

# AI Concepts and Techniques

Lesson 7: Explaining problems solving techniques in AI  
Informed search and Local search

Lesson	Date	Topics
1	18-Apr-22	Introduction to AI
2	21-Apr-22	Basics of Python
3	25-Apr-22	Linear Algebra + Python for AI (I)
4	28-Apr-22	Python for AI (II)
5	4-May-22	AI approaches
6	6-May-22	AI concepts and techniques Agents AI & Problem Solving Search problems Uninformed search
<b>7</b>	<b>9-May-22</b>	<b>AI concepts and techniques</b> <b>Informed search</b> <b>Local search</b>
8	12-May-22	AI concepts and techniques Constraint satisfaction problems Knowledge-based agents
9	17-May-22	Written Test
10	23-May-22	Future of AI, revision and review

# Recap Uniformed Searching

---

## Performance Measure

- Completeness – Ability to find always find a solution.
- Optimality – Will find the least-cost solution.
- Time Complexity – Time needed.
- Space Complexity – Storage space needed for generated nodes.

## Uninformed Strategies

- Breadth first (FIFO)
- Depth First (LIFO)
- Depth Limited
- Iterative Deepening
- Uniform Cost (Priority)
- Bidirectional Search

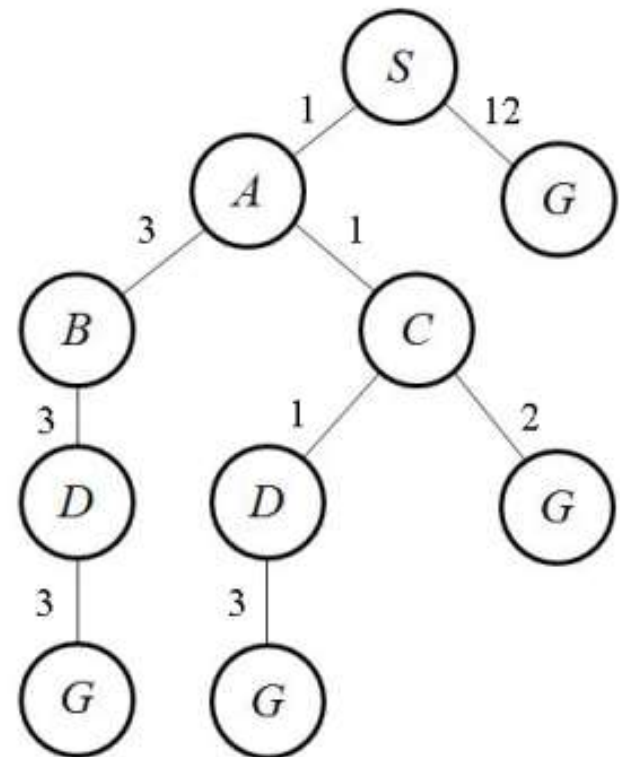
# Uninformed Search Strategies

## Uniform Cost (Priority)

- Incorporates step costs
- Expand the least-cost unexpanded node
- Similar to Dijkstra algorithm, but with goal test
- Equivalent to breadth-first search if steps costs are equal

*S* is the initial state

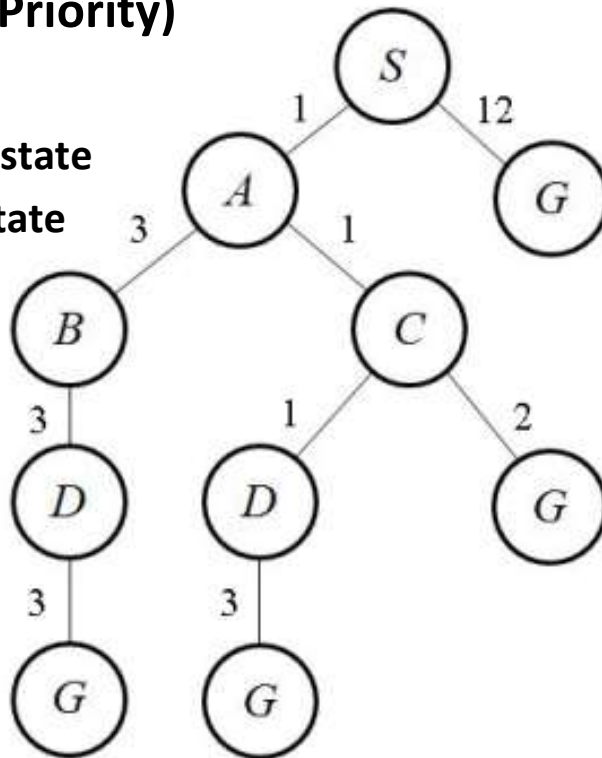
*G* is the end state



# Uninformed Search Strategies

## Uniform Cost (Priority)

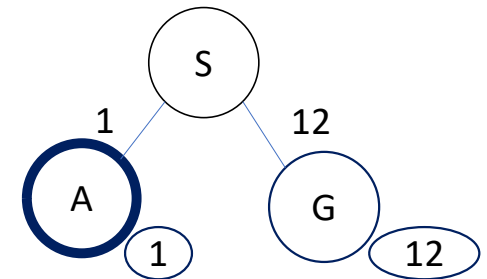
**S** is the initial state  
**G** is the end state



