

# Search

Dra. M<sup>a</sup> Dolores Rodríguez Moreno

# Objectives

## Specific Objectives

- To understand the role of the search in AI
- Main search algorithms

## Source

- Stuart Russell & Peter Norvig (2009). Artificial Intelligence: A Modern Approach. (3rd Edition). Ed. Pearsons.

# Outline

- **Introduction**
- Problem formulation
- Problem types
- Basic search algorithms
- Conclusions

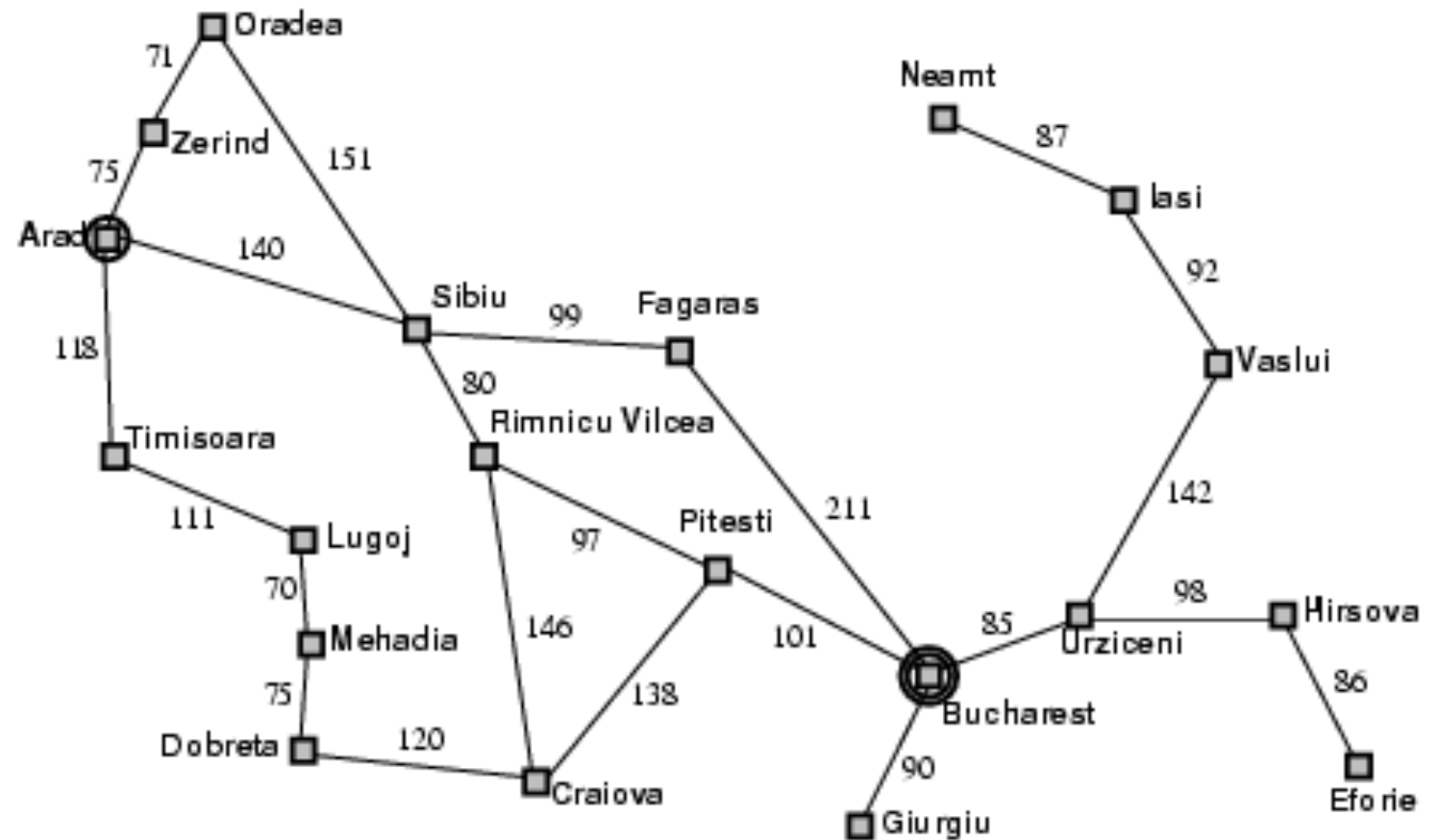
# Introduction

- Early AI works were directed to:
  - Proof of theorems
  - Solving crosswords
  - Games
- All in AI is search
  - Not entirely true (obviously) but more than we can imagine
  - Finding a good/best solution to a problem among several possible solutions

# Outline

- Introduction
- **Problem formulation**
- Problem types
- Basic search algorithms
- Conclusions

# Problem formulation



# Problem formulation (I)

- Agent must maximize its performance measure
- Example: On holiday in Romania; currently in Arad  
Flight leaves tomorrow from Bucharest
- Formulate goal:
  - be in Bucharest
- Formulating the problem:
  - states: multiple cities
  - actions: drive between cities
- Finding a solution:
  - Sequence cities, eg., Arad, Sibiu, Fagaras, Bucharest
- Process of finding such a solution: **Search**

## Problem formulation (II)

- Assumptions of the environment:
  - Static: search and formulation is done without considering changes in the environment
  - Observable: the initial state is known
  - Discrete: the alternative locations are known
  - Deterministic: each state is determined by the current state and the action executed
- The solutions are simple sequences of actions, they are executed without considering perceptions



# Problem formulation (III)

- A problem is defined by four items:
  1. initial state e.g., "at Arad"
  2. actions or successor function  $S(x)$  = set of action–state pairs  
e.g.,  $S(\text{Arad}) = \{ \langle \text{Arad} \rightarrow \text{Zerind}, \text{Zerind} \rangle, \dots \}$
  3. goal test, can be
    - explicit, e.g.,  $x = \text{"at Bucharest"}$
    - implicit, e.g.,  $\text{Checkmate}(x)$
  4. path cost (additive)
    - e.g., sum of distances, number of actions executed, etc.
    - $c(x,a,y)$  is the step cost, assumed to be  $\geq 0$
- A solution is a sequence of actions leading from the initial state to a goal state

## Problem formulation (IV)

- Real world is absurdly complex
  - state space must be abstracted for problem solving
- (Abstract) state = set of real states
- (Abstract) action = complex combination of real actions
  - e.g., "Arad → Zerind" represents a complex set of possible routes, detours, rest stops, etc.
- For guaranteed realizability, any real state "in Arad" must get to some real state "in Zerind"
- (Abstract) solution =
  - set of real paths that are solutions in the real world
- Each abstract action should be "easier" than the original problem

