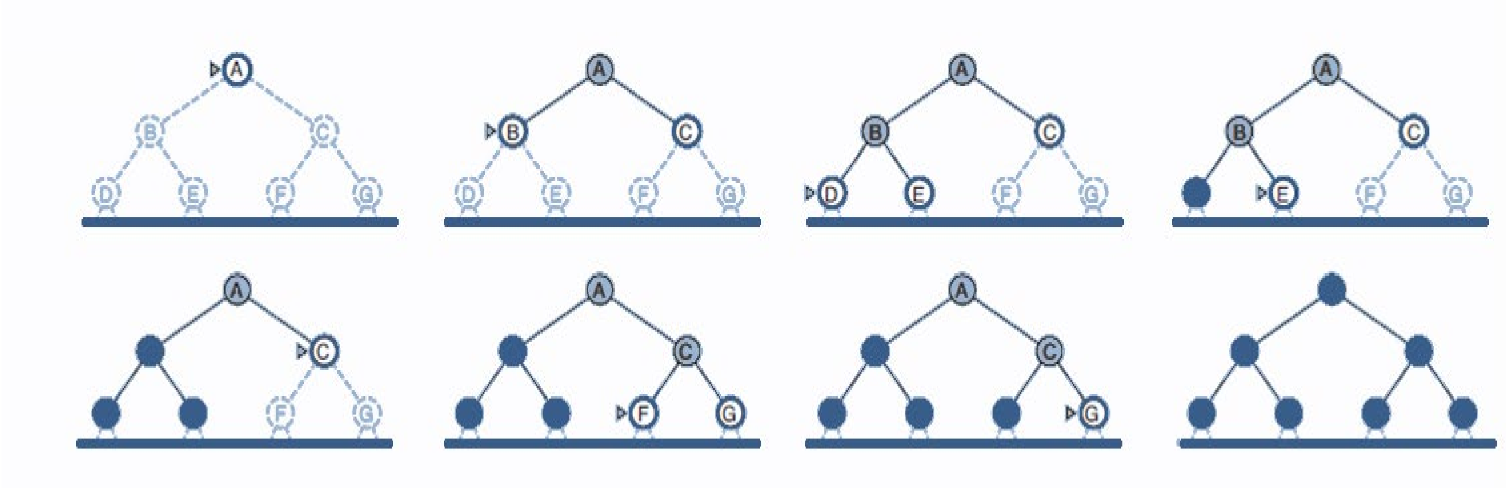
Limit = 1



Iterative Deepening Search

Iteratively estimate the max depth *l* of DLS one-by-one

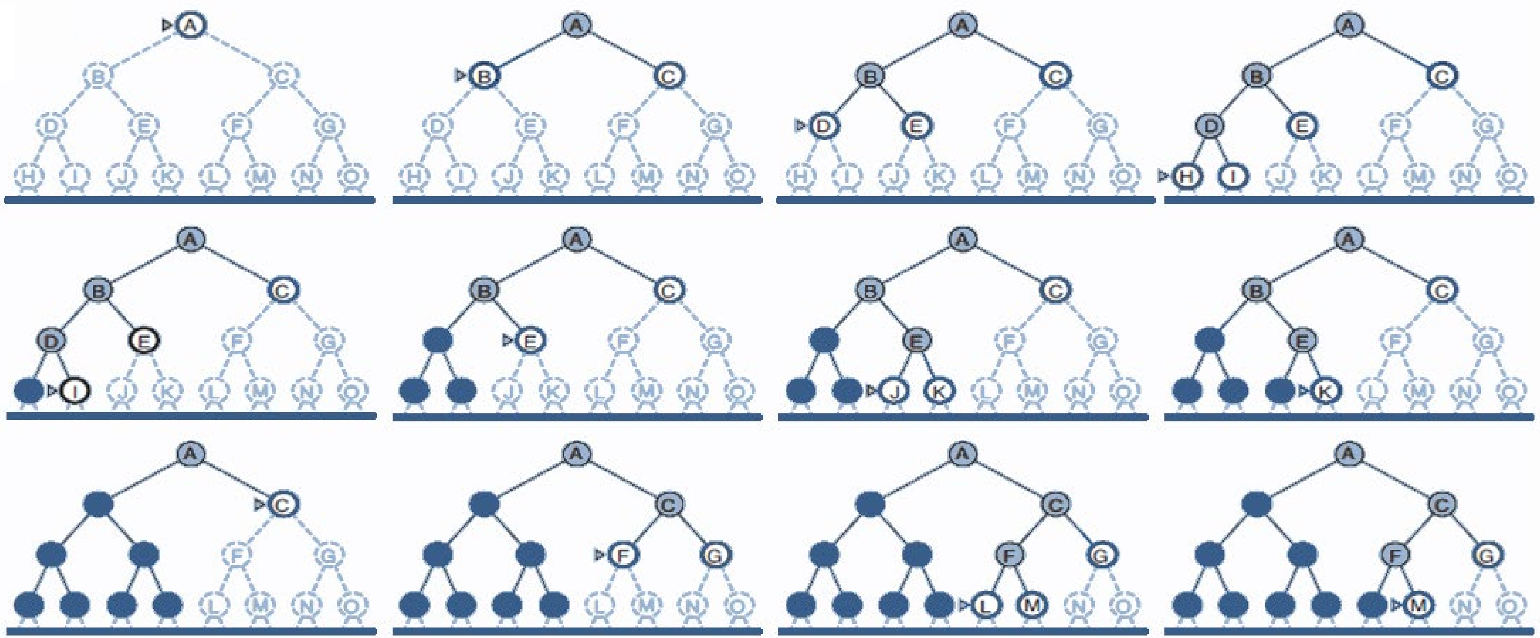
Limit = 2



Iterative Deepening Search

Iteratively estimate the max depth *l* 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

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

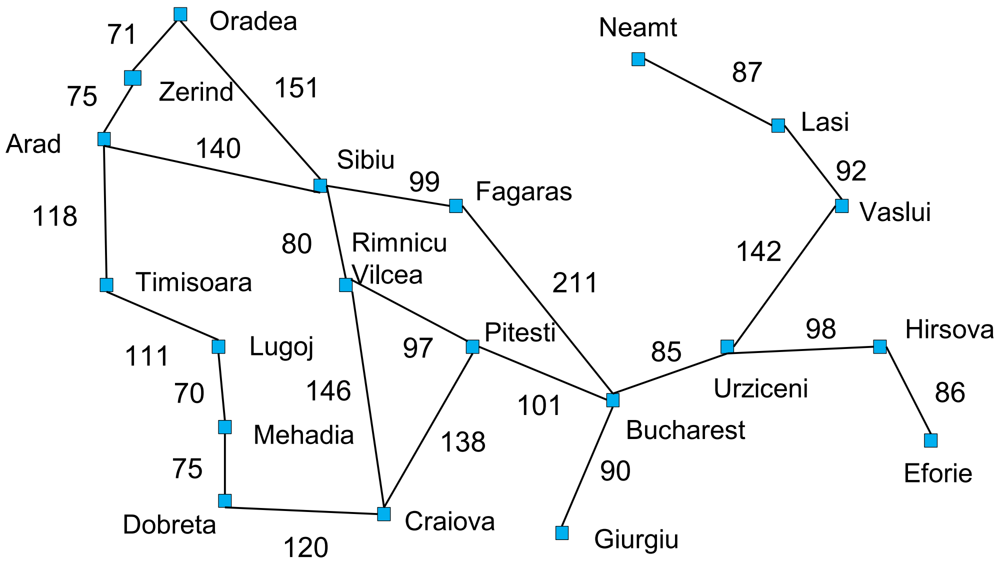
- Evaluation function $h(n)$:estimate of cost from n to $goal$
- Function Greedy-Search(problem) returns solution
  - Return Best-First-Search(problem, h) // $h(goal) = 0$

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



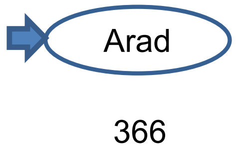
- Useful but potentially fallible (heuristic)
- Heuristic functions are problem-specific