Visited Nodes

1. S,

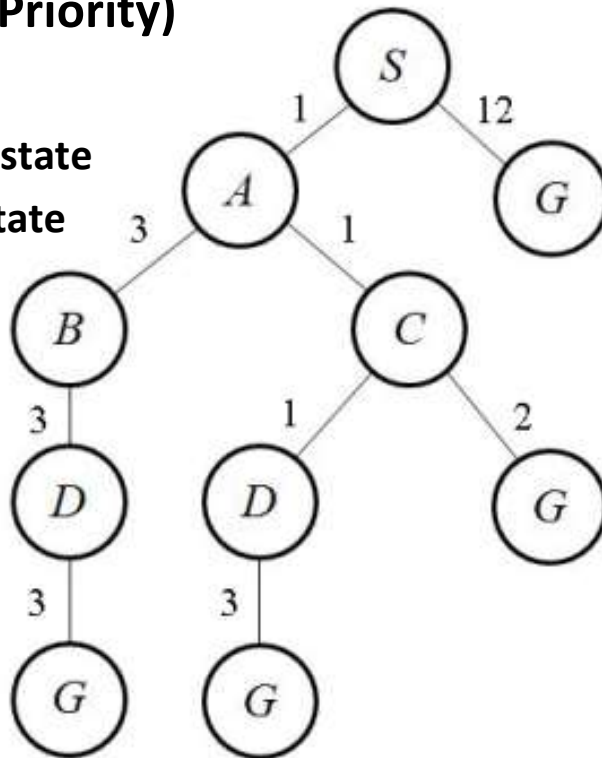
2. SA,



# Uninformed Search Strategies

## Uniform Cost (Priority)

**S** is the initial state  
**G** is the end state

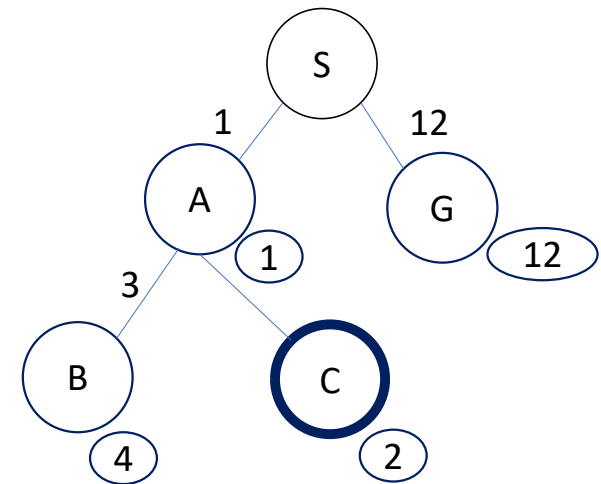


Visited nodes

1. **S**

2. **SA**

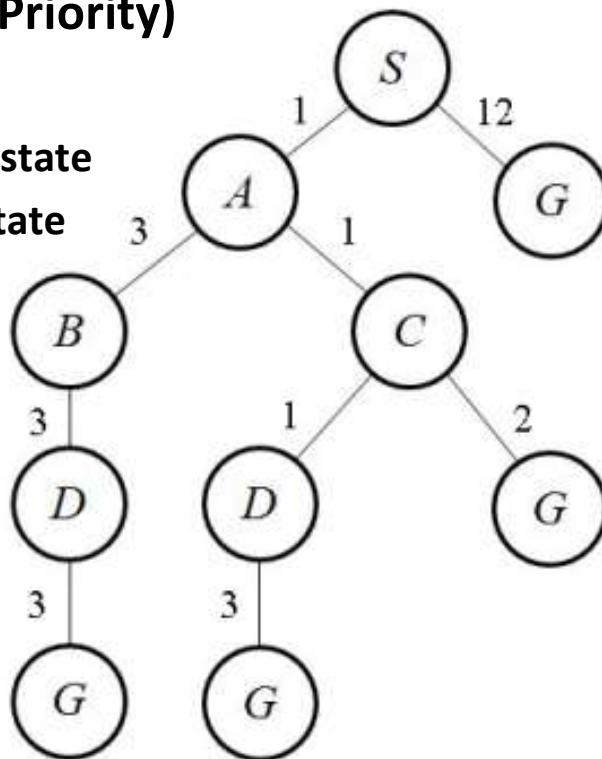
3. **SAC**



# Uninformed Search Strategies

## Uniform Cost (Priority)

**S** is the initial state  
**G** is the end state



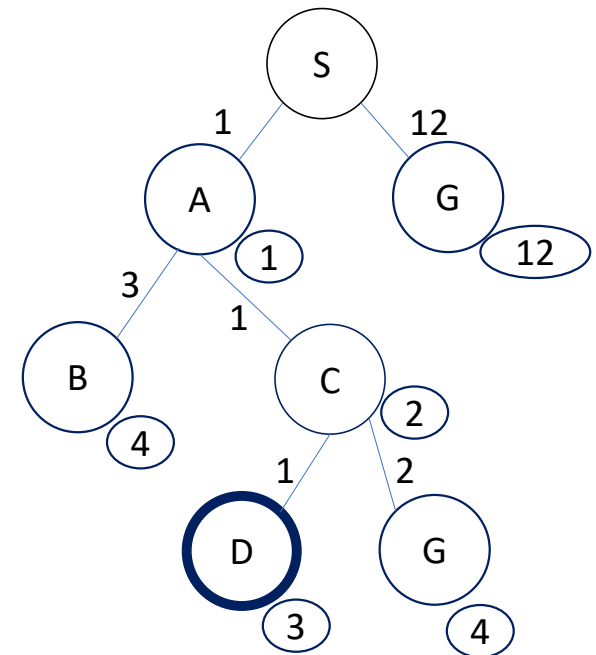
Visited nodes

1. S

2. SA

3. SAC

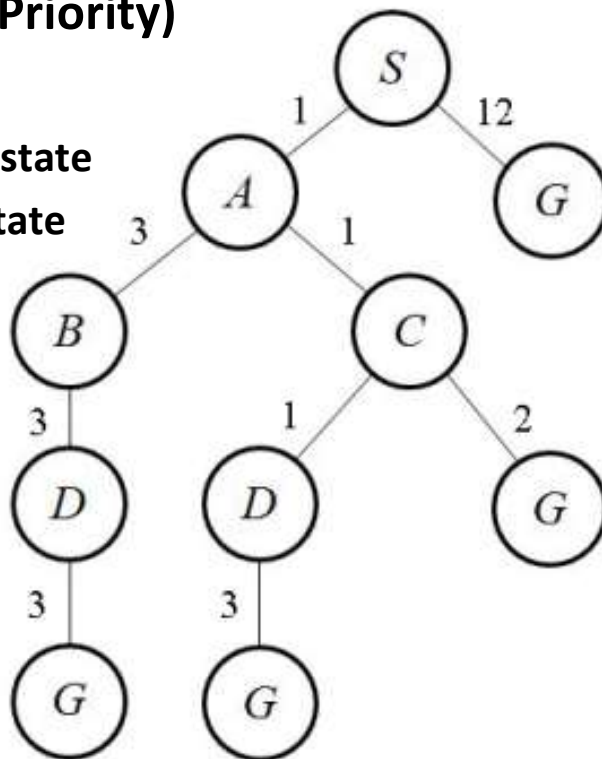
4. SACD



# Uninformed Search Strategies

## Uniform Cost (Priority)

**S** is the initial state  
**G** is the end state



Visited nodes

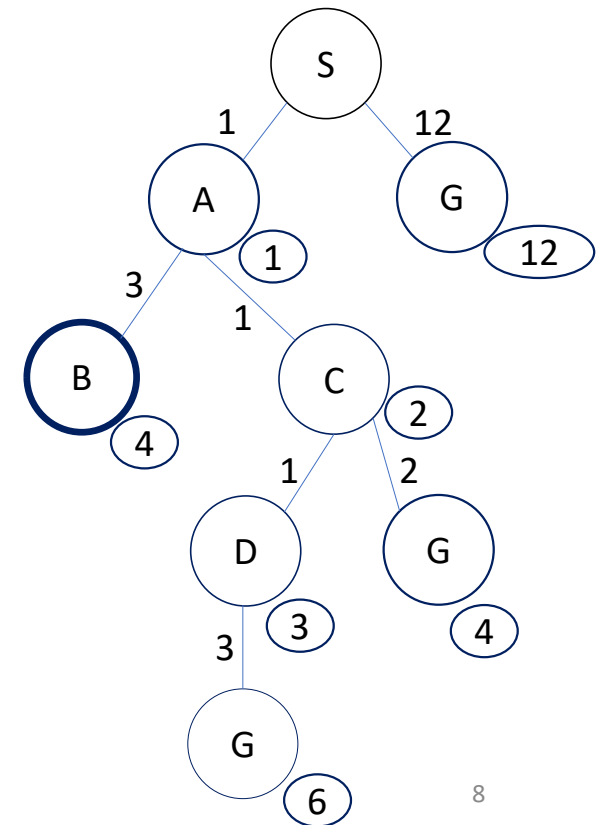
1. S

2. SA

3. SAC

4. SACD

5. SAB

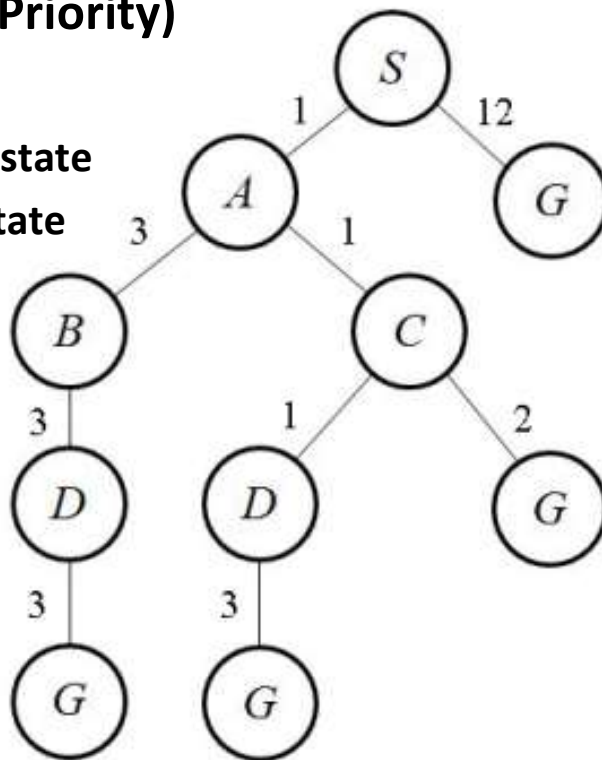




# Uninformed Search Strategies

## Uniform Cost (Priority)

**S** is the initial state  
**G** is the end state



Visited nodes

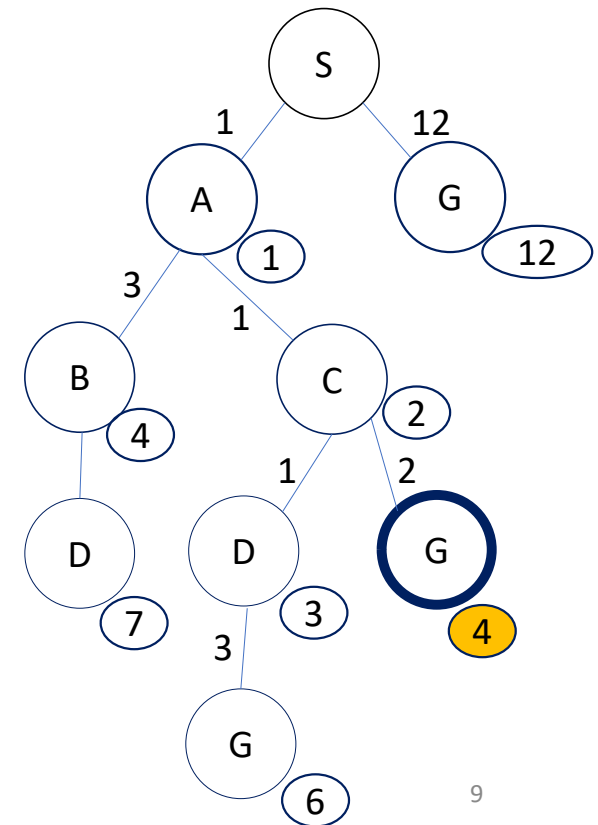
1. S,

2. SA,

3. SAC

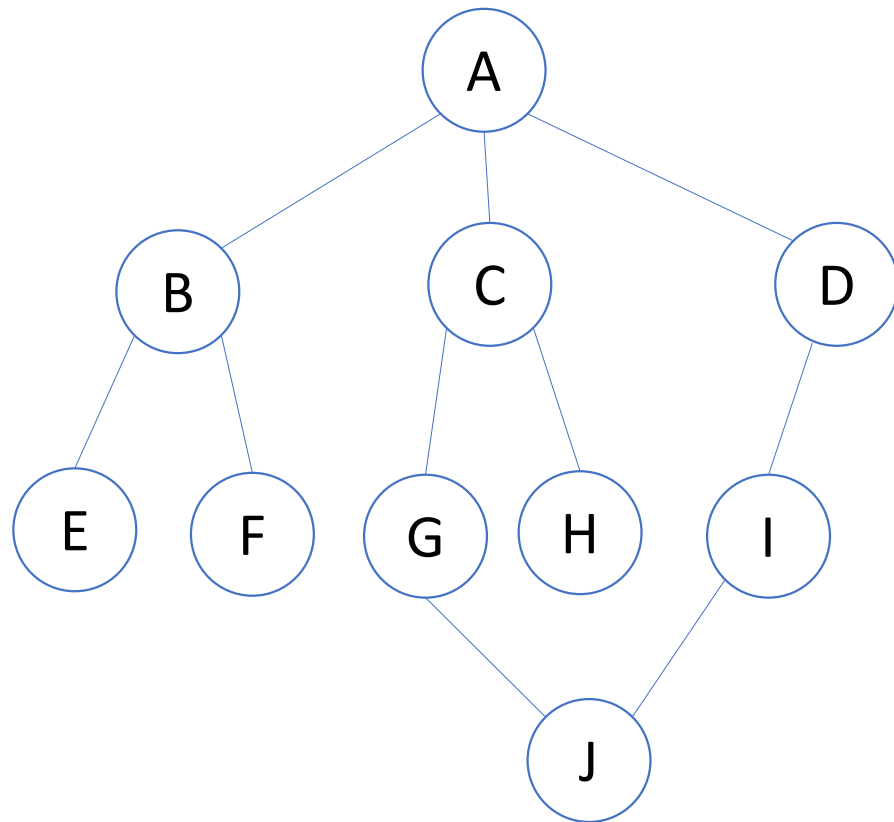
4. SACD

5. SAB



# Exercise

- Consider the different uniformed search strategies.
- What order would the nodes be expanded for depth first, and iterative deepening?