# Outline

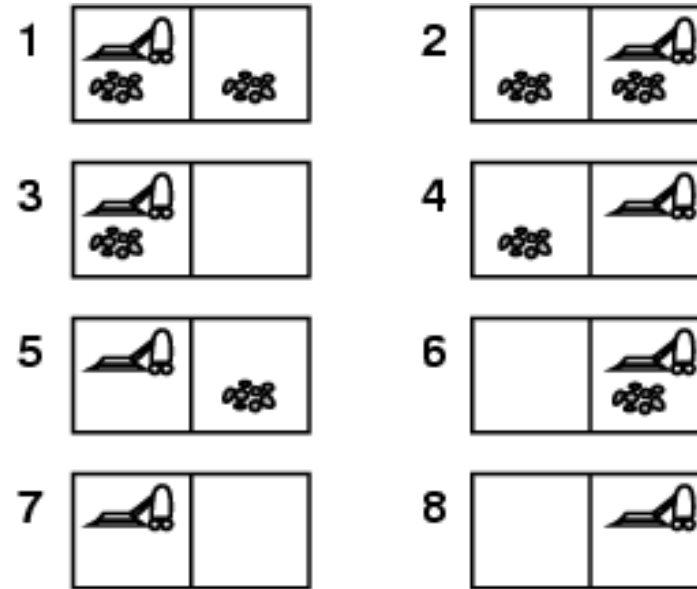
- Introduction
- Problem formulation
- **Problem types**
- Basic search algorithms
- Conclusions

# Problem types

- Deterministic, fully observable → **single-state problem**
  - Agent knows exactly which state it will be in; solution is a sequence
- Non-observable → **sensorless problem** (conformant problem)
  - Agent may have no idea where it is; solution is a sequence
- Nondeterministic and/or partially observable → **contingency problem**
  - percepts provide new information about current state
  - often interleave search with execution
- Unknown state space → **exploration problem**

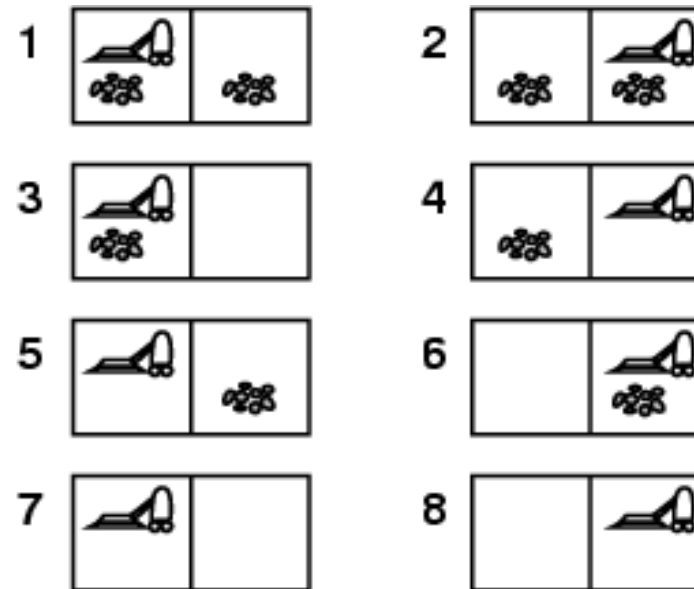
# Problem types: example

- Single-state, start in #5.  
**Solution?**



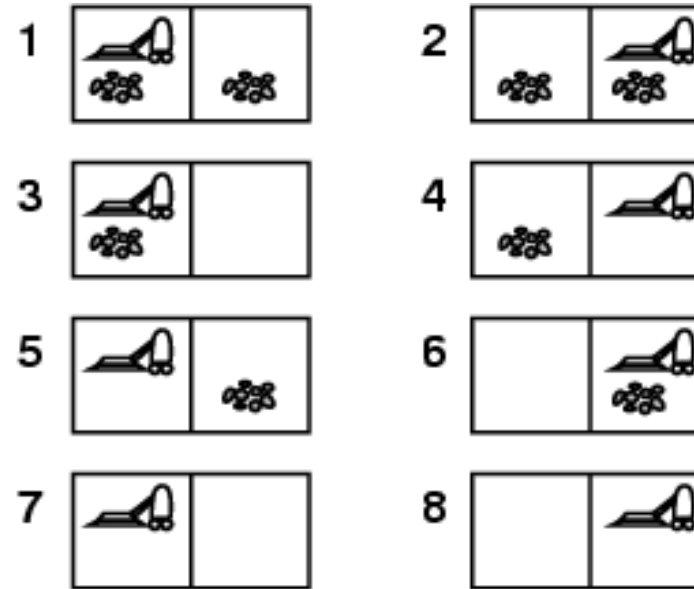
## Problem types: example (I)

- Single-state, start in #5.  
**Solution?** [*Right, Suck*]



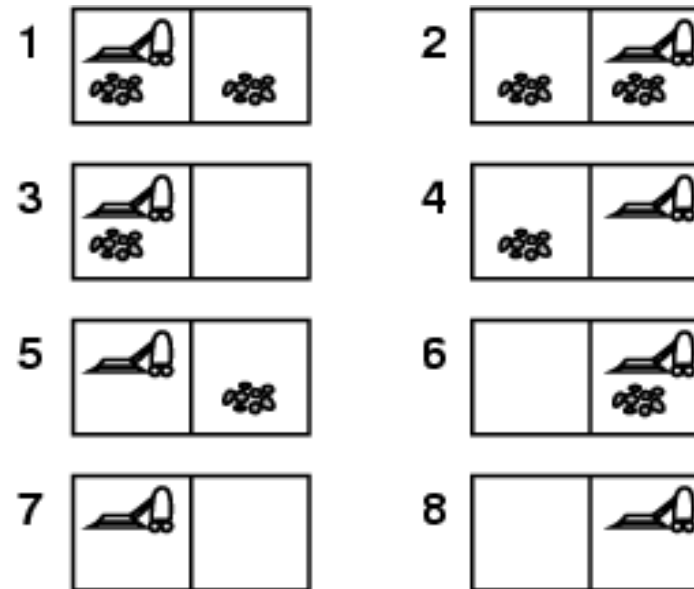
## Problem types: example (II)

- Single-state, start in #5.  
**Solution?** *[Right, Suck]*
- Sensorless, start in  $\{1,2,3,4,5,6,7,8\}$  e.g.,  
Right goes to  $\{2,4,6,8\}$   
**Solution?**