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

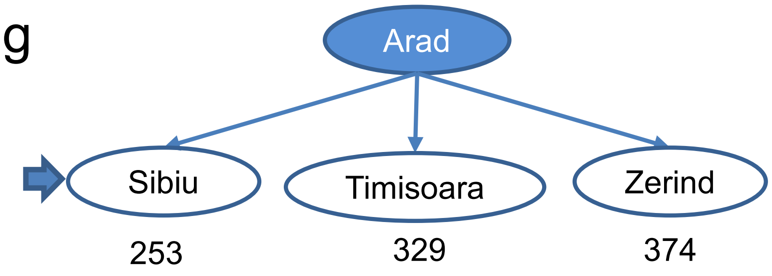


Straight-line distance to Bucharest

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

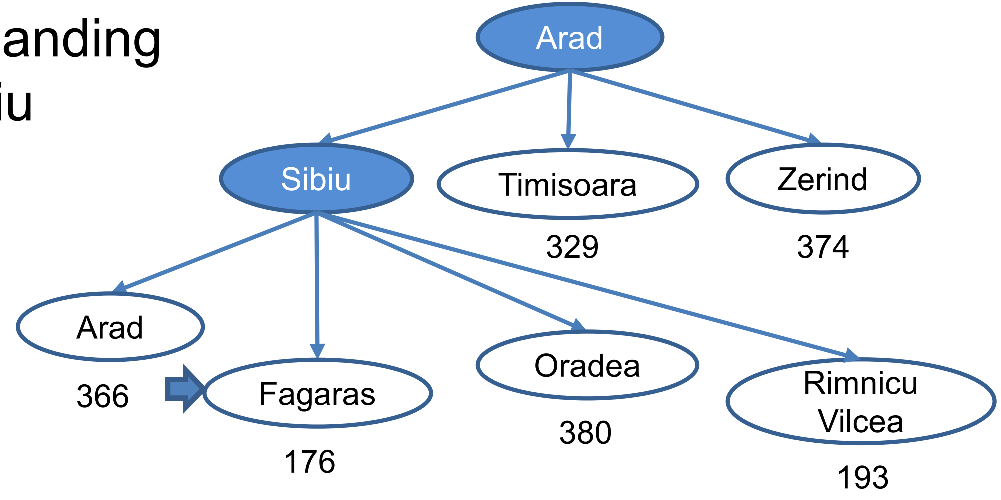


Straight-line distance to Bucharest

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

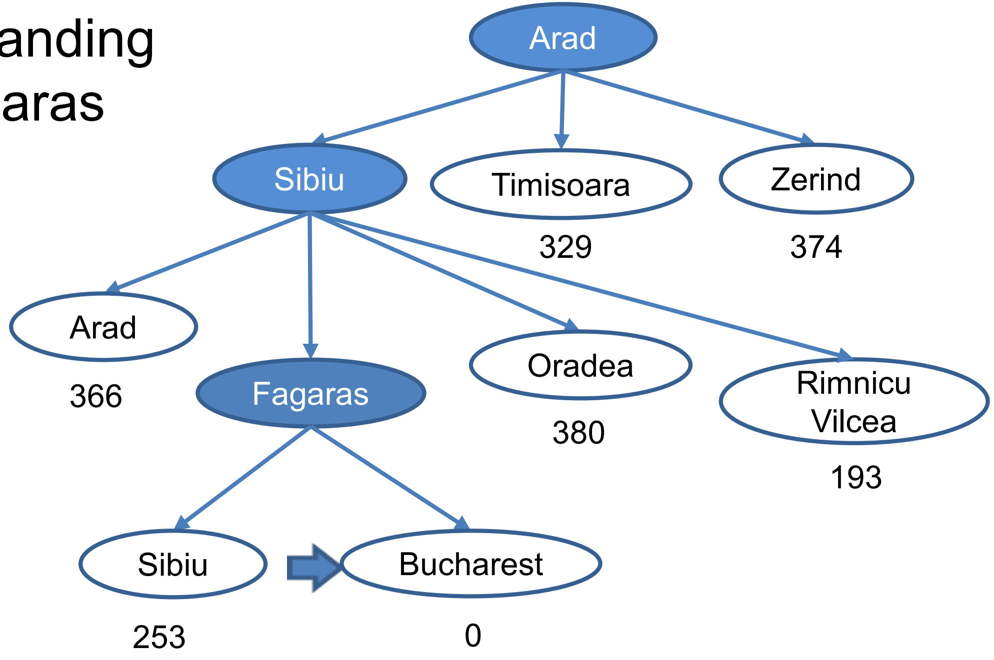


Straight-line distance to Bucharest

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



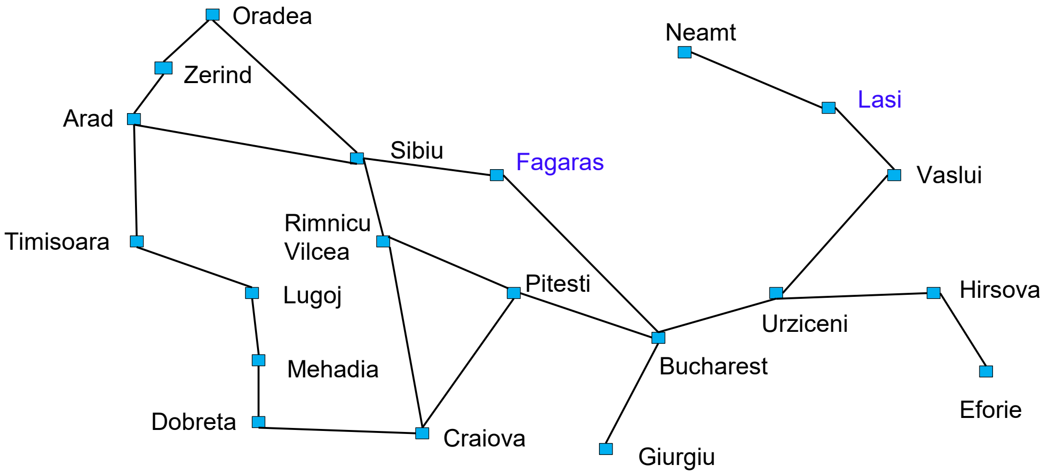
Straight-line distance to Bucharest

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

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

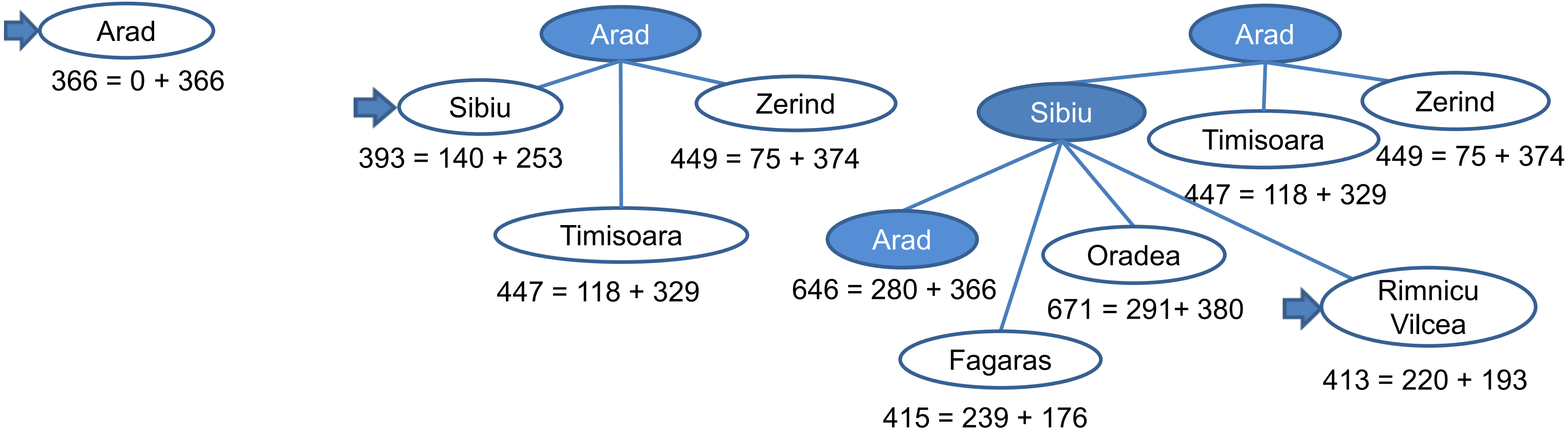
Evaluation function: $f(n) = g(n) + h(n)$

- *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 $g + h$

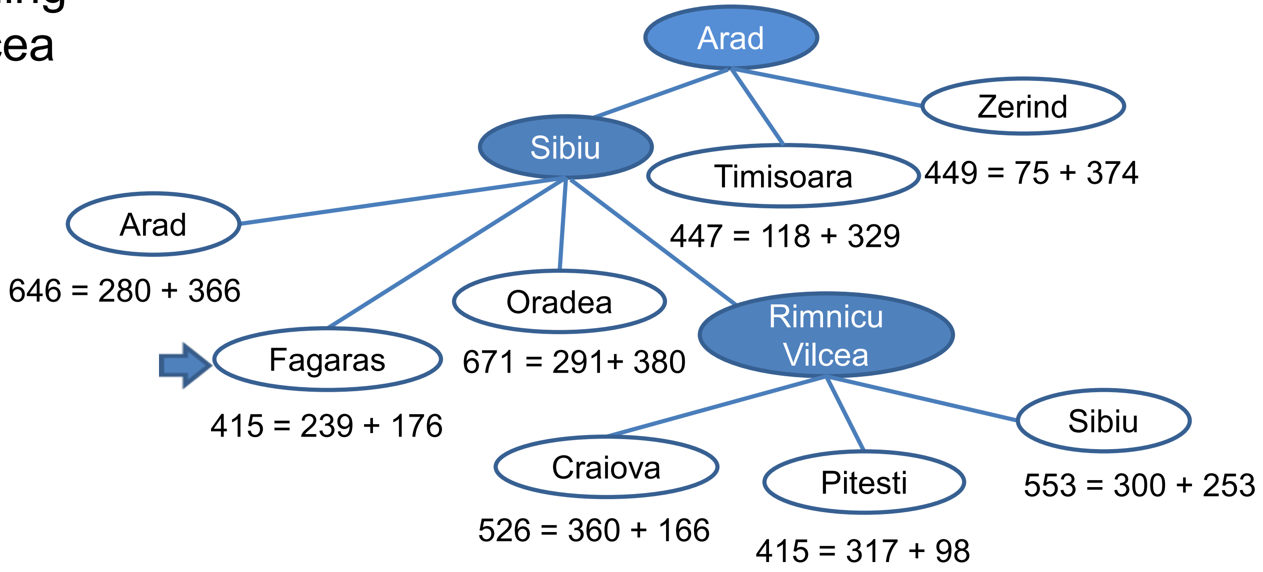
(a) The initial state (b) After expanding Arad (c) After expanding Sibiu