# Uninformed Search Strategies

Source: Artificial Intelligence a Modern Approach

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes <sup>a</sup>	Yes <sup>a,b</sup>	No	No	Yes <sup>a</sup>	Yes <sup>a,d</sup>
Time	$O(b^d)$	$O(b^{1+\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes <sup>c</sup>	Yes	No	No	Yes <sup>c</sup>	Yes <sup>c,d</sup>

**Figure 3.21** Evaluation of tree-search strategies.  $b$  is the branching factor;  $d$  is the depth of the shallowest solution;  $m$  is the maximum depth of the search tree;  $\ell$  is the depth limit. Superscript caveats are as follows: <sup>a</sup> complete if  $b$  is finite; <sup>b</sup> complete if step costs  $\geq \epsilon$  for positive  $\epsilon$ ; <sup>c</sup> optimal if step costs are all identical; <sup>d</sup> if both directions use breadth-first search.

# Uninformed Search Strategies

Depth	Nodes	Time	Memory
0	1	1 millisecond	100 kbytes
2	111	0.1 second	11 kilobytes
4	11,111	11 seconds	1 megabyte
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	11,111 terabytes

# Informed vs Uniformed Search

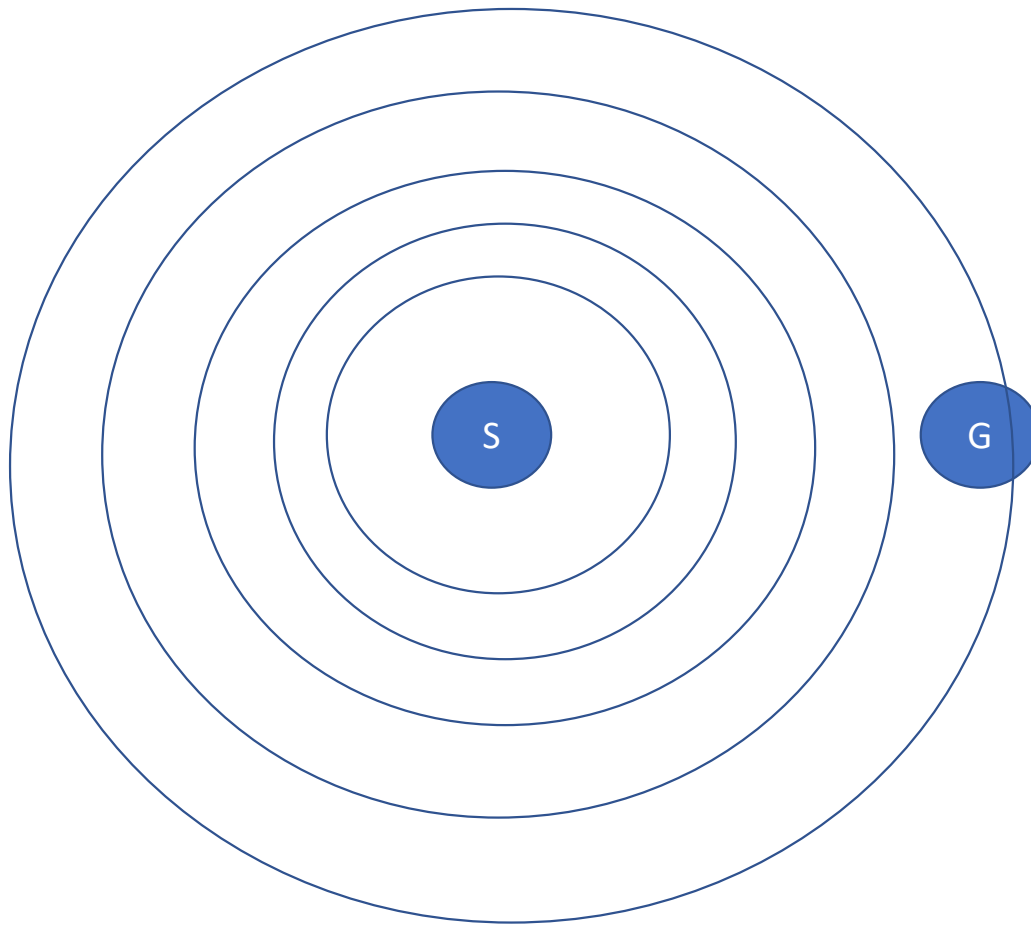
---

**An informed search strategy is one that is capable of identifying the promising branches of a search space**

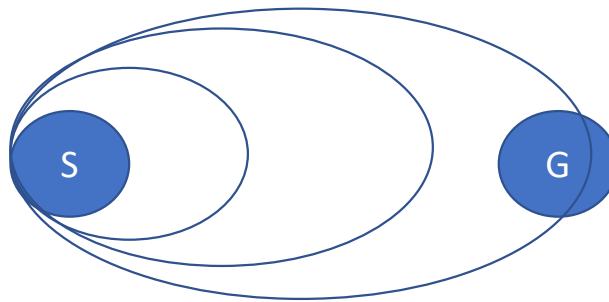
A search strategy that can search the most promising branches of the state-space first can:

- Find a solution more quickly
- Find solutions when there is limited time
- Often find a better solution (it ignores the unpromising search areas so avoids alternative bad solutions)

# Uninformed Search



# Informed Search



# Informed vs Uniformed Search

**Informed search is more efficient. Available only when specific data on the problem is known.**



**Uniformed search is a blind or brute-force search**





# Informed vs Uniformed Search

**Informed search is more efficient. Available only when specific data on the problem is known.**



Uses specific data (knowledge) beyond the problem definition

Expands a node based on evaluation function  $f(n)$  that often contains a heuristic function  $h(n)$

Heuristic function is the additional knowledge imparted to the search algorithm

# Informed Search Strategies

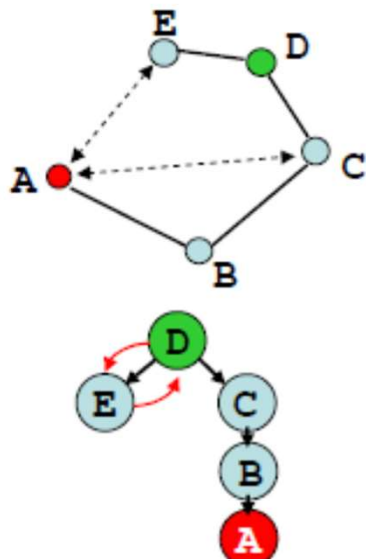
---

## **Common types of Informed (Heuristic) Search**

- **Greedy Best-First**
- **A\***
- **Space Saving Techniques**

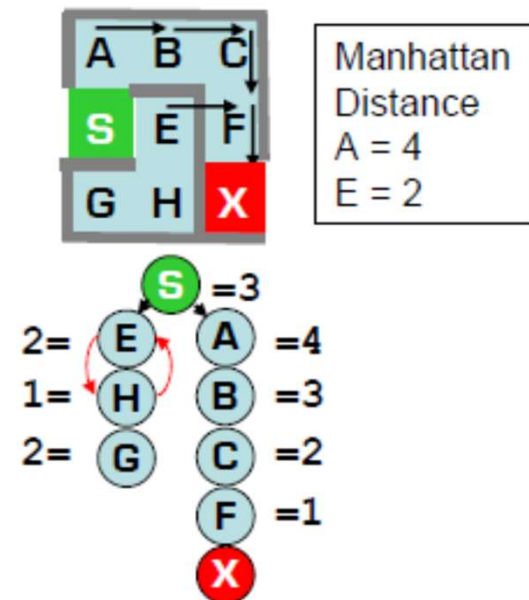
# Example Heuristics

- Straight-line distance
- Manhattan Distance



Problem Space

Search Tree



# Informed Search Strategies – Greedy Best First

Evaluates nodes by using just heuristics

$$f(n) = h(n)$$

$h(n)$  is the straight line distance heuristic.

Expand the node closest to the solution



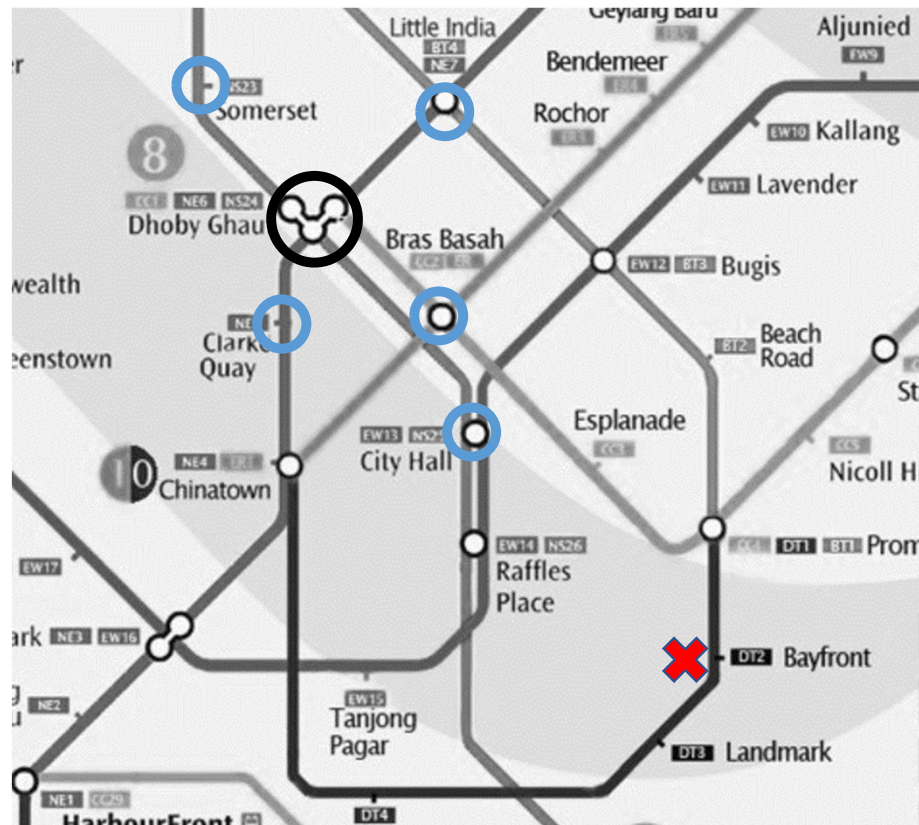
# Informed Search Strategies – Greedy Best First