## Problem types: example (III)

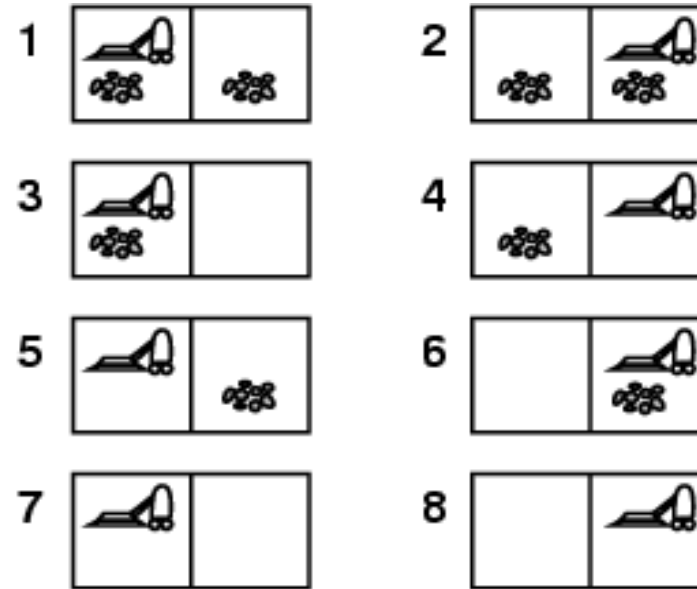
- Single-state, start in #5.  
**Solution?** *[Right, Suck]*
- Sensorless, start in {1,2,3,4,5,6,7,8} e.g.,  
Right goes to {2,4,6,8}  
**Solution?** *[Right, Suck, Left, Suck]*





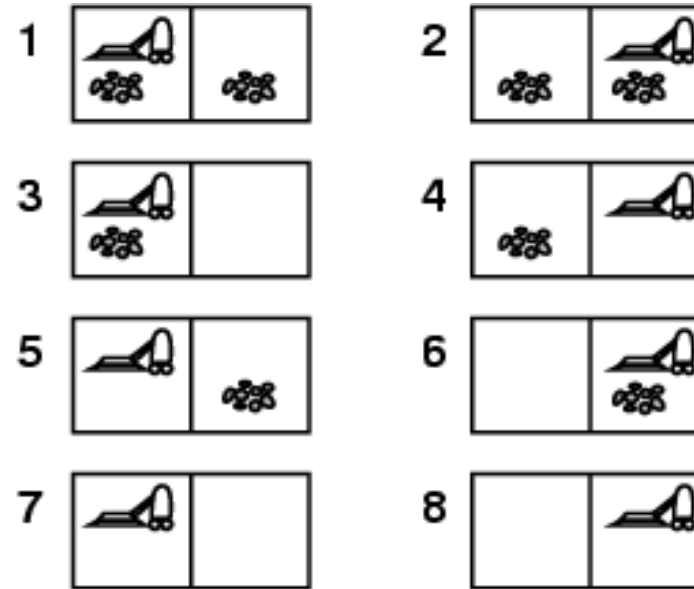
## Problem types: example (IV)

- Single-state, start in #5.  
**Solution?** *[Right, Suck]*
- Sensorless, start in {1,2,3,4,5,6,7,8} e.g.,  
Right goes to {2,4,6,8}  
**Solution?** *[Right, Suck, Left, Suck]*
- Contingency
  - Nondeterministic: Suck may dirt the carpet
  - Partially observable: location, dirt at current location
  - Percept: [L, Clean], i.e., start in #5 or #7  
**Solution?**



## Problem types: example (V)

- Single-state, start in #5.  
**Solution?** *[Right, Suck]*
- Sensorless, start in {1,2,3,4,5,6,7,8} e.g.,  
Right goes to {2,4,6,8}  
**Solution?** *[Right, Suck, Left, Suck]*
- Contingency
  - Nondeterministic: Suck may dirt the carpet
  - Partially observable: location, dirt at current location
  - Percept: [L, Clean], i.e., start in #5 or #7  
**Solution?** *[Right, **if** dirt **then** Suck]*



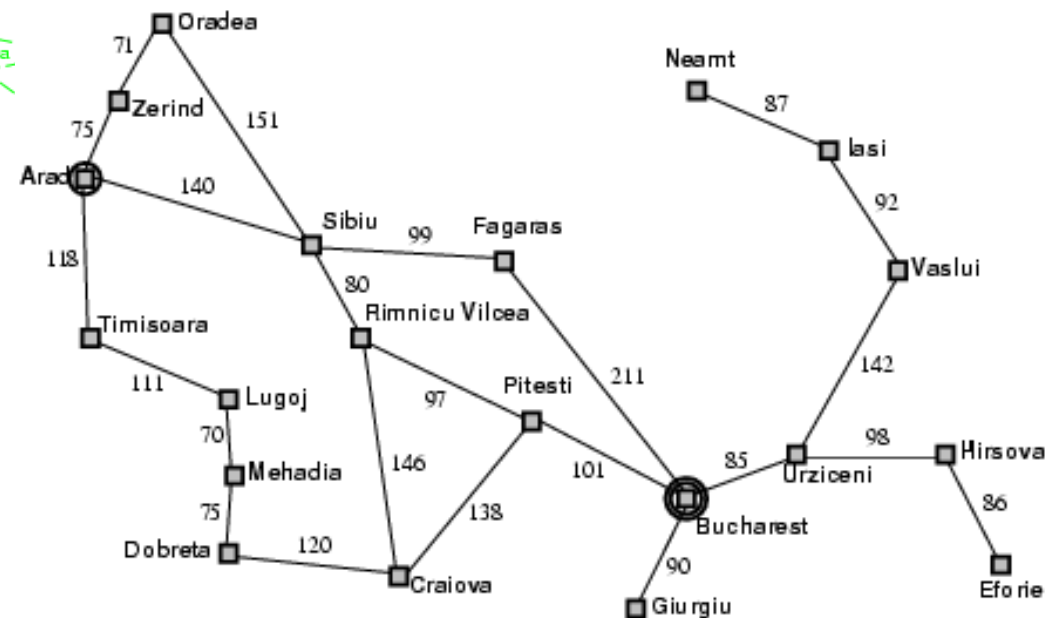
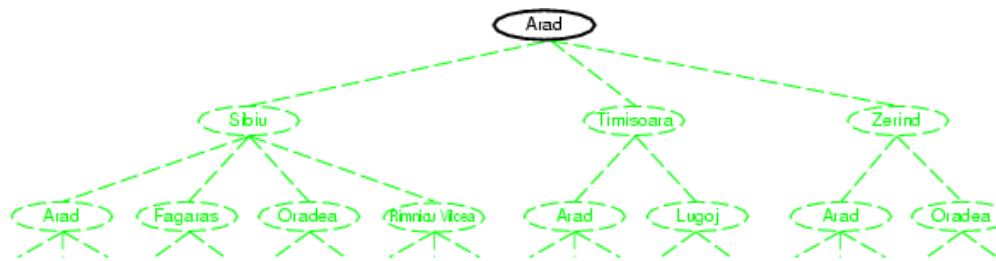
# Outline

- Introduction
- Problem formulation
- Problem types
- **Search algorithms**
- Conclusions