Example: Route-finding from Arad to Bucharest

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

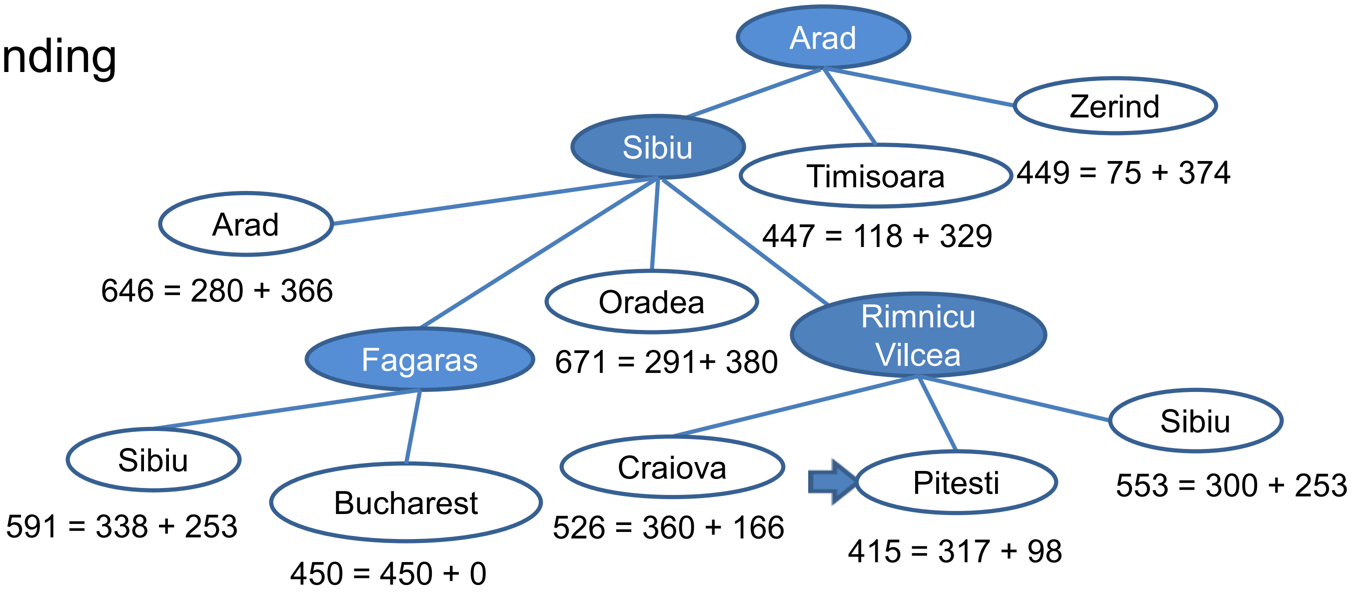
(d) After expanding
Rimnicu Vilcea



Example: Route-finding from Arad to Bucharest

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

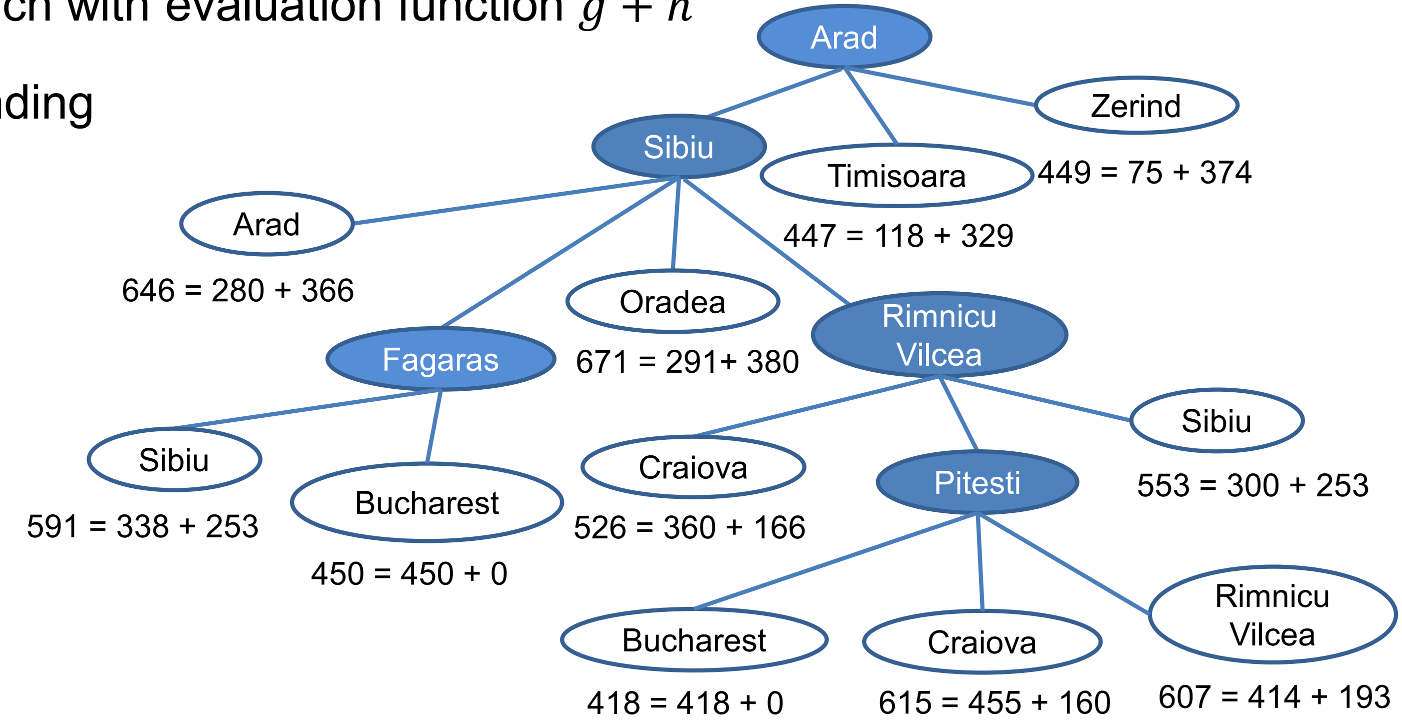
(e) After expanding
Fagaras



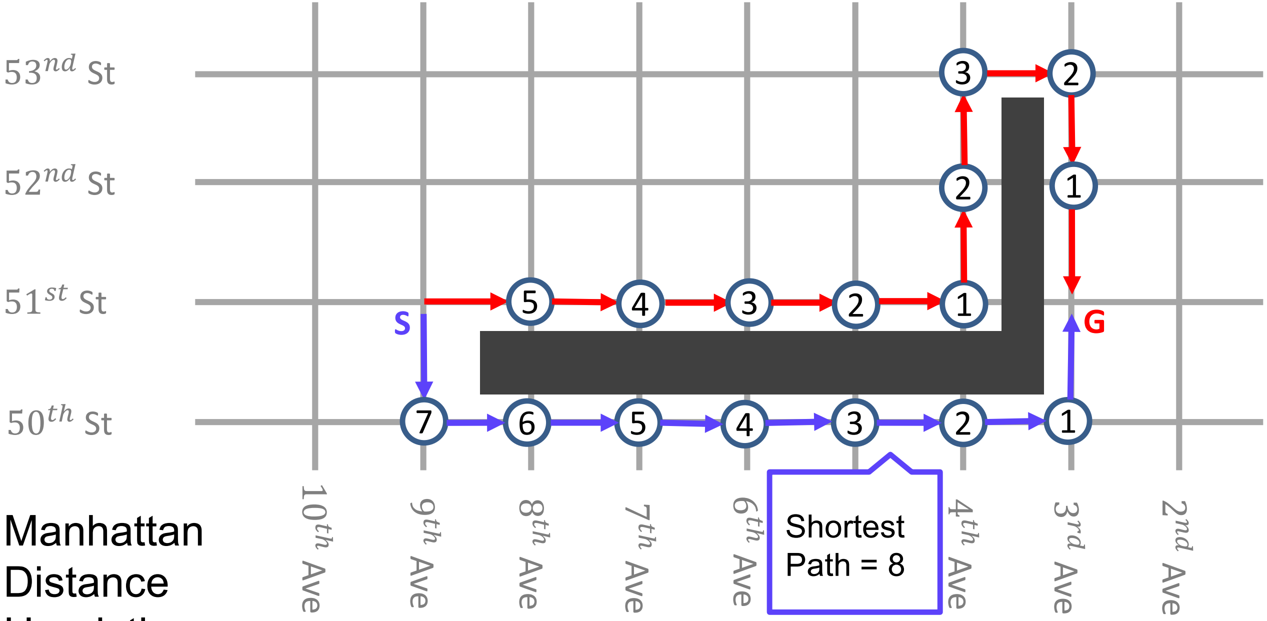
Example: Route-finding from Arad to Bucharest