**To get from Dhoby Ghaut to Bayfront, which node (station) will be picked first?**

**Connected station from Dhoby Ghaut:**

***{Clarke Quay, Brash Basah, City Hall, Little India, Somerset}***

**City Hall will be picked first as it has the shortest straight line distance (at least on this map) to Bayfront.**



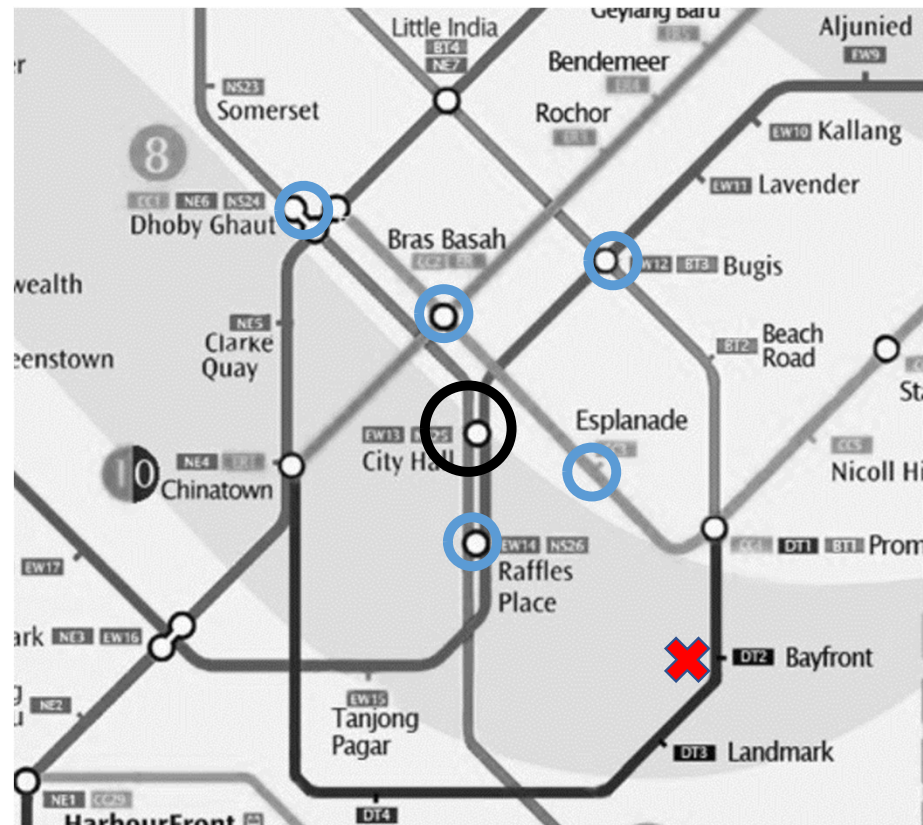
# Informed Search Strategies – Greedy Best First

**Expanding Dhoby Ghaut leads to**

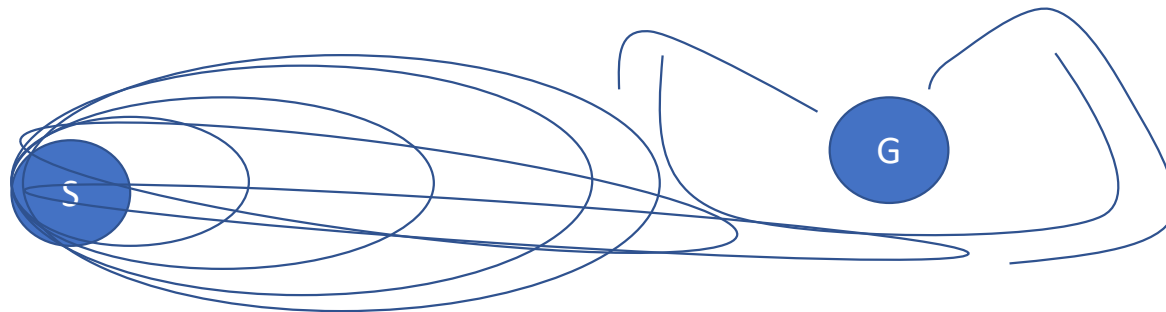
- Clarke Quay
- Brash Basah
- City Hall,
- Little India
- Somerset

**Expand City Hall ....**

**→ Esplanade**



# Informed Search Strategies – Greedy Best First



“Greedy” can lead to worse results than being “careful” because solution may not be optimal

# Informed Search Strategies – A\*

## A\*

Node to be expanded is chosen based on an evaluation function based on the goal function and the cost to reach the node

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

Idea is to avoid expanding paths that are already expensive.

$g(n)$  = cost so far to reach  $n$

$h(n)$  = estimated cost from  $n$  to goal

$f(n)$  = estimated total cost of path through  $n$  to goal





# Informed Search Strategies

---

How do we choose the heuristics? *To ensure that A\* search is optimal, the heuristic must satisfy certain conditions: Admissibility and Consistency*

**Heuristic  $h(n)$  is admissible** if for every node  $n$ :

$$h(n) \leq h^*(n)$$

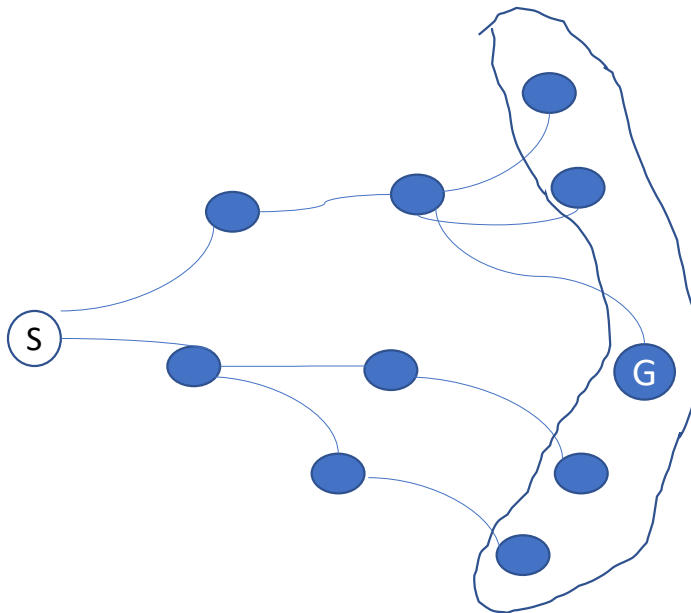
where  $h^*(n)$  is the true cost to reach the goal state from  $n$

i.e. the heuristic **never overestimates the cost to reach the goal.**

**Heuristic is optimistic.**

**Question: Is straight line distance heuristic admissible?**

# Intuition of Admissibility Condition



$$f = g + h$$
$$h(s) < \text{true cost}$$
$$S \rightarrow G$$

If  $h$  is optimistic, then the estimated cost is always less than the true cost, so the path  $p$  must have a cost that is less than the true cost of any other paths on the front tier. Any paths that go beyond the frontier, must have a cost greater than that, because we agree that the step cost is always 0 or more. So that means this path  $P$  must be the minimum cost path.

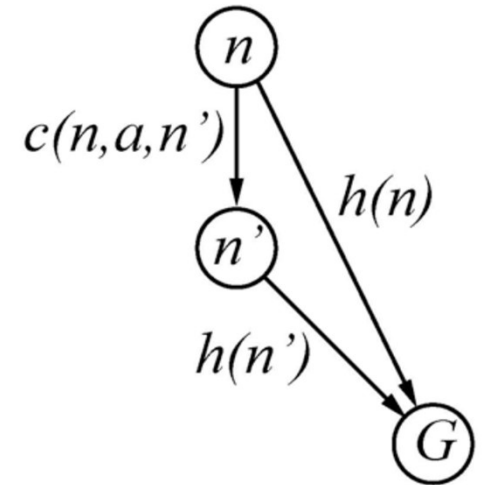
# Informed Search Strategies

How do we choose the heuristics?

Heuristic is consistent if for every node  $n$ , every successor  $n'$  of  $n$  generated by action  $a$

$$h(n) \leq c(n, a, n') + h(n')$$

i.e. the estimated cost of reaching the goal from  $n$  is no greater than the step cost of getting to  $n'$  plus the estimated cost of reaching the goal from  $n'$