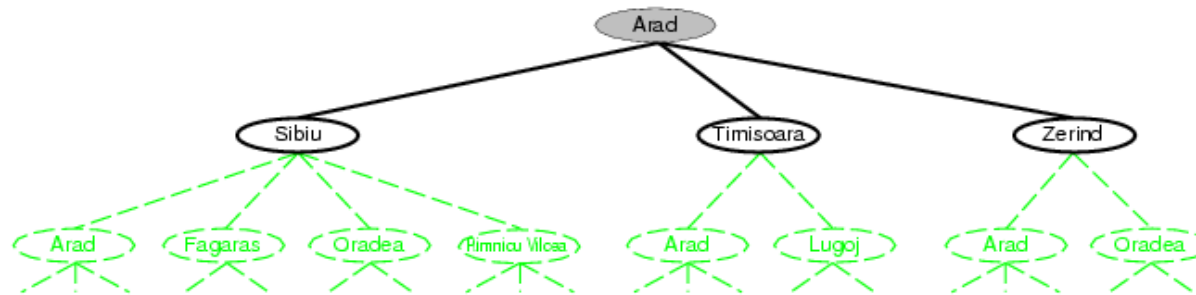
# Search algorithms

- We have formulated problems, we now need to solve them: search tree
- In general we can have a search graph rather than a tree when the state can be reached from multiple paths
- Basic idea:
  - offline, simulated exploration of state space by generating successors of already-explored states (i.e. **expanding** states)

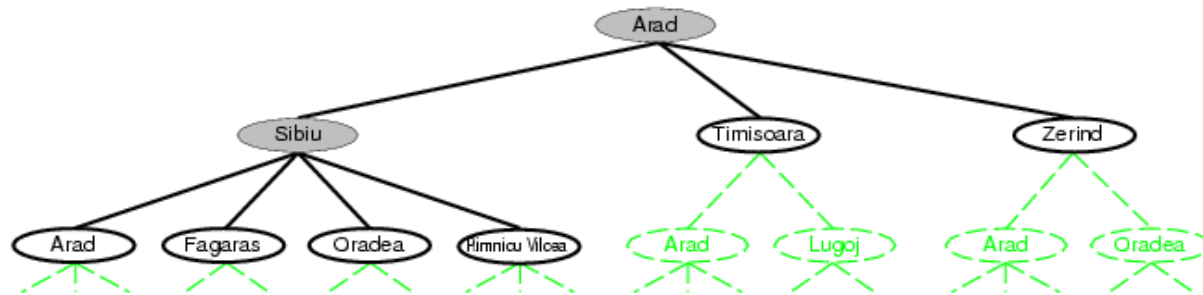
# Search algorithms: tree search example



# Search algorithms: tree search example

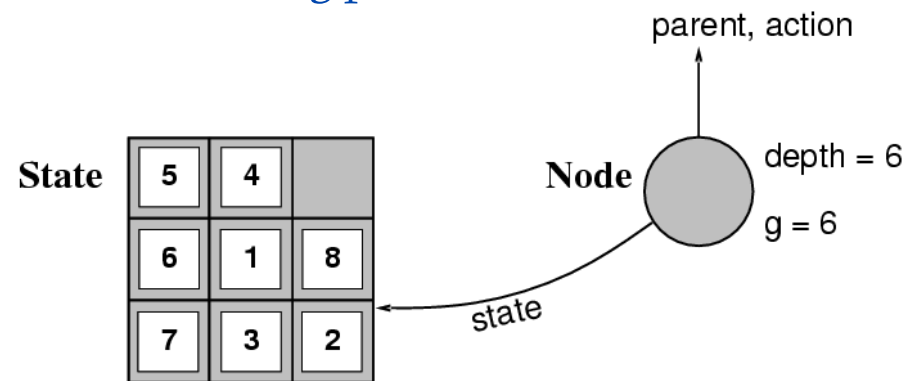


# Search algorithms: tree search example



# Search algorithms: states vs. nodes

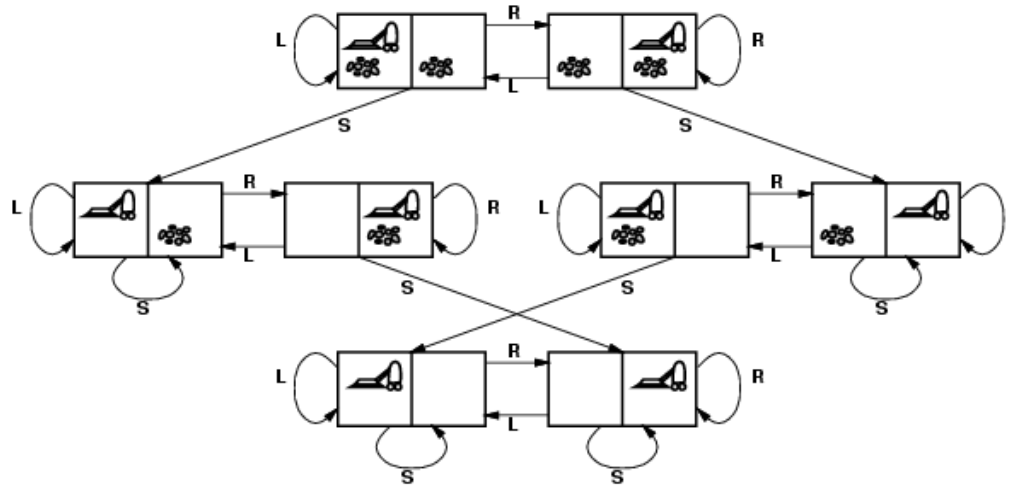
- A **state** is a (representation of) a physical configuration
- A **node** is a data structure constituting part of a search tree includes **state**, **parent node**, **action**, **path cost**  $g(x)$ , **depth**



- The Expand function to create the corresponding states

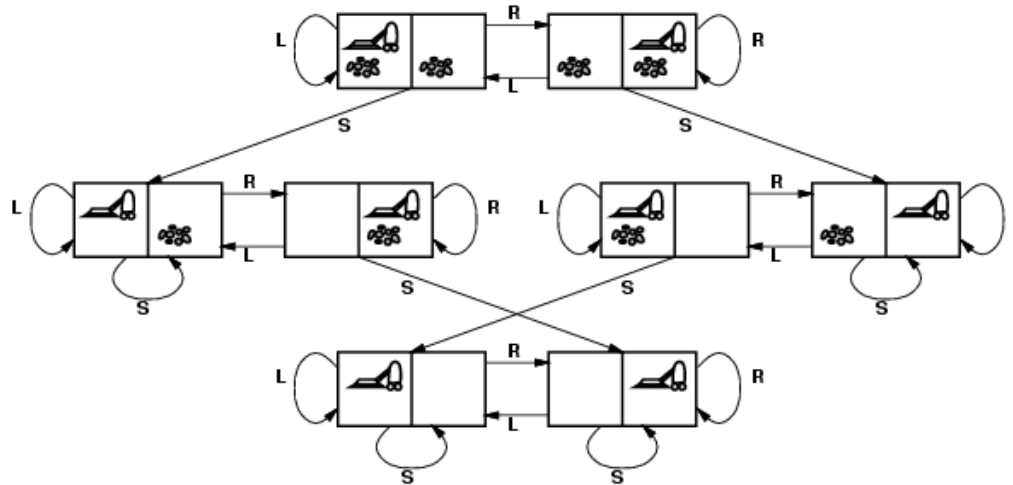


# Search algorithms: Vacuum world (I)



- states?
- actions?
- goal test?
- path cost?