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

(f) After expanding Pitesti

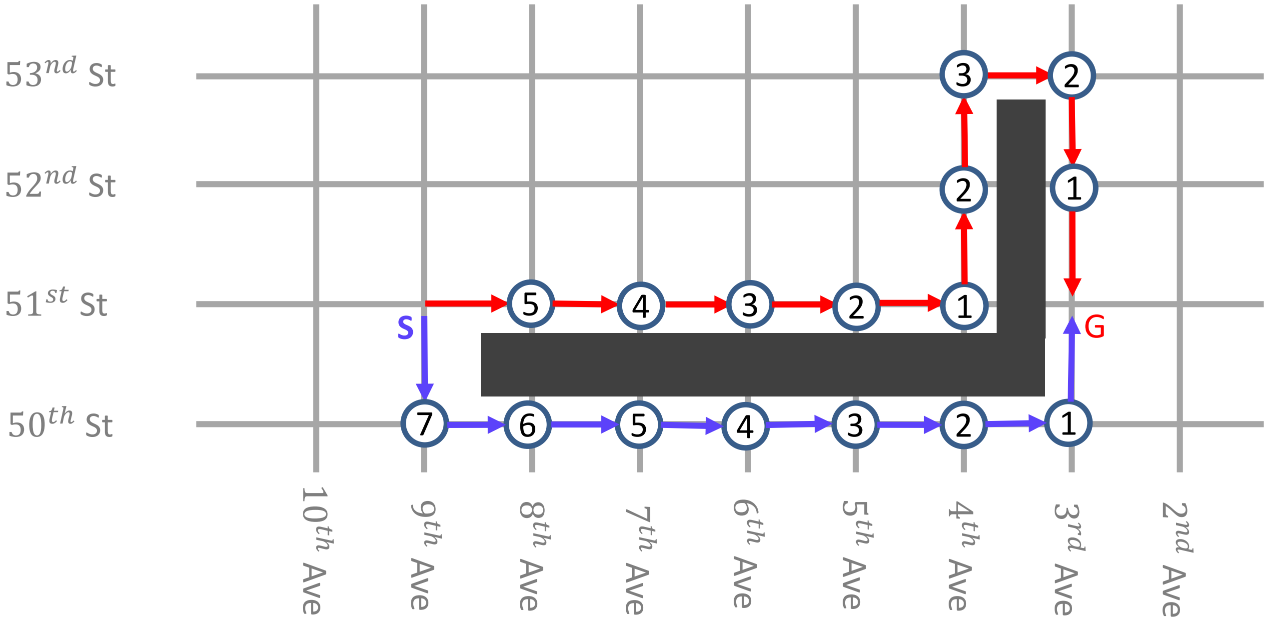


Example: Route-finding in Manhattan

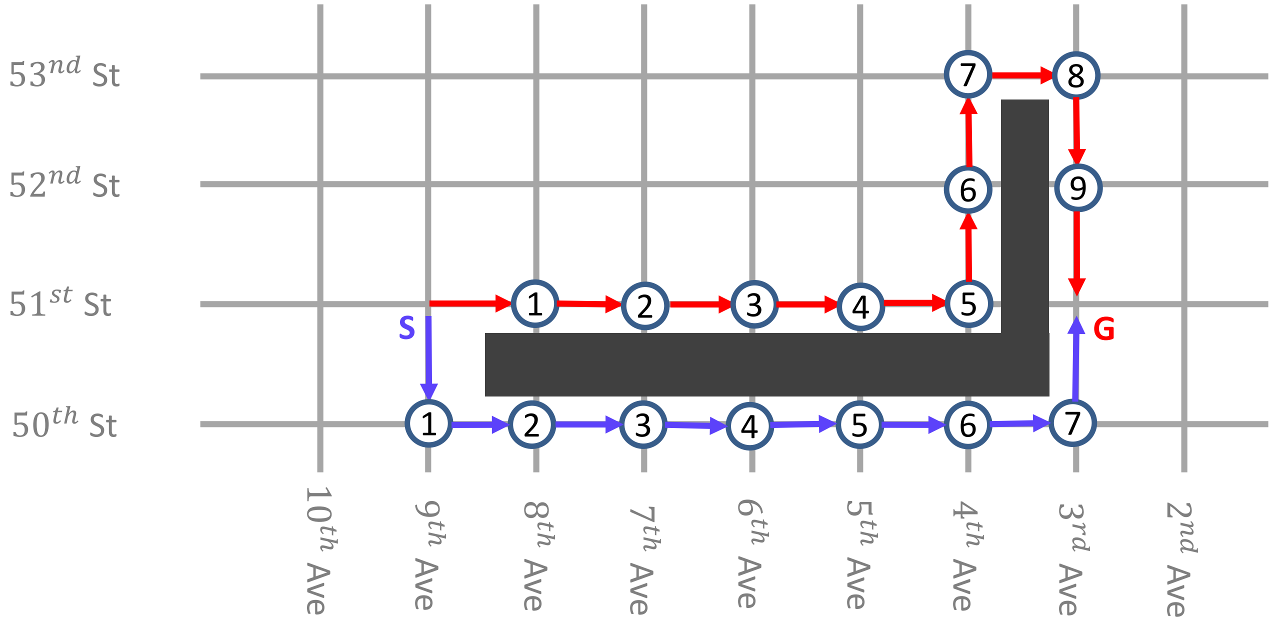


Manhattan Distance Heuristic

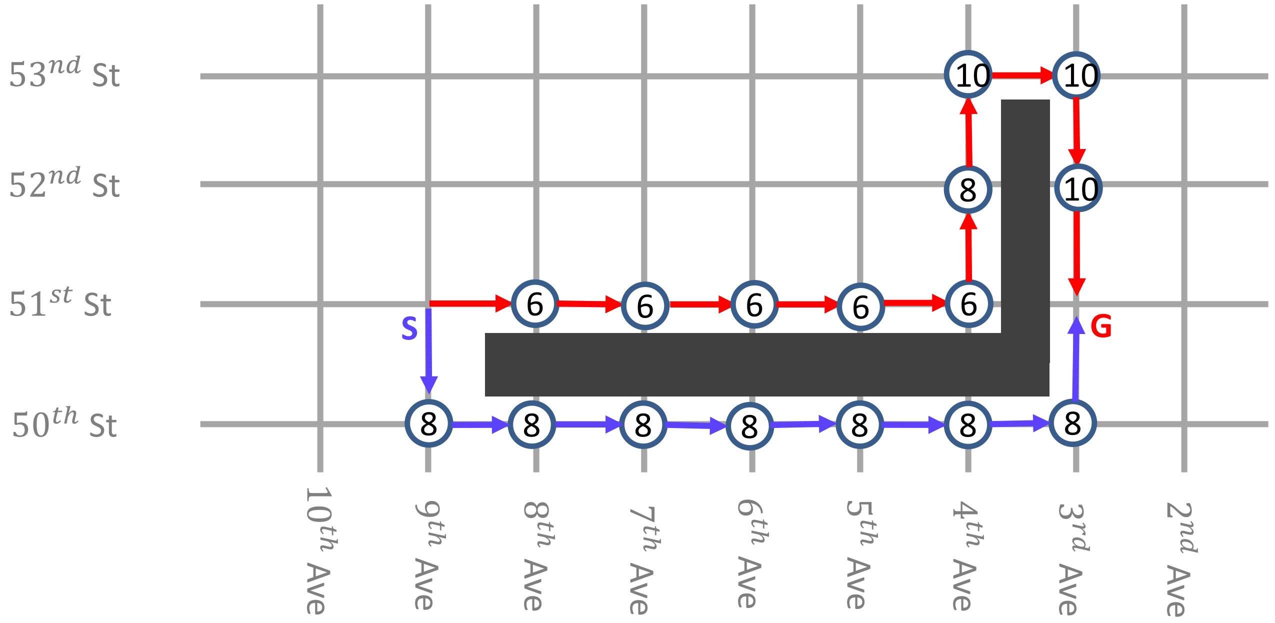
Example: Route-finding in Manhattan (Greedy)



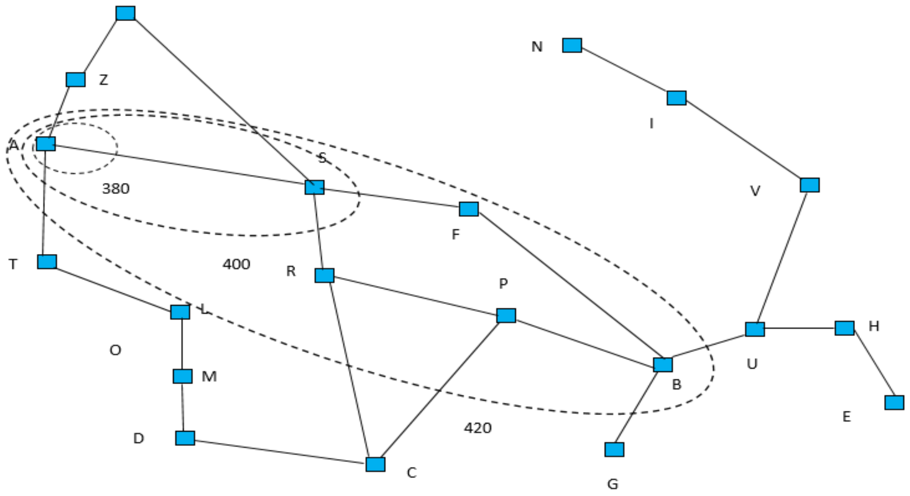
Example: Route-finding in Manhattan (UCS)