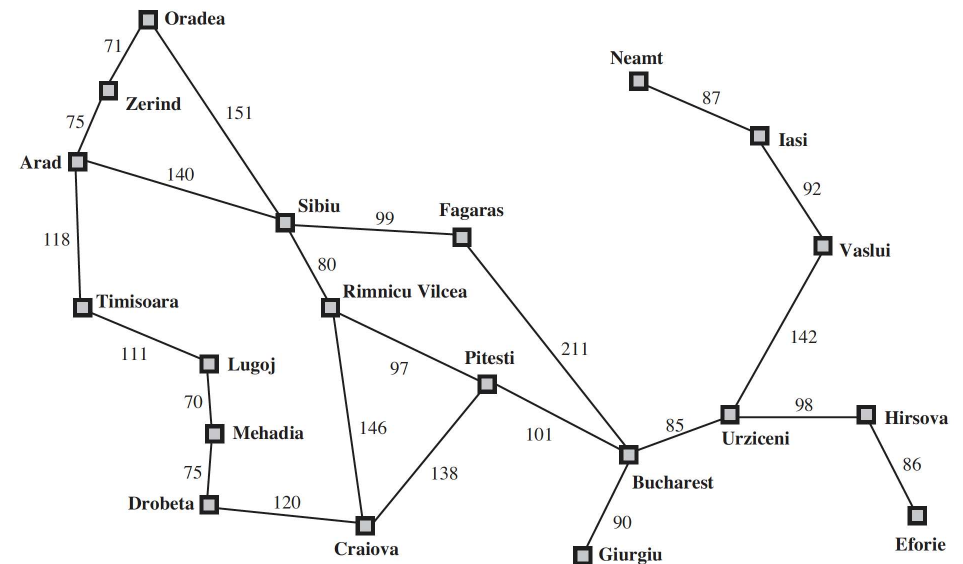
# Exercise

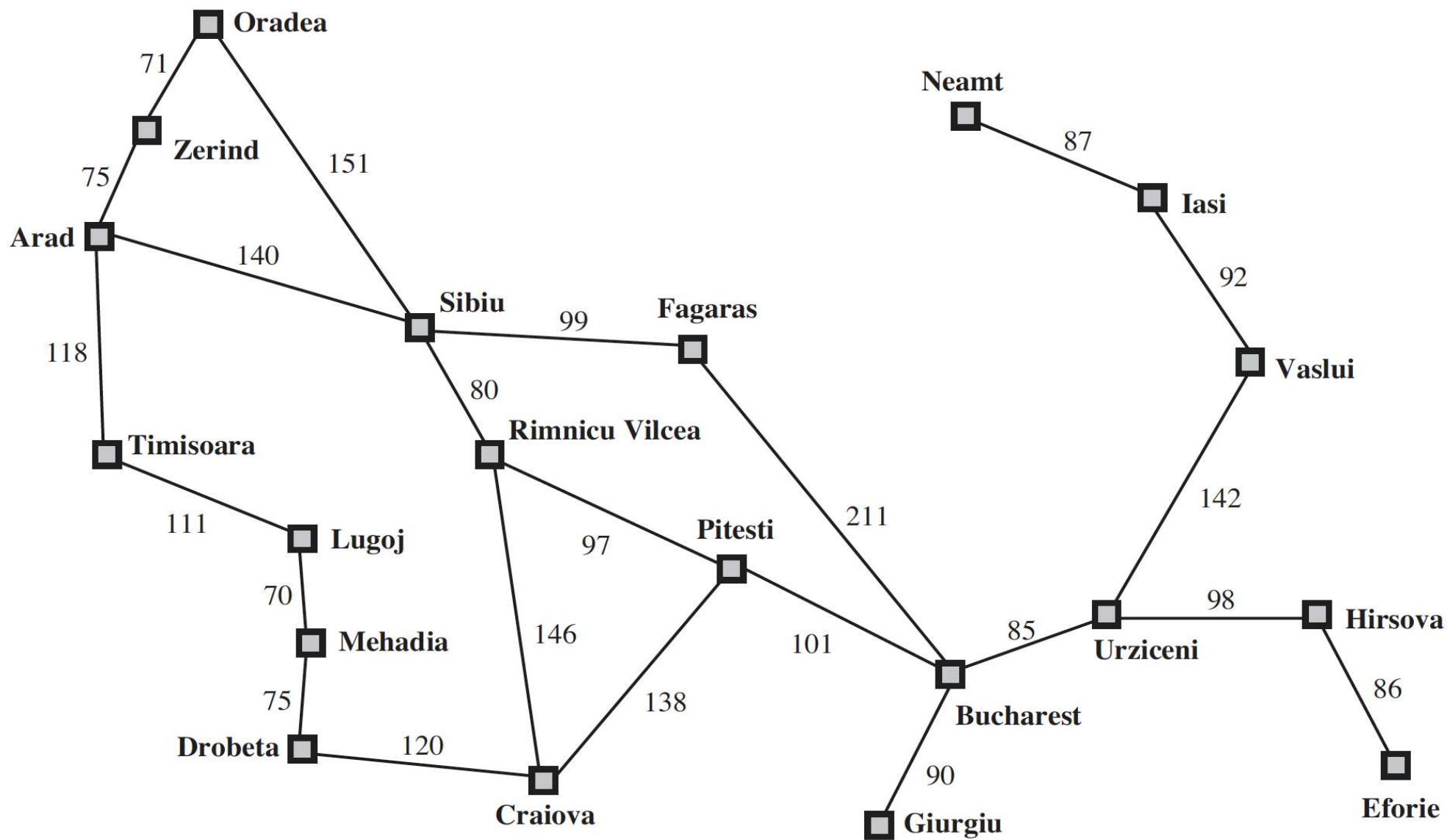
## a) Goal: To get to Bucharest from Arad

Use A\* search to find the path. Clear working will help greatly.

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

Straight line distance





# Informed Search Space Saving Strategies

---

## Space Saving Techniques

- Iterative-deepening A\* (IDA\*)
- Recursive best-first search (RBFS)
- Memory bound A\* (MA\*)
- Simplified memory bound A\* (SMA\*)

# Informed Search Space Saving Strategies

## Iterative-deepening A\* (IDA\*)

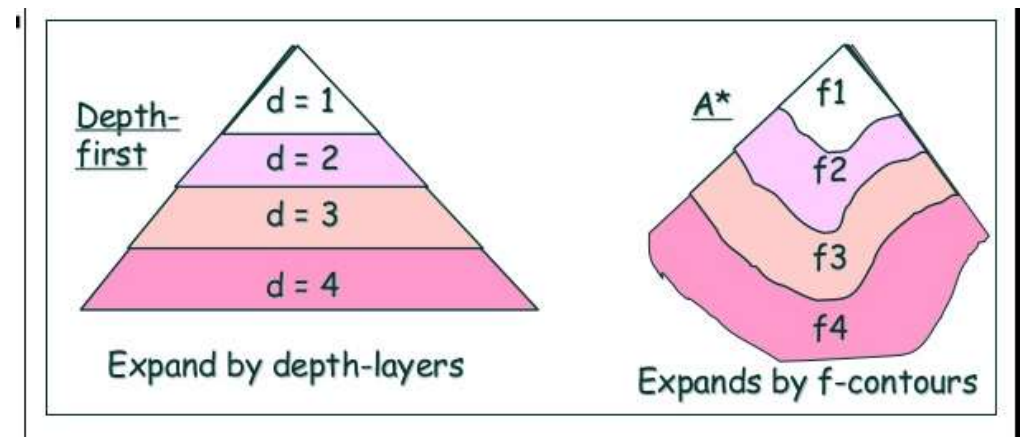
IDA\* is complete, optimal and optimally efficient.  
Space usage is linear.

Combines the advantages of A\* and iterative deepening

## Depth limited search using

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

IDA\* gives benefits of A\* without the requirement to keep all reach states in memory at a cost of visiting some states multiple times. It is a very important and commonly used algorithm for problems that don't fit in memory.



# Informed Search Space Saving Strategies

---

## **Recursive best-first search (RBFS)**

- **Best first search**
- **Limits recursion by keeping track of the  $f$  value of the best alternative path from any ancestor node.**

## **Memory bound $A^*$ ( $MA^*$ )**

## **Simplified memory bound $A^*$ ( $SMA^*$ )**



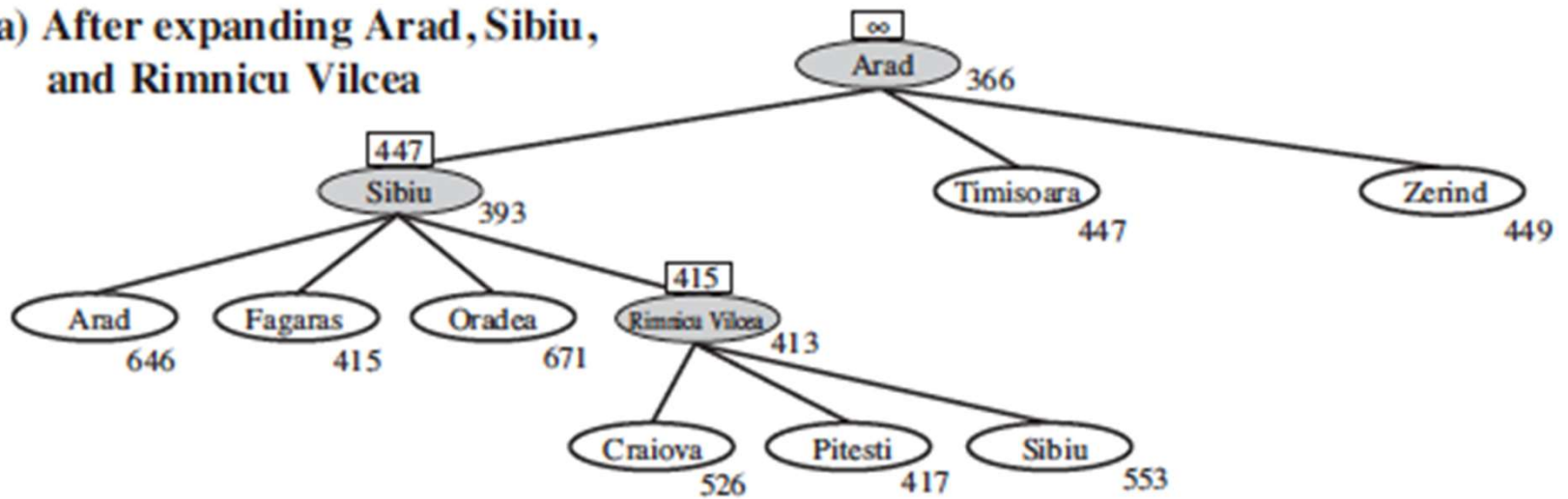
## Recursive best-first search (RBFS)

---

- Similar to a depth-first search
- Limits recursion by keeping track of the f-value of the best alternative path from any ancestor node.
- If current node exceeds limit, recursion winds back to alternative path
- A recursion unwinds, RBFS replaces the the f-value of each node along the path with the best f-value of its children.