## Search algorithms: Vacuum world (II)



- states? integer dirt and robot location
- actions? *Left, Right, Suck*
- goal test? no dirt at all locations
- path cost? 1 per action

# Search algorithms: The 8-puzzle (I)

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

- states?
- actions?
- goal test?
- path cost?

## Search algorithms: The 8-puzzle (II)

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

- states? locations of tiles
- actions? move blank left, right, up, down
- goal test? = goal state (given)
- path cost? 1 per move

[Note: optimal solution of  $n$ -Puzzle family is NP-hard]

# Search strategies

- A search 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: number of nodes generated
  - space complexity: maximum number of nodes in memory
  - optimality: does it always find a least-cost solution?
- Time and space complexity are measured in terms of
  - b: maximum branching factor of the search tree
  - d: depth of the least-cost solution
  - m: maximum depth of the state space (may be  $\infty$ )

# Outline

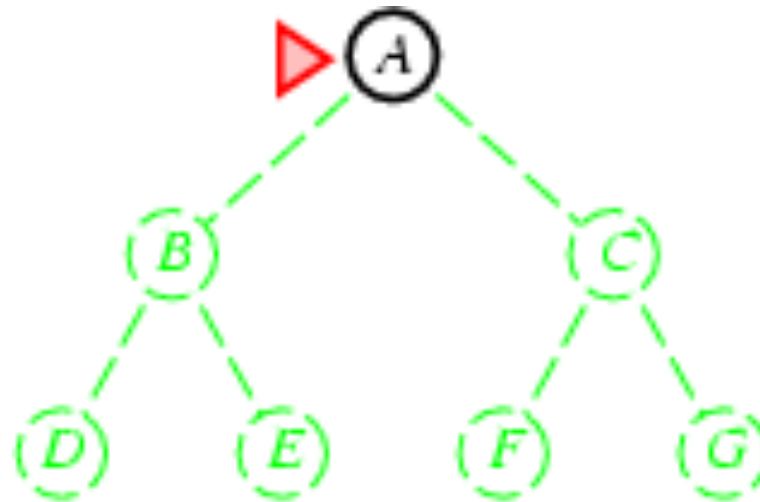
- Introduction
- Problem formulation
- Problem types
- Basic search algorithms
  - **Uninformed search**
  - Informed search
- Conclusions

# Introduction

- Uninformed search strategies use only the information available in the problem definition
  - Breadth-first search/ Búsqueda en anchura
  - Uniform-cost search/ Búsqueda de coste uniforme
  - Depth-first search/ Búsqueda en profundidad
  - Depth-limited search/ Búsqueda en profundidad limitada
  - Iterative deepening search/Búsqueda de profundización iterativa

# Breadth-first search

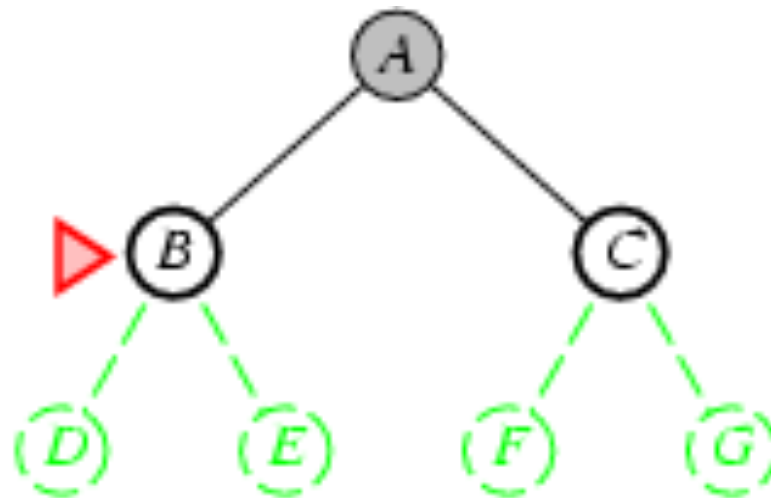
- Expand shallowest unexpanded node
- Implementation:
  - *fringe* is a FIFO queue, i.e., new successors go at end





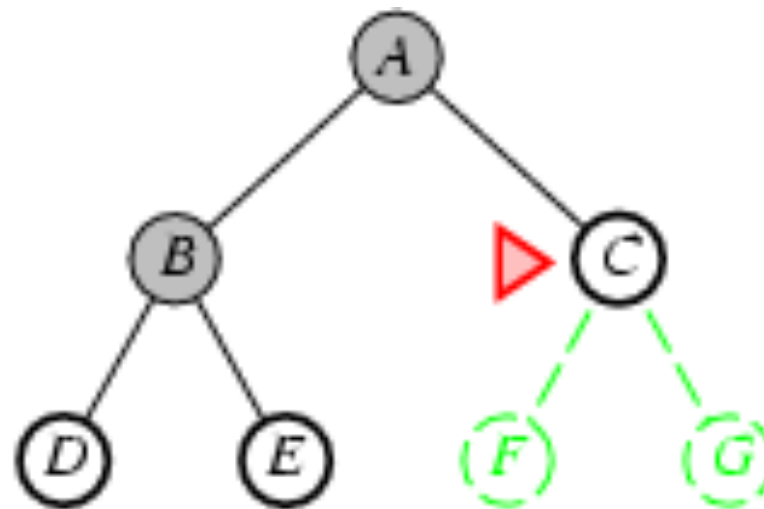
# Breadth-first search

- Expand shallowest unexpanded node
- Implementation:
  - *fringe* is a FIFO queue, i.e., new successors go at end