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

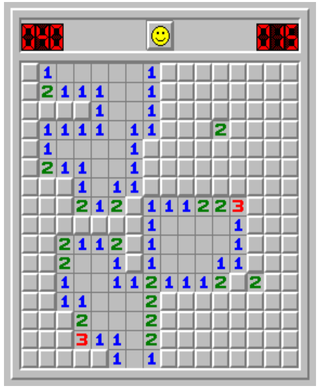
Constraint Satisfaction Problem (CSP)

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

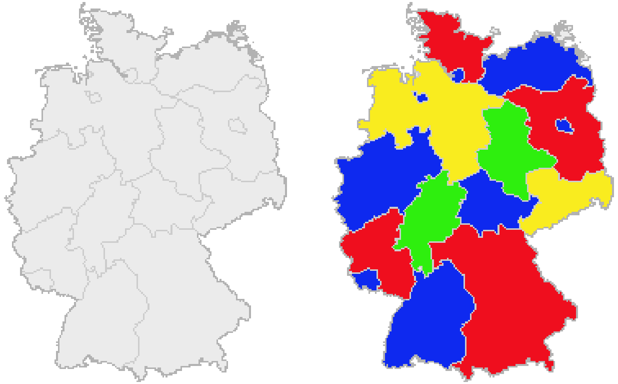
- a set of constraints specifying allowable combinations of values for subsets of variables

Example: 8-queens

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

Example: 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)

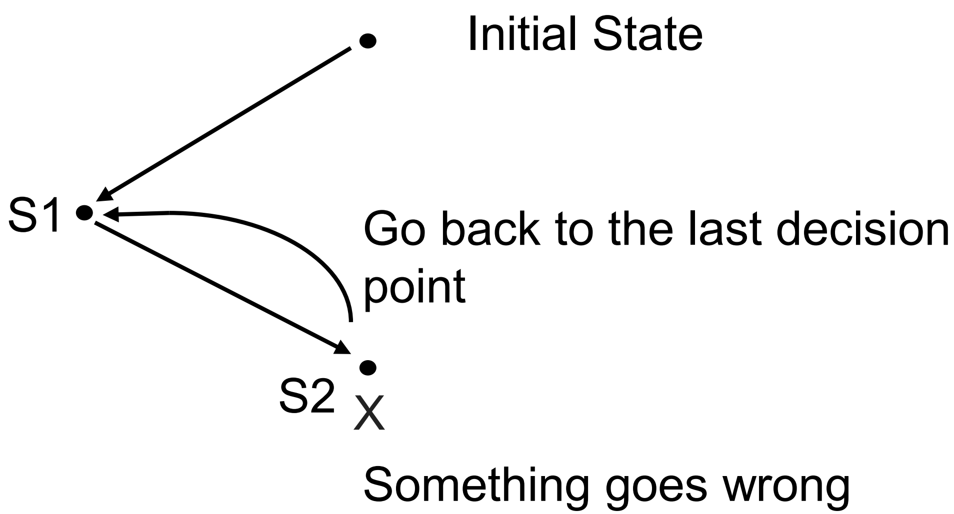
Example

- 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

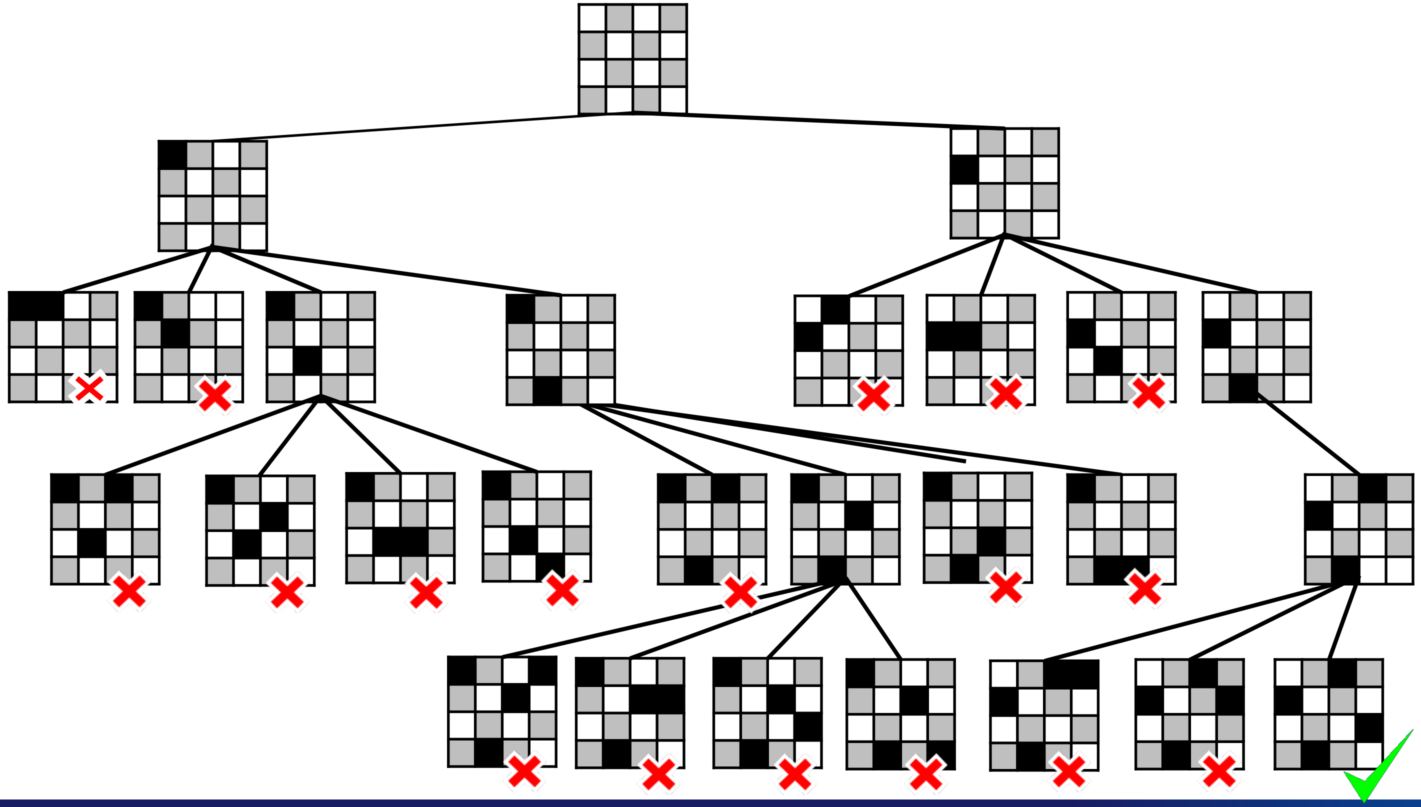
Backtracking Search

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)



Heuristics for CSPs

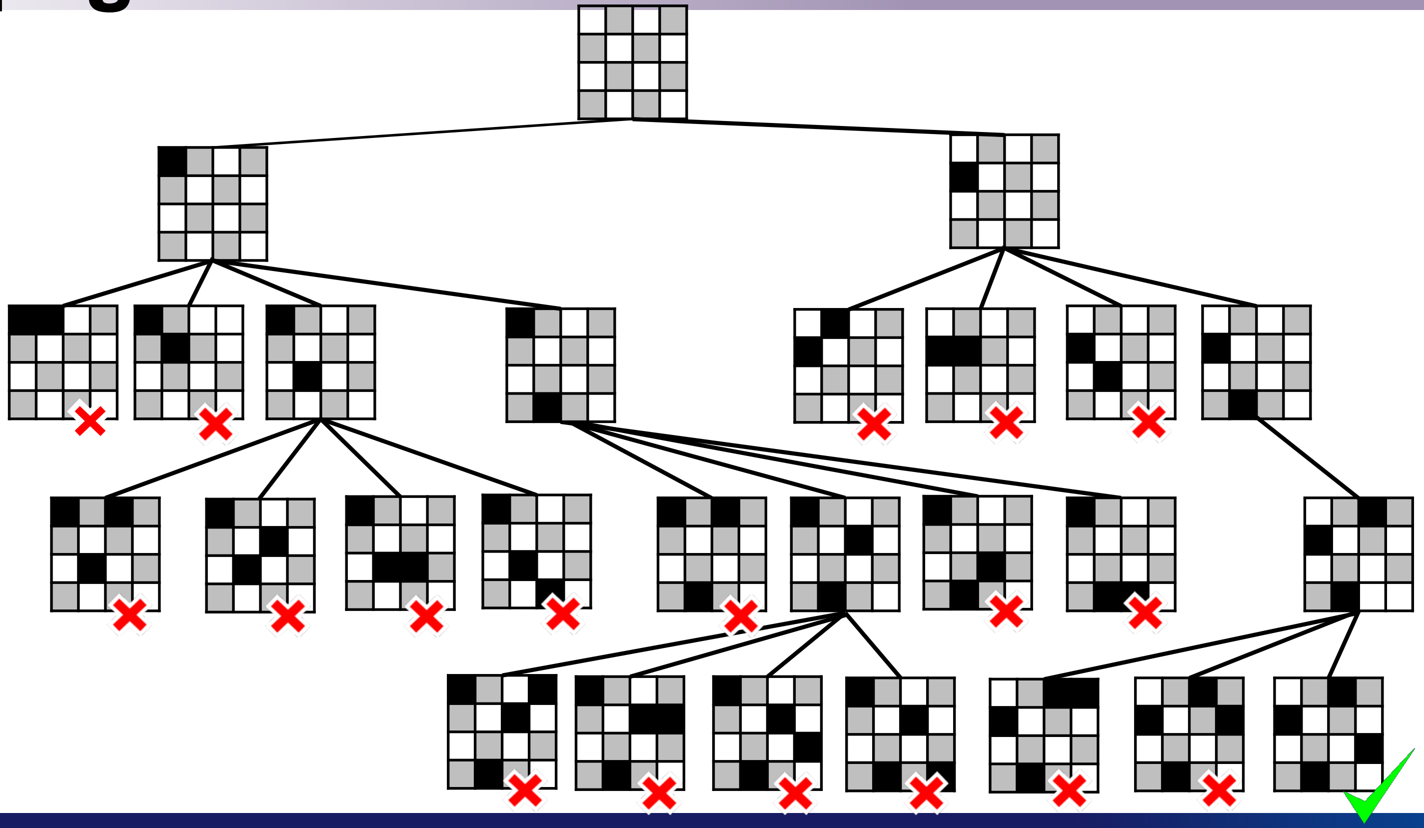
Plain backtracking is an uninformed algorithm!!