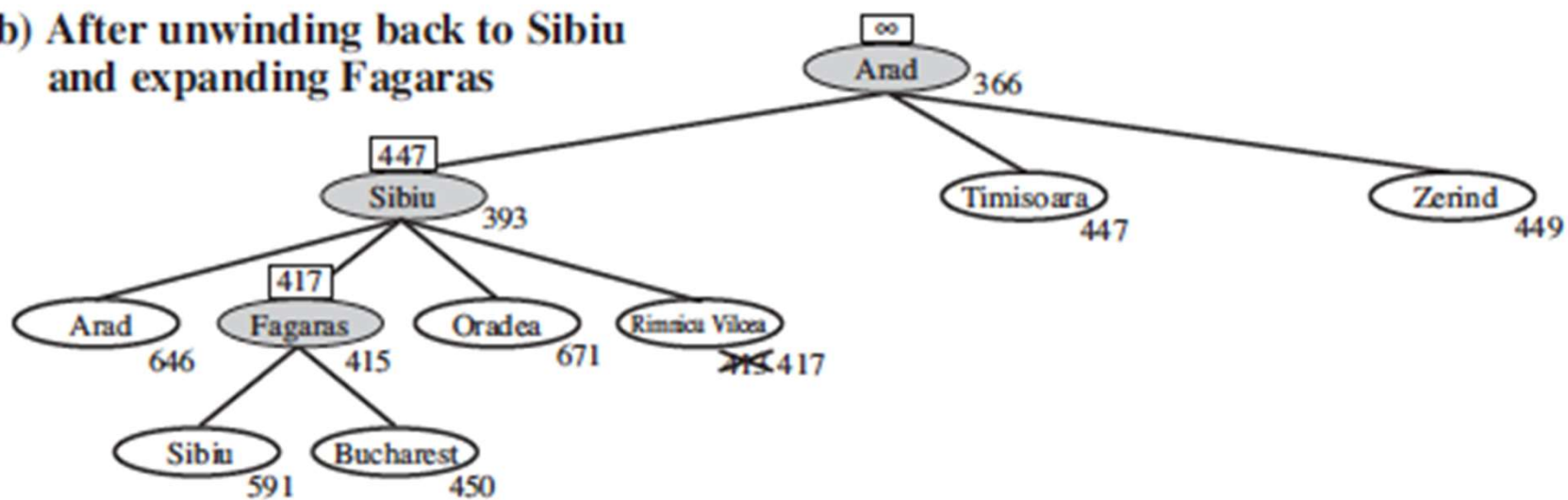
# Recursive best-first search (RBFS)

(a) After expanding Arad, Sibiu, and Rimnicu Vilcea



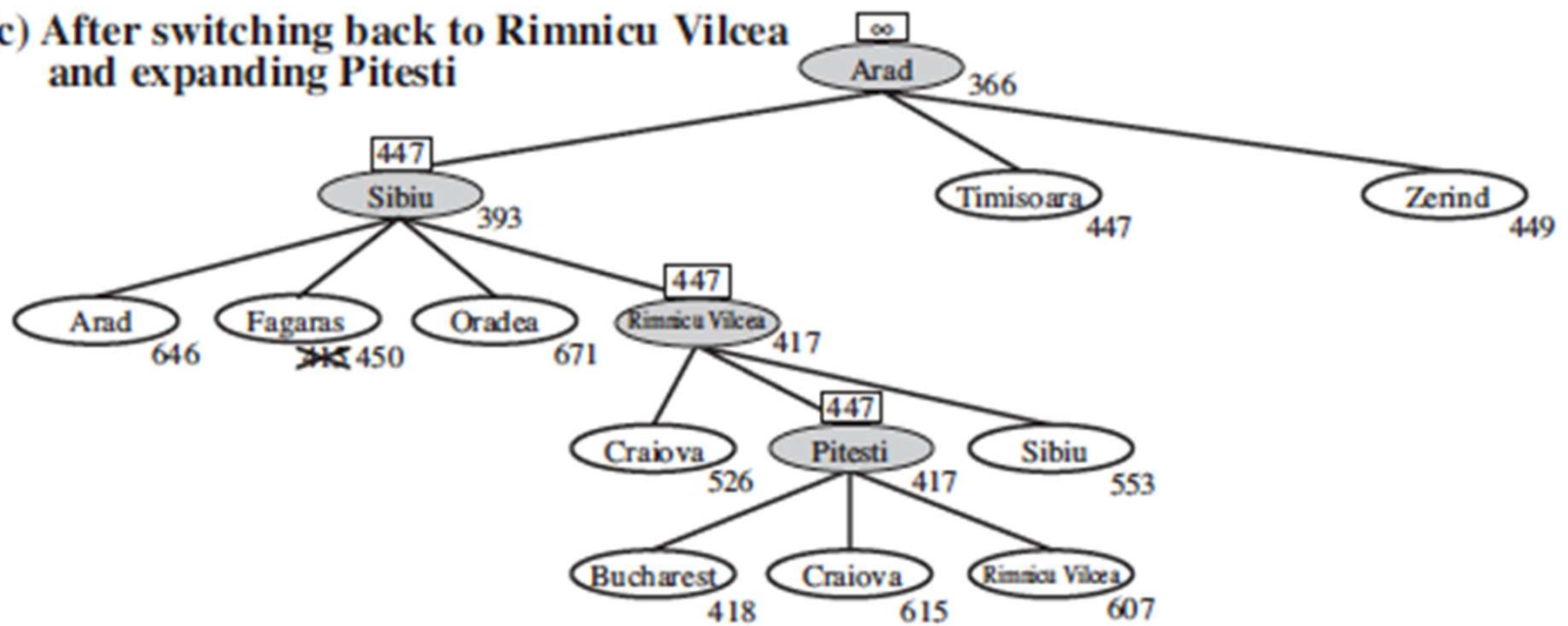
## Recursive best-first search (RBFS)

(b) After unwinding back to Sibiu and expanding Fagaras



## Recursive best-first search (RBFS)

(c) After switching back to Rimnicu Vilcea and expanding Pitesti



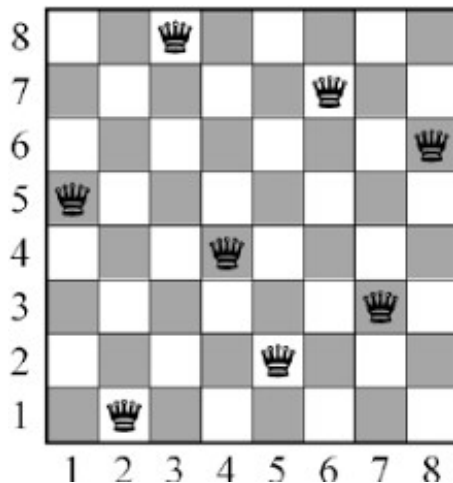
## Simplified memory bound A\*

---

- Proceed like A\* until all memory is used up.
- When memory full, drops the worst leaf node (highest f-value)
- Like RBFS, backs up the value of the forgotten node to the parent node

# Local Search

What if we don't care about the path, only the solution?



Examination Timetable			
Date	Time	Subject	Venue
5/5/2011	8-9am	English	Hall
6/5/2011	8-9.30am	Higher Chinese	Seminar Room 1 & 2
	9.30am-10.30am	Chinese	Theatrette
	10.30am-11.30am	Malay	Theatrette
	2.30-4pm	Math	Theatrette
	8-9.30am	Humanities	Theatrette
10/5/2011	8-9.30am	Physics	Theatrette
12/5/2011	8-9am	Chinese (3rd Lang.)	Seminar Room 2
	8-9.30am	French	Seminar Room 2
	8-9.30am	Japanese	Seminar Room 2
	8-9am, 9.30-10.30am	Malay (3rd Lang.)	Ea-10/11



# Local search algorithms

---

**Search problems can be formulated in terms of optimization. In this case we are not concerned with start state or the path taken. We only care about the solution (goal).**

Recall that the search space contains all the feasible solutions. Local search algorithm move from one solution to another by applying local changes until an optimal solution is found (or when constrained by time)

**To evaluate the potential answers we need an objective function that tells us about the solution**

**e.g. 8 Queens: # of queens “attacking”**

**e.g. VLSI: size of circuit board**

**The solution (goal) is found by minimizing or maximizing this function.**



# Local search algorithms

---

- Keep “current” state and try to improve it
- Memory efficient as only remember “current” state
- Ignore previous states and path taken

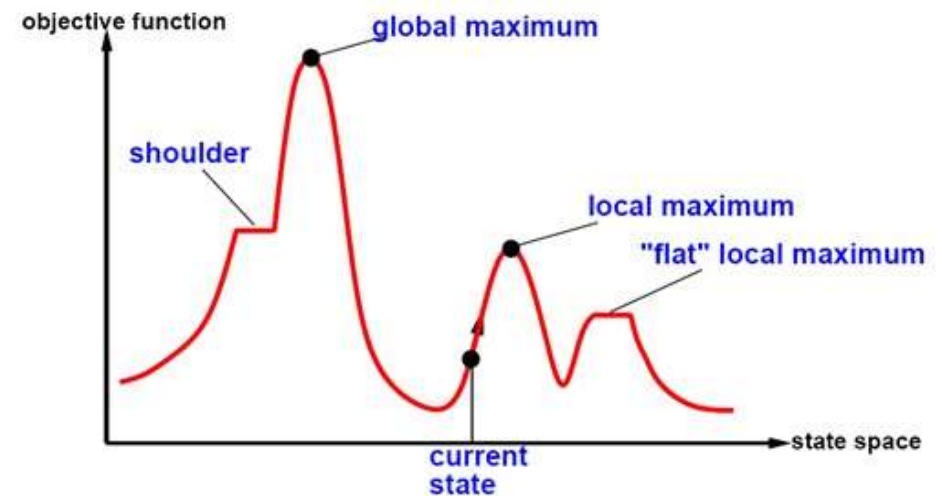
# Local search algorithms

---

- Hill climbing search
- Simulated annealing
- Local beam search

# Hill-Climbing Algorithms

- Choose the 'better' neighbour, move there, don't remember the path.
  - Evaluate the initial state
  - Loop until a solution is found or there is no new operator left to apply.
    - Select and apply an operator
    - Evaluate new state
      - If is goal → exit with success
      - If better than current state → set as new current state
- Issues:
  - Local max/ min, ridges, plateaus
- Variations:
  - Steepest-ascent hill climbing
  - Stochastic hill climbing