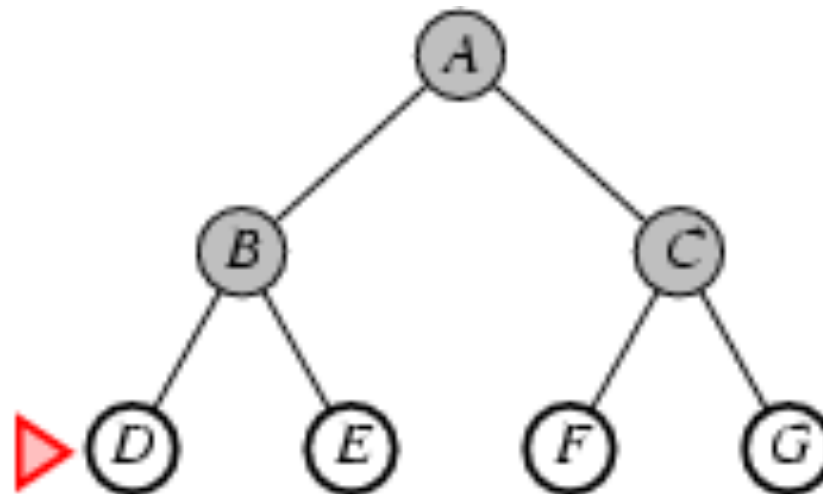
# Breadth-first search

- Expand shallowest unexpanded node
- Implementation:
  - *fringe* is a FIFO queue, i.e., new successors go at end



# Breadth-first search

- Expand shallowest unexpanded node
- Implementation:
  - *fringe* is a FIFO queue, i.e., new successors go at end



# Properties of breadth-first search

- Complete? Yes (if  $b$  is finite)
- Time?  $1 + b + b^2 + b^3 + \dots + b^d + b(b^d - 1) = O(b^{d+1})$
- Space?  $O(b^{d+1})$  (keeps every node in memory)
- Optimal? Yes (if cost = 1 per step)

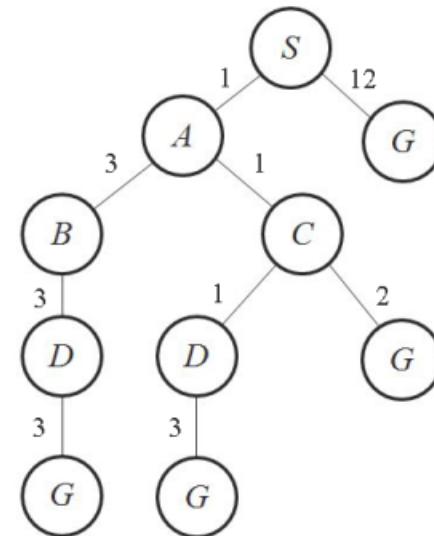
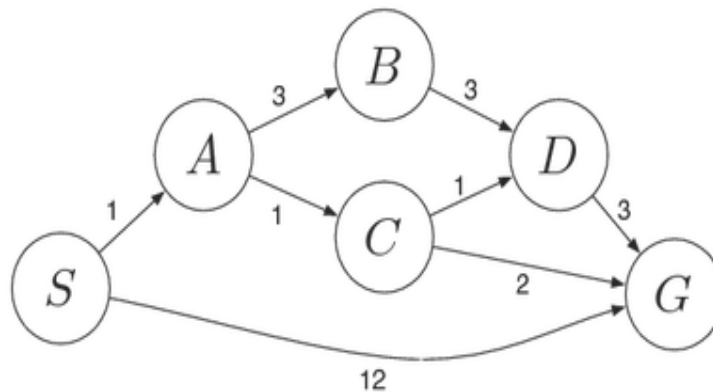
**Space** is the bigger problem (more than time)

Each state has  $b$  successors (branching factor)

$d$  is the shallower depth

# Uniform-cost search

- Expand least-cost unexpanded node
- **Implementation:**
  - *fringe* = queue ordered by path cost
- Find the solution with minimum cumulative cost, i.e. an optimal solution



# Uniform-cost search (Solution)

Initialization: { [ S , 0 ] }

Iteration1: { [ S->A , 1 ] , [ S->G , 12 ] }

Iteration2: { [ S->A->C , 2 ] , [ S->A->B , 4 ] , [ S->G , 12 ] }

Iteration3: { [ S->A->C->D , 3 ] , [ S->A->C->G , 4 ] , [ S->A->B->D , 7 ] , [ S->G , 12 ] }

Iteration 4: { [ S->A->C->D->G , 6 ] , [ S->A->C->G , 4 ] , [ S->A->B->D , 7 ] , [ S->G , 12 ] }

Iteration 5: { [ S->A->C->G , 4 ] , [ S->A->C->D->G , 6 ] , [ S->A->B->D->G , 10 ] , [ S->G , 12 ] }

Solution: S->A->C->G.

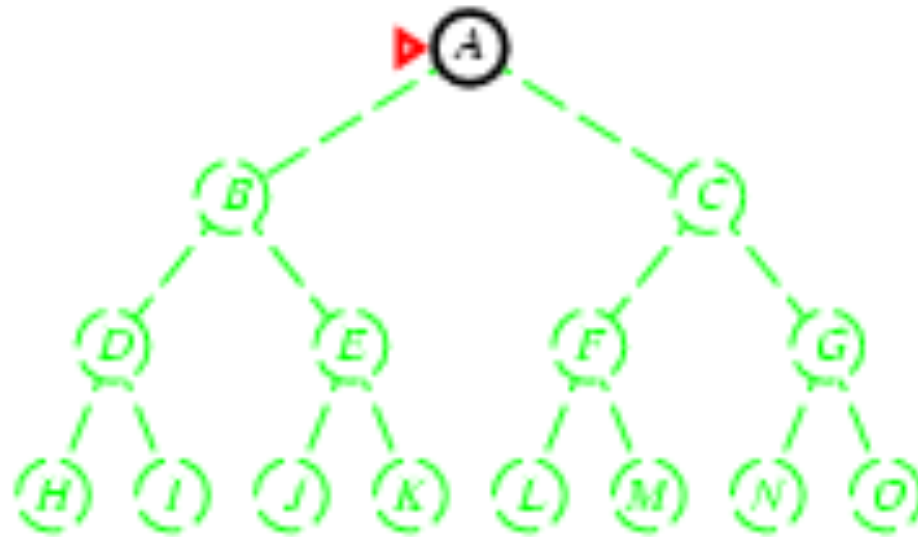
# Uniform-cost search

- Complete? Yes, if step cost  $\geq \epsilon$
- Time? # of nodes with  $g \leq$  cost of optimal solution,  $O(b^{\text{ceiling}(C^*/\epsilon)})$  where  $C^*$  is the cost of the optimal solution
- Space? # of nodes with  $g \leq$  cost of optimal solution,  $O(b^{\text{ceiling}(C^*/\epsilon)})$
- Optimal? Yes – nodes expanded in increasing order of  $g(n)$