More intelligent search that takes into consideration

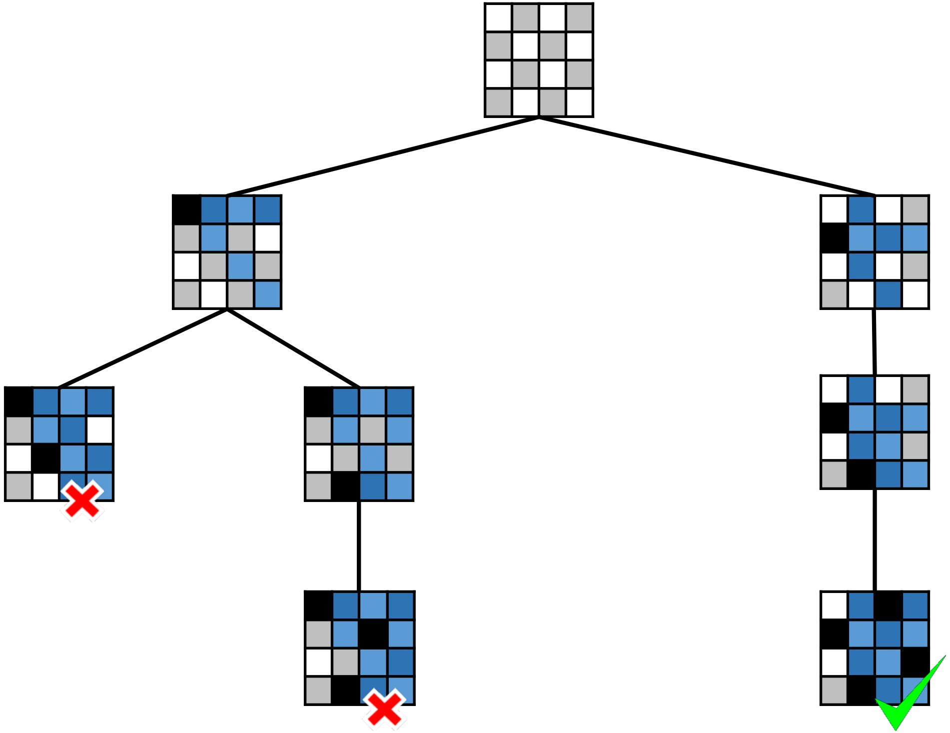
- Which variable to assign next
- What order of the values to try for each variable
- Implications of current variable assignments for the other unassigned variables
    - forward checking and **constraint propagation**

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

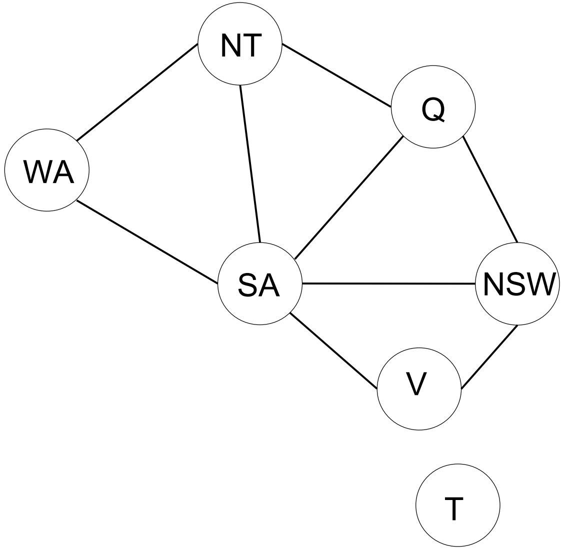
Example (4-Queens) without Constraint Propagation



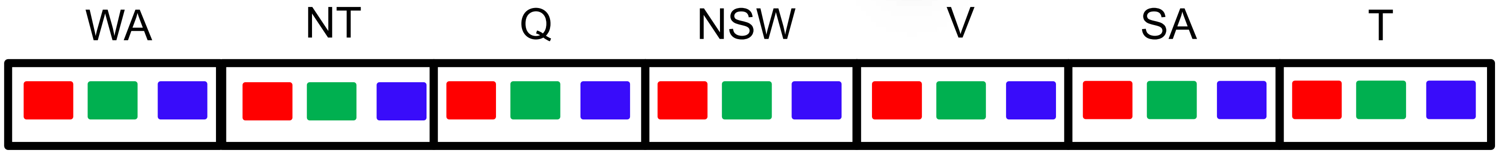
Search Tree of 4-Queens with Constraint Propagation



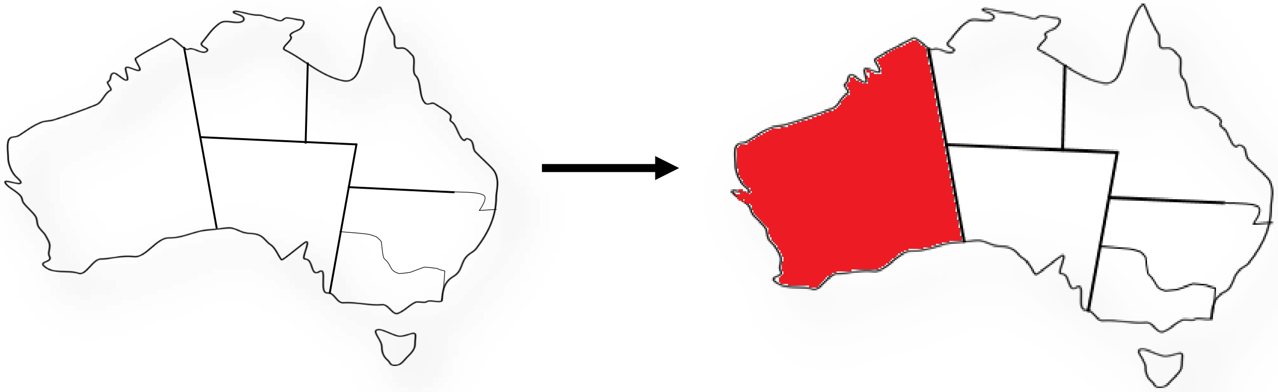
Example (Map Colouring)

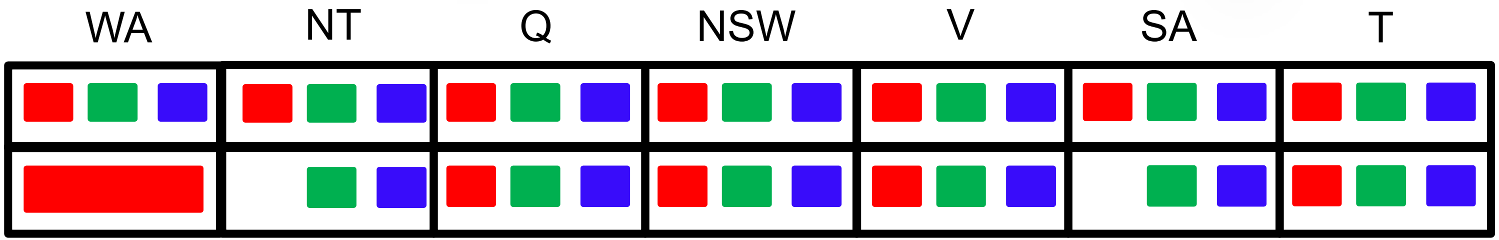


Example (Map Colouring)...