**May not give the best solution but a good solution in reasonable time**

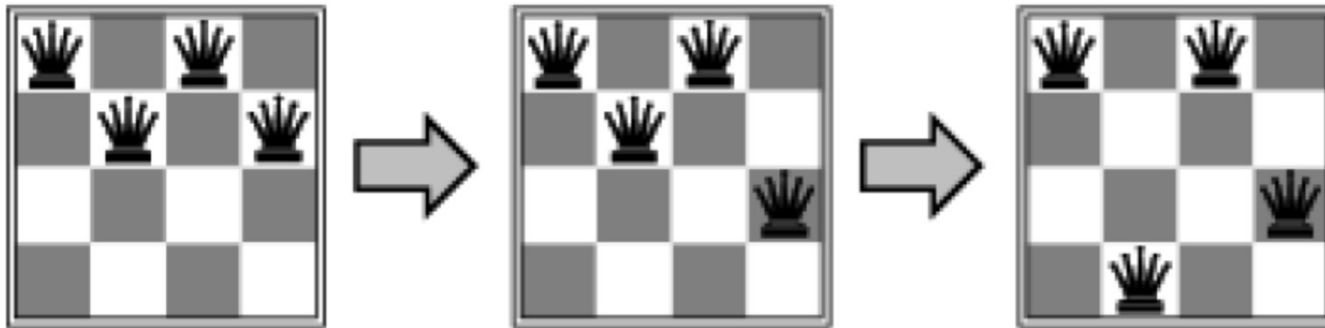
# n-Queens Problem

Goal:

**Put  $n$  queens on an  $n \times n$  board with no two queens on the same row, column, or diagonal**

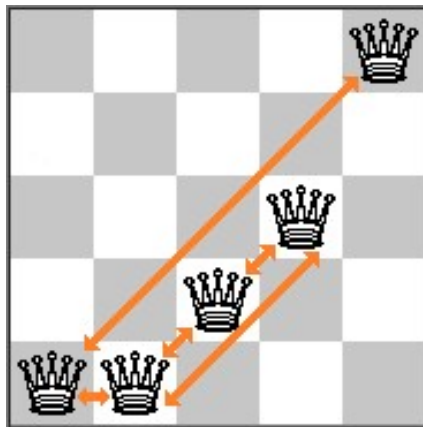
Neighbor: move one queen to another row

Search: go from one neighbor to the next...

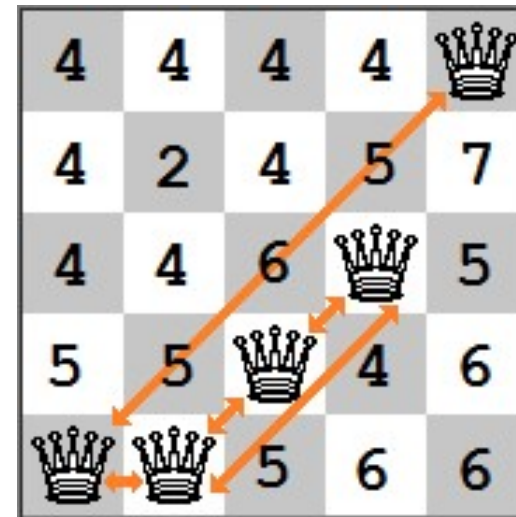


**Cost =  $x$**

# Hill Climbing – n Queens



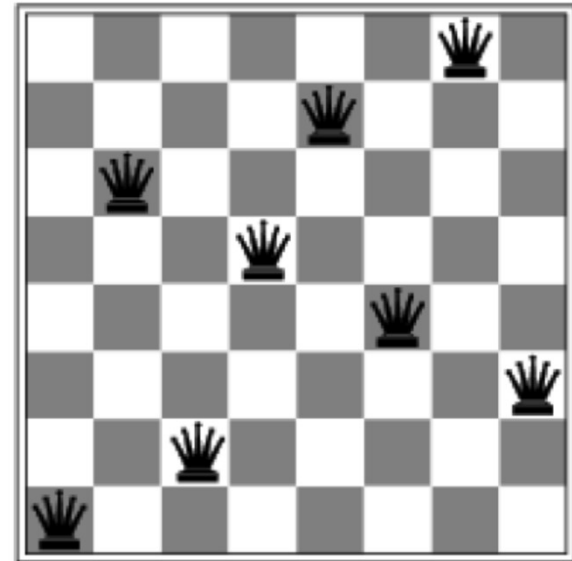
**Cost = 5**



## Hill Climbing – n Queens

18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	♙	13	16	13	16
♙	14	17	15	♙	14	16	16
17	♙	16	18	15	♙	15	♙
18	14	♙	15	15	14	♙	16
14	14	13	17	12	14	12	18

What's the  
Problem?



What's the  
Problem?

# Hill-Climbing Variations (Avoiding local minima)

## Stochastic hill climbing

- Random selection among the uphill moves
- Can change probability with steepness

## Random walk hill climbing

- At each step either
  - Choose best neighbour (probability  $p$ )
  - Choose a random neighbour (probability  $1-p$ )

18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	13	13	16	13	16
17	14	17	15	14	16	16	16
17	14	16	18	15	14	15	16
18	14	15	15	14	14	16	16
14	14	13	17	12	14	12	18

# Hill-Climbing Variations (Avoiding local minima)

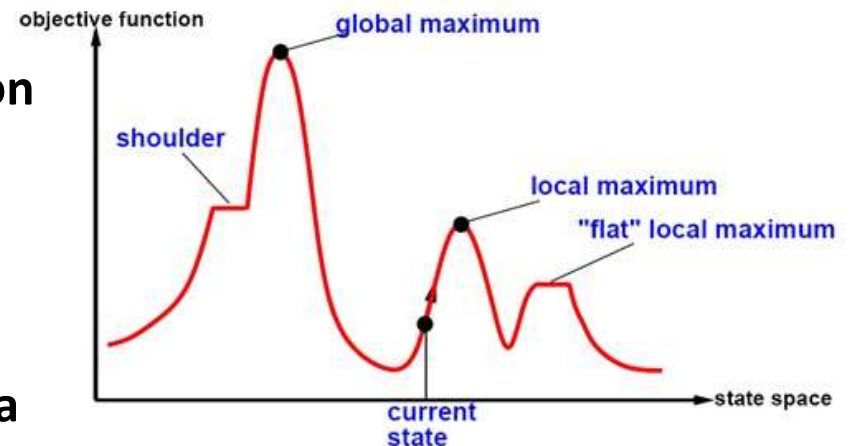
## Random Restart

*If at first you don't succeed try, try again.*

- Run for a fixed period of time (expected completion time) or until solution found
- Randomly restart

## Tabu Search (avoids endless loops)

- Remember x number of previous states in a list of fixed length
- Add newest state to the list – delete the oldest
- Never make the step that is currently tabu'd
- Worsening moves can be accepted

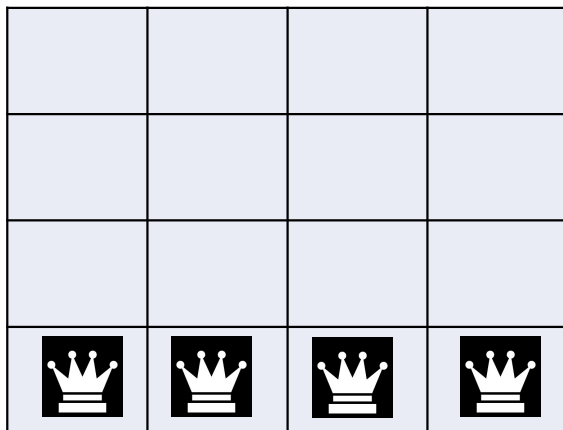




# Exercise





Use steepest ascent hill climbing approach to attempt and solve the following n-queen problem where  $n = 4$ . The heuristics will be the number of conflicting queen-pairs.

Draw or show the move towards the goal, together with the heuristic costs involved at each step. You may use python code to calculate the heuristic cost for a state if needed.

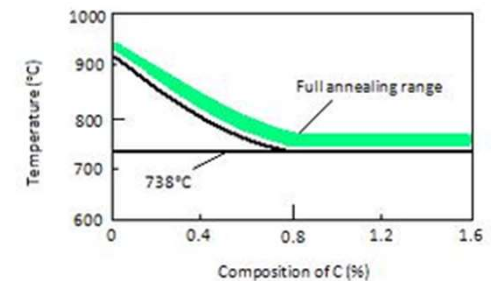
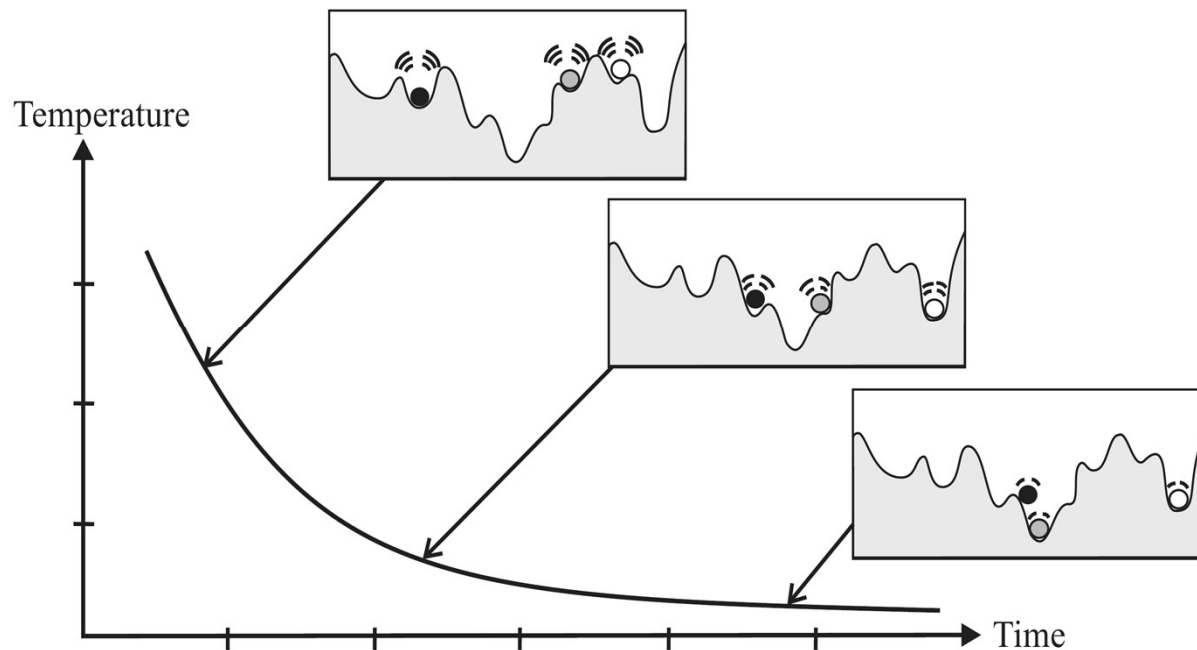


Initial state

Clearly list the heuristic costs involved in each step

4	3	3	4
4	4	4	4
4	5	5	4
			

# Simulated annealing



Annealing is the process of hardening metals by heating them to a high temperature and gradually cooling them, allowing the metal to reach a low-energy crystalline state. In simulated annealing solutions, we “shake hard” at higher temperature and less so at lower temperature.