*If all costs are equal  $\rightarrow O(b^d)$*

# Depth-first search

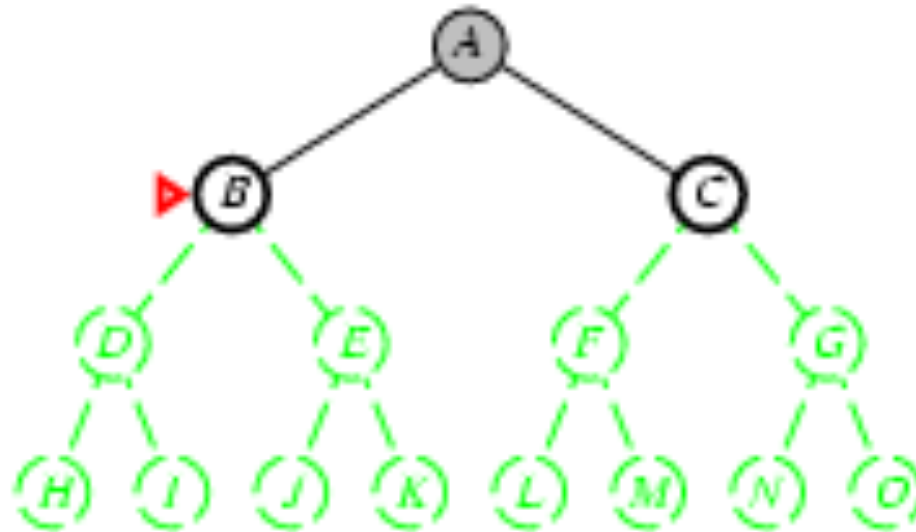
- Expand deepest unexpanded node
- **Implementation:**
  - *fringe* = LIFO queue, i.e., put successors at front





# Depth-first search

- Expand deepest unexpanded node
- **Implementation:**
  - *fringe* = LIFO queue, i.e., put successors at front



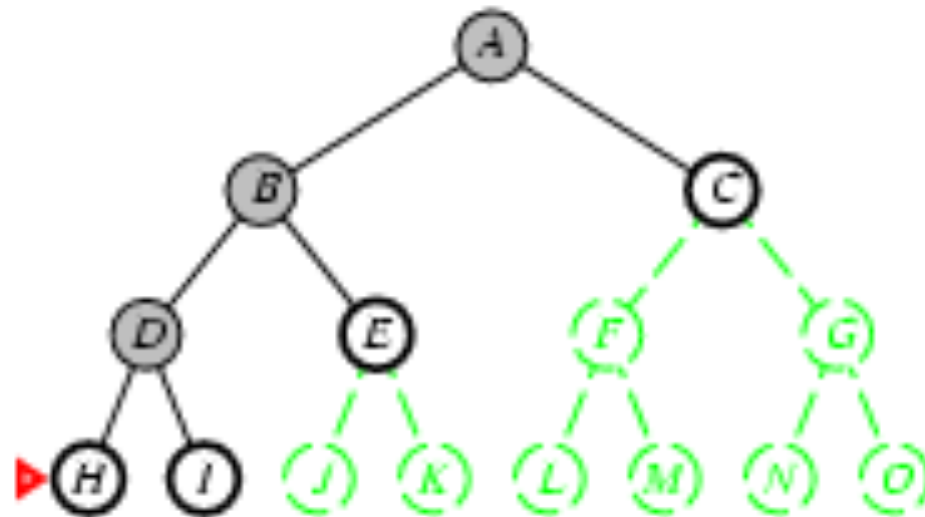
# Depth-first search

- Expand deepest unexpanded node
- **Implementation:**
  - *fringe* = LIFO queue, i.e., put successors at front



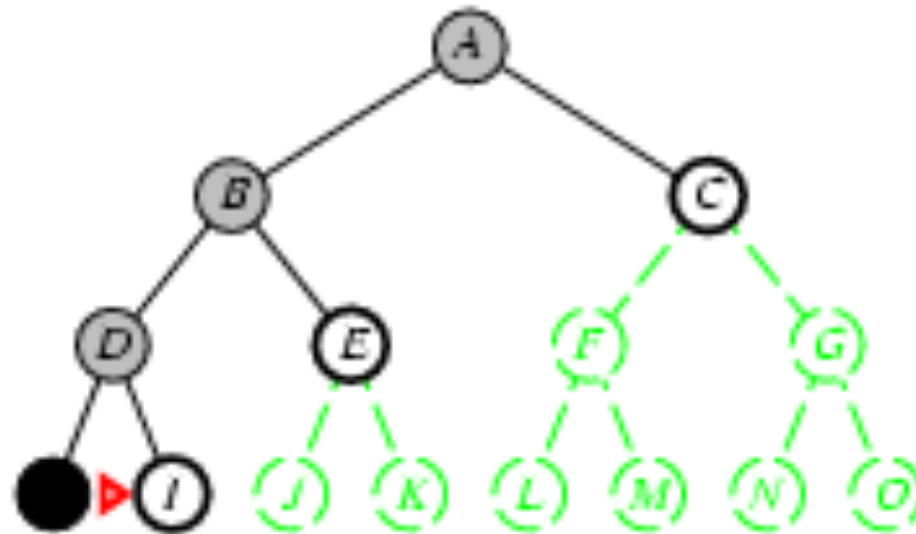
# Depth-first search

- Expand deepest unexpanded node
- **Implementation:**
  - *fringe* = LIFO queue, i.e., put successors at front



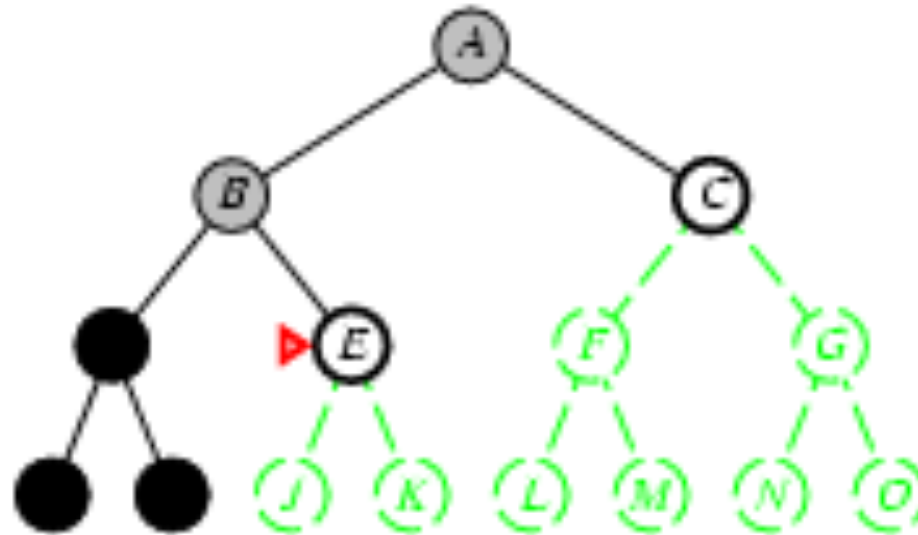
# Depth-first search

- Expand deepest unexpanded node
- **Implementation:**
  - *fringe* = LIFO queue, i.e., put successors at front