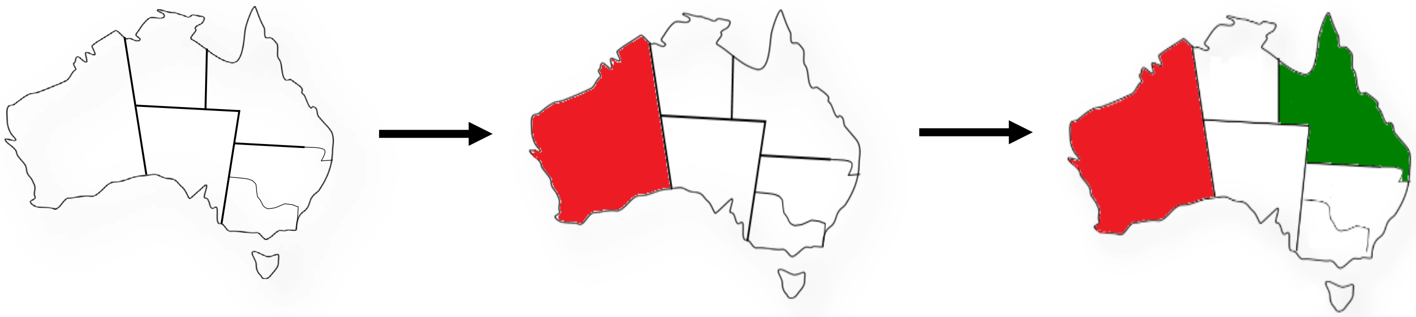


Example (Map Colouring)...

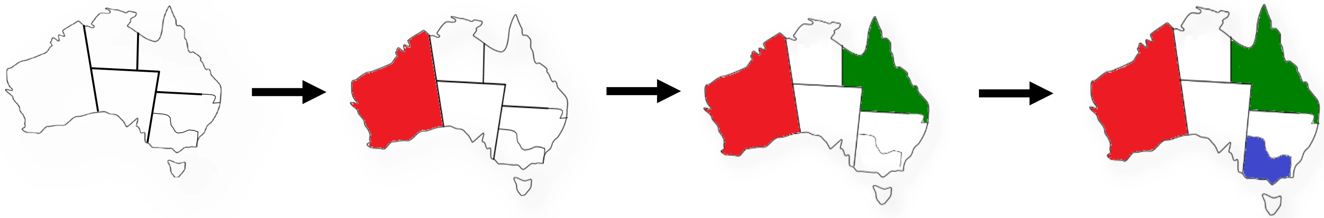




Example (Map Colouring)...



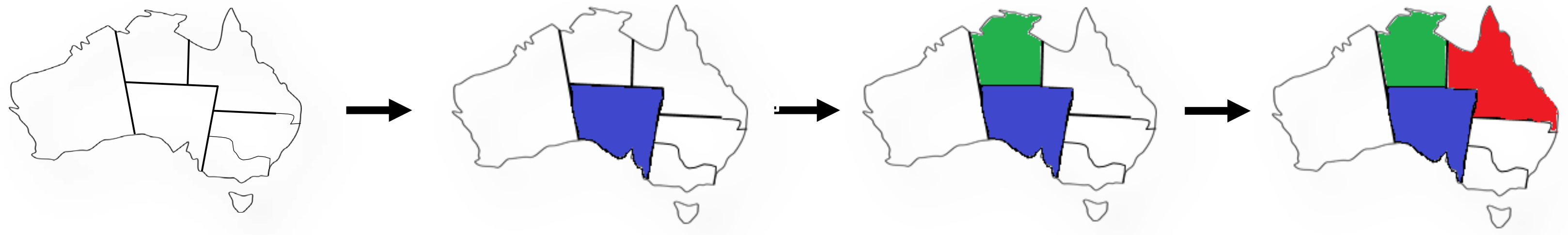
Example (Map Colouring)...



Most Constrained Variable

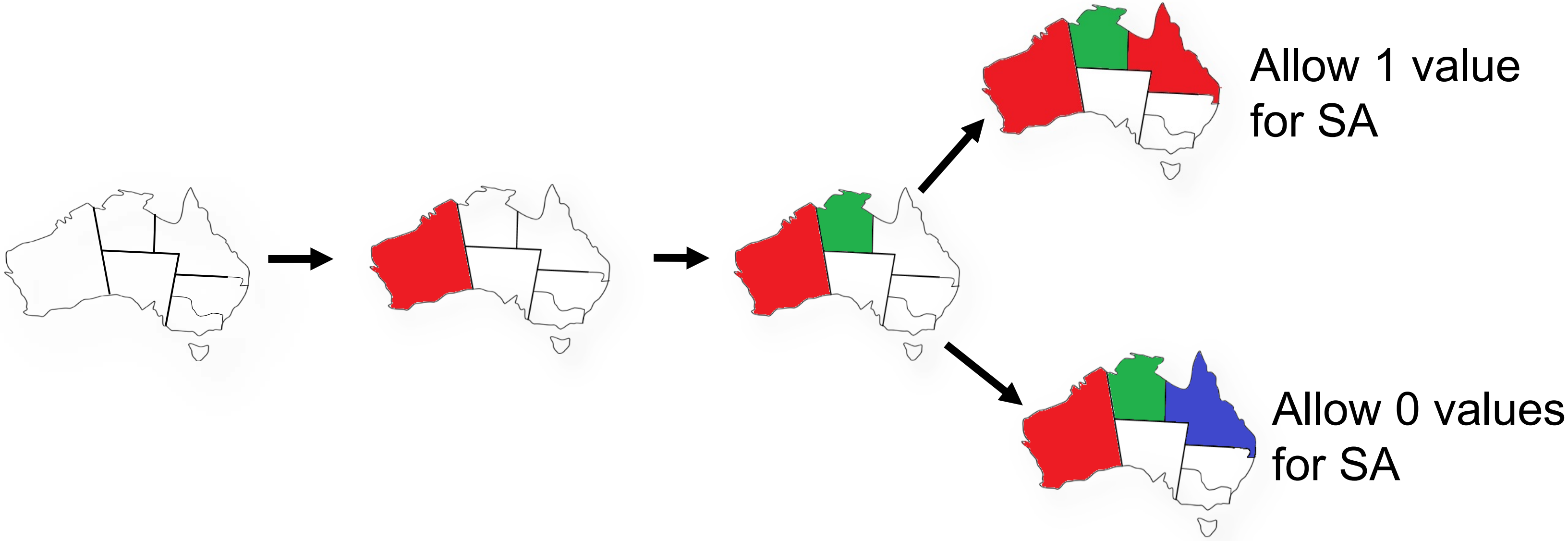
Or minimum remaining values (MRV) heuristic

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.

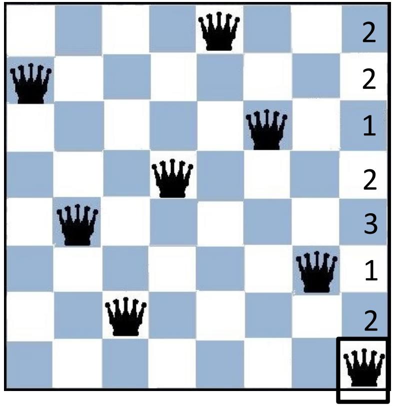
Least Constraining Value

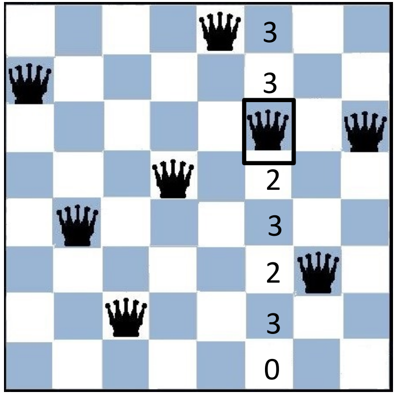


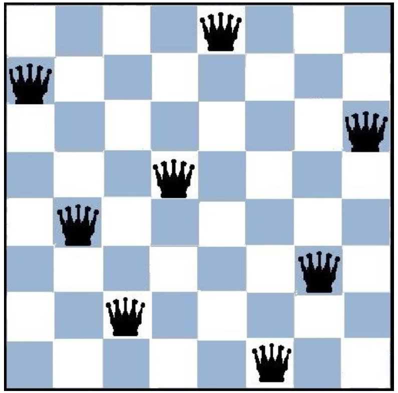
Choose the value that leaves maximum flexibility for subsequent variable assignments