# Simulated annealing

---

Similar to stochastic and random walk hill-climbing

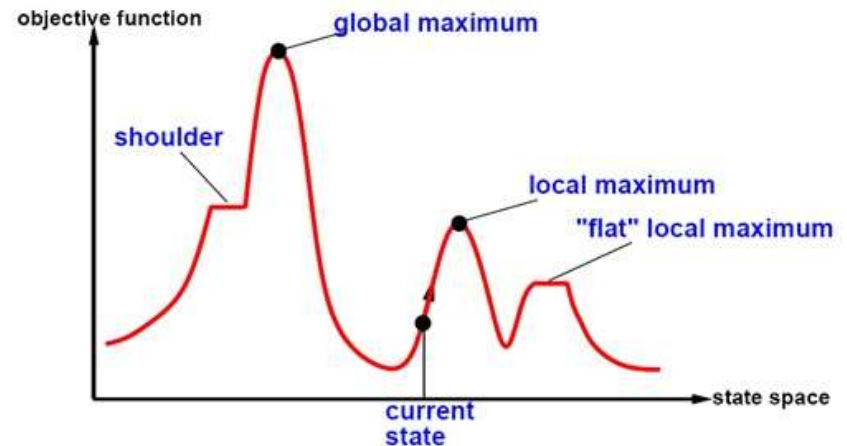
- instead of picking the best moves, it picks a random move.

*Choose neighbour based on*

- Objective function
- Probability
- Better objective functions have higher probability
- **If move improves the situation, it is always accepted, otherwise, it is accepted with probability less than 1.**
- **Probability decreases exponentially with 'badness' of the move**
- **Probability also decreases with  $T$  (higher probability when  $T$  is high and less when  $T$  decreases)**
  - *$T$  decreases according to a schedule*

# Local Beam Search

- Start with  $k$  random initial states
- Generate successors of all  $k$  states
- If any one of the successor is a goal, halts
- Select the  $k$  best children and repeat.
- Useful information is passed among parallel search threads – may lead to lack of diversity among  $k$  states
- Use stochastic beam search (randomly choose  $k$  successors instead of the best)



# Offline vs Online

- Offline
  - Compute solution before setting foot in the real world.
- Online
  - Interleaves computation and action.
  - Takes action, observes, computes, takes next action.
  - Allows agent to focus on contingencies that actually arise rather than those that might arise but probably won't.
  - Useful for dynamic environment or unknown environment



# Adversarial Search

- **Multiagent**
  - Collaborative or competitive
- **Games**
  - Purely rule based / known moves e.g. Chess
  - Stochastic e.g. Backgammon (with dice)
  - Stochastic & Partial observability e.g. Poker



# Game search

---

- **Normal search, optimal solution is a sequence of actions leading to a goal state.**
- **In Adversarial (game) search, the opponent has a say, so contingency plan is required**
- **Rational opponent – maximizes its own utility (payoff) function**

# Game Search Problem Formulation

---

- **Problem formulation**
  - Initial state: initial board position + whose move it is
  - Operators: legal moves a player can make
  - Goal (terminal test): game over?
  - Utility (payoff) function: measures the outcome of the game and its desirability
- **Search objective:**
  - Find the sequence of player's decisions (moves) maximizing its utility (payoff)
  - Consider the opponent's moves and their utility

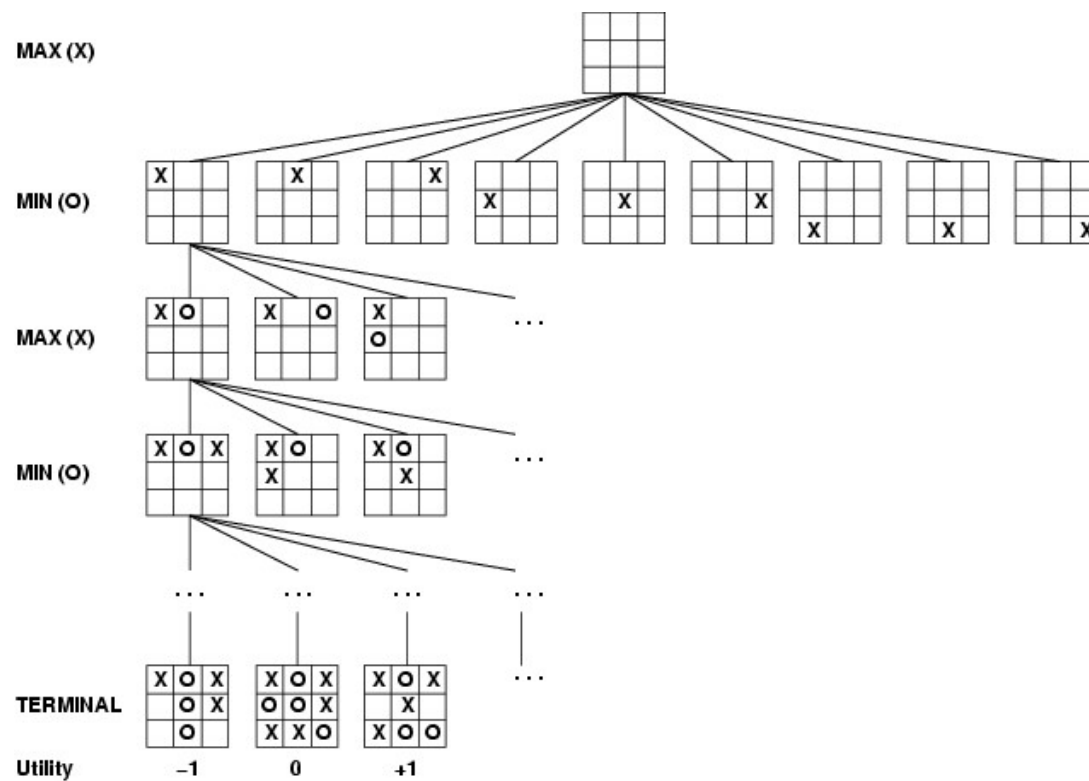


# Minimax Algorithm

---

- **How to deal with the contingency problem?**
  - Assuming the opponent is always rational and always optimizes its behavior (opposite to us), we consider the best opponent's response
  - Then the minimax algorithm determines the best move

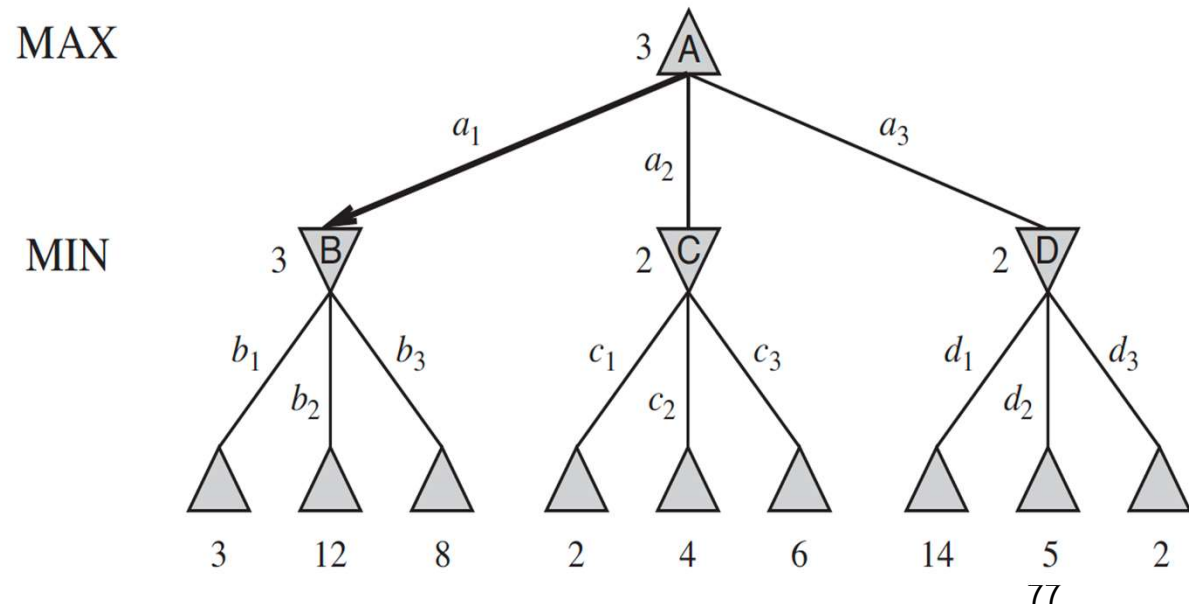
# Game tree (2-player, deterministic, turns)



# Minimax (2-Ply Game)

$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if } \text{TERMINAL-TEST}(s) \\ \max_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MIN} \end{cases}$$

What is the best move for MAX at A?  
What is the best move for MIN at B?



# Summary

## Topics

Agents

AI & problem solving

Search problems

Uninformed search

Informed search

- Greedy Best-First
- A\*

Local search

- Hill climbing search
- Simulated annealing
- Local beam search

Constraint satisfaction problems

Knowledge-based agents

Session 8