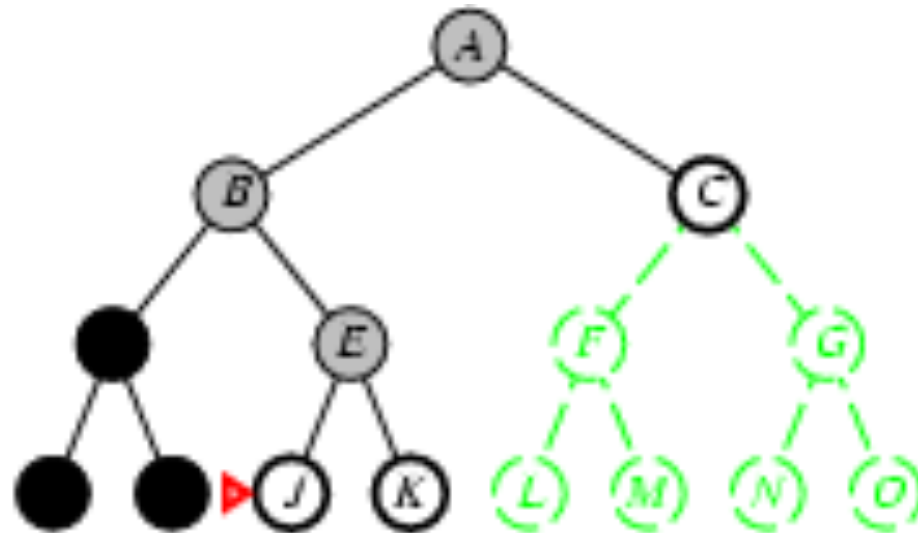
# Depth-first search

- Expand deepest unexpanded node
- **Implementation:**
  - *fringe* = LIFO queue, i.e., put successors at front



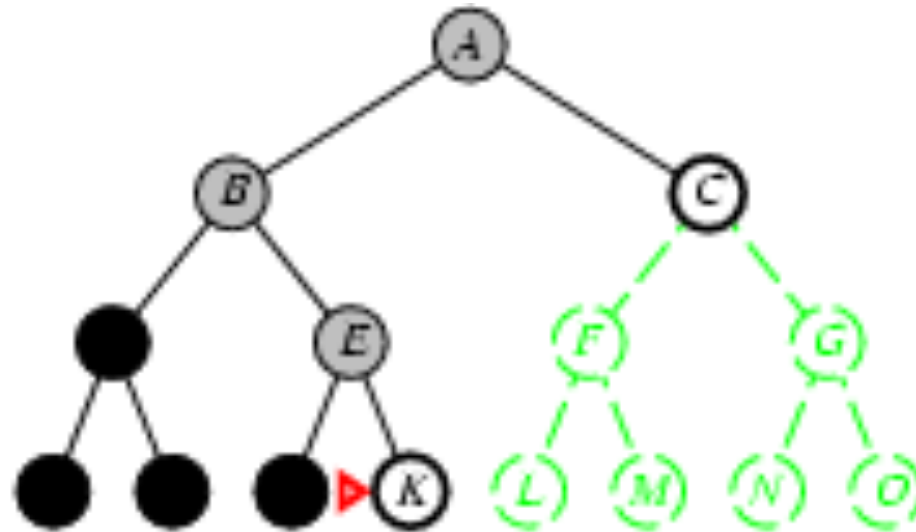
# Depth-first search

- Expand deepest unexpanded node
- **Implementation:**
  - *fringe* = LIFO queue, i.e., put successors at front



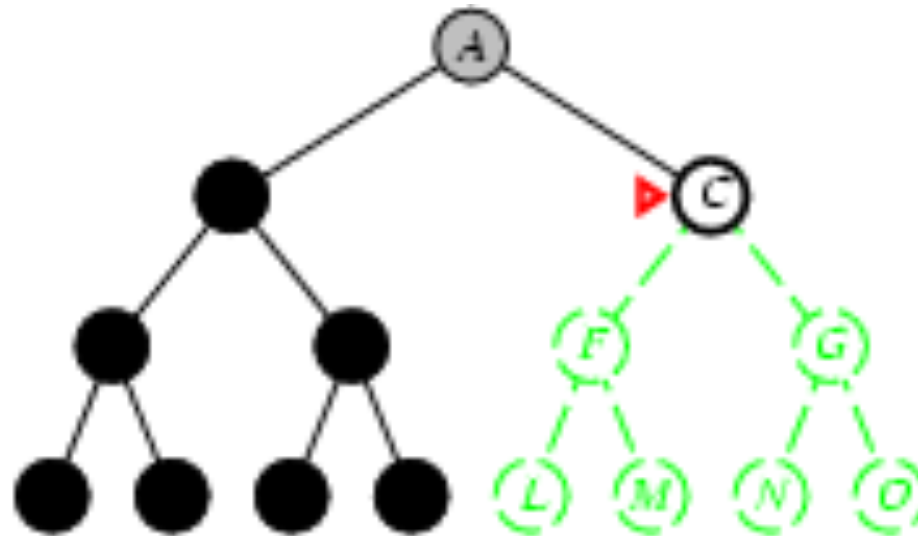
# Depth-first search

- Expand deepest unexpanded node
- **Implementation:**
  - *fringe* = LIFO queue, i.e., put successors at front



# Depth-first search

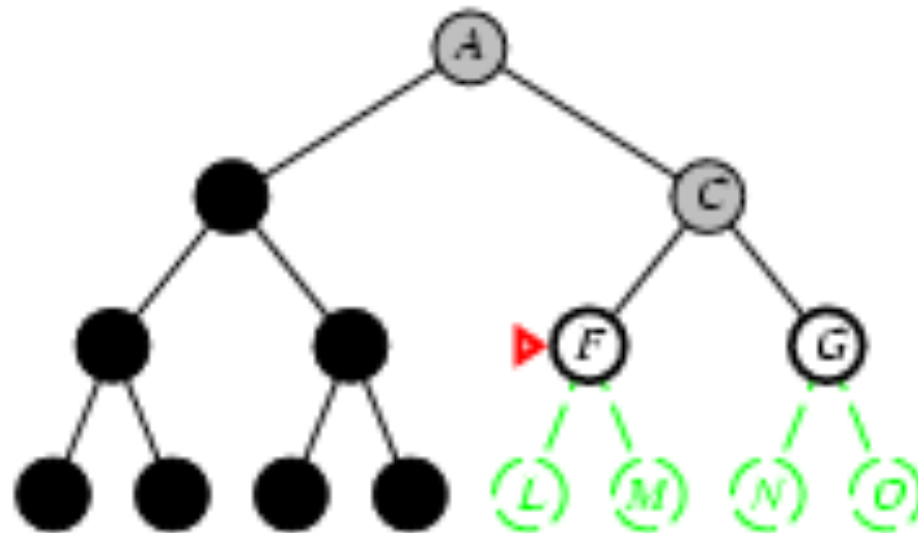
- Expand deepest unexpanded node
- **Implementation:