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







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

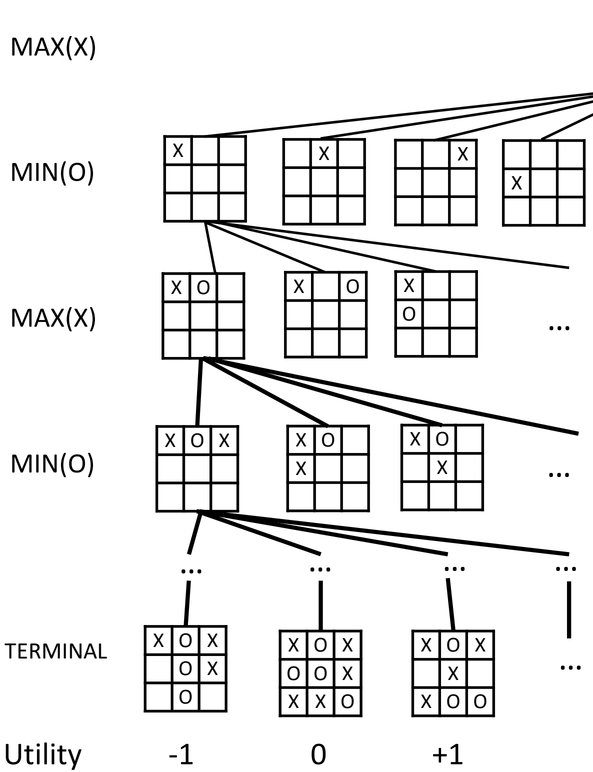
Game as a Search Problem

- 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

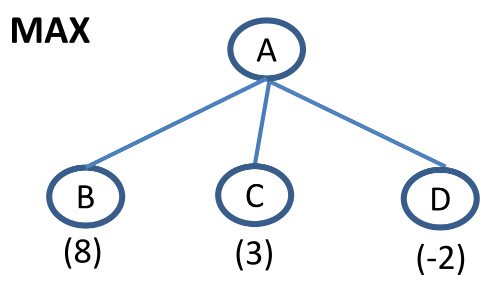


- The utility value of the terminal state is from the point of view of MAX

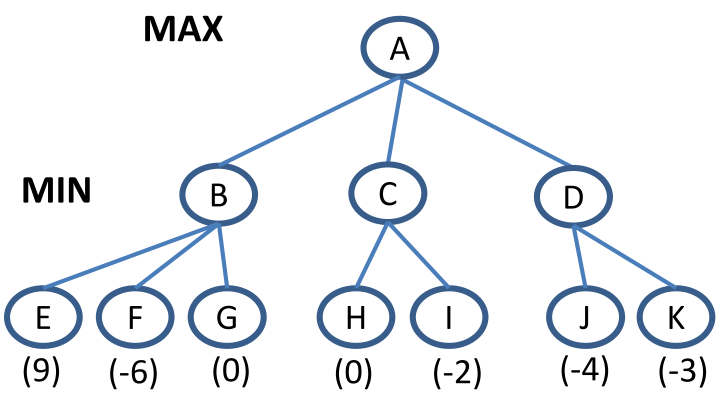
- MAX uses the search tree to determine the best move

What Search Strategy?

One-play



Two-play



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

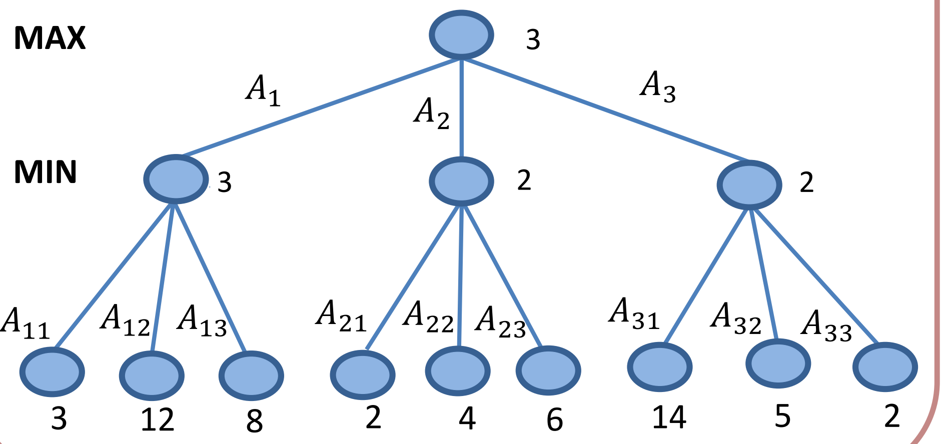
Perfect decisions: no time limit is imposed

- 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

	X	O	
	O	X	

- 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

	X	O	
	O	X	

O considers the game now

	X	O	
	X	X	
	X		

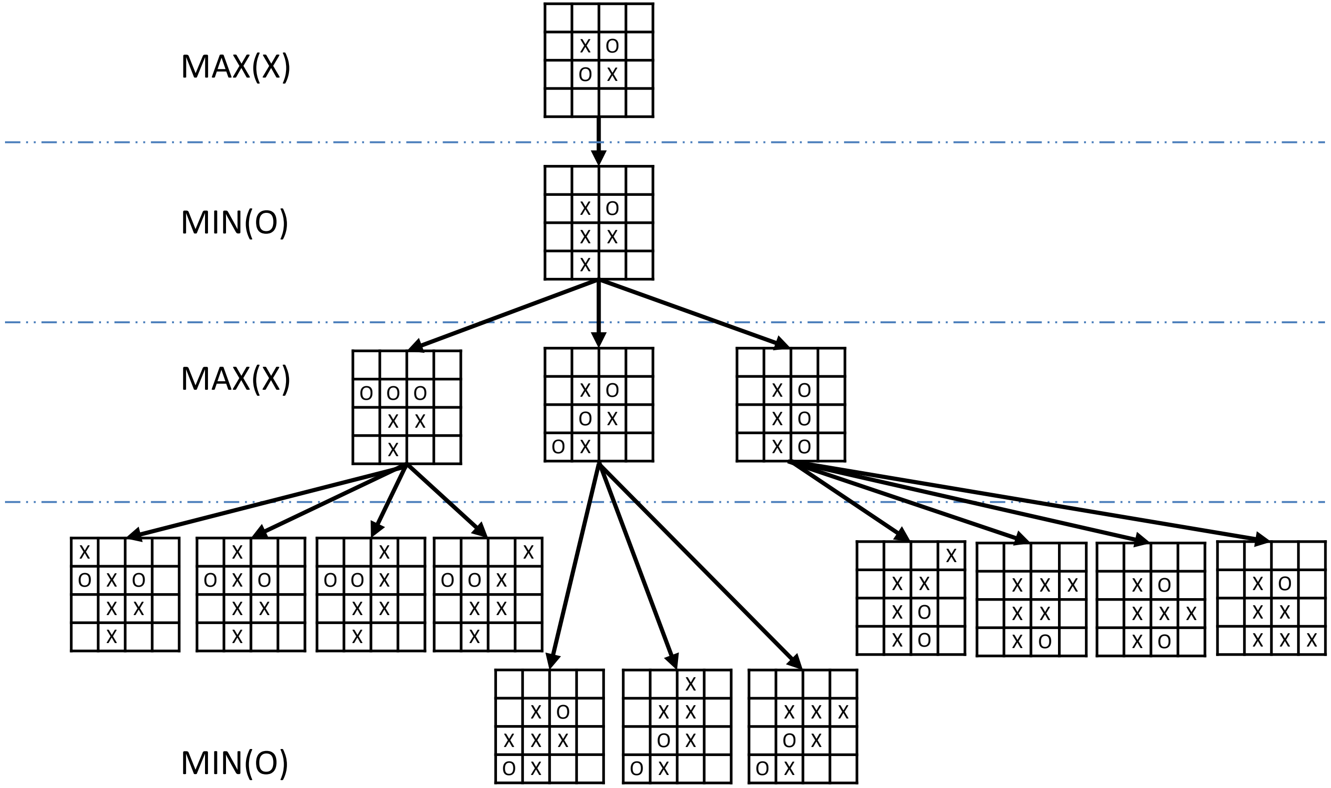
X considers the game now

O	O	O	
	X	X	
	X		

	X	O	
	O	X	
O	X		

	X	O	
	X	O	
	X	O	

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

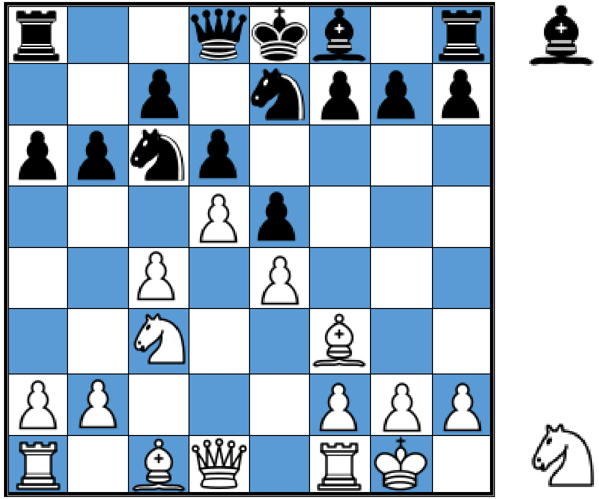
Time/space requirements → complete game tree search is intractable → impractical to make perfect decisions

Modifications to minimax algorithm

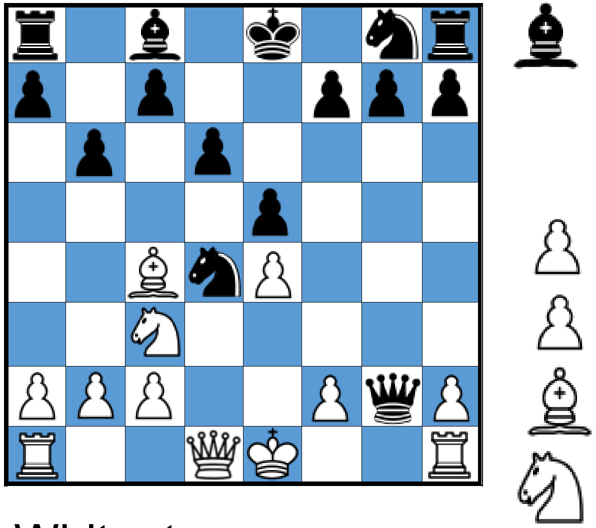
1. replace utility function by an estimated desirability of the position
   - Evaluation function
2. partial tree search
   - E.g., depth limit
   - Replace terminal test by a cut-off test

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

AlphaGo: Key Ideas

- Objective: Reduce search space without sacrificing quality
- Key Idea 1: Take advantage of human top players' data
  - Deep learning
- Key Idea 2: Self-play
  - Reinforcement learning
- Key Idea 3: Looking ahead
  - Monte Carlo tree search
  - We learned Minimax search with evaluation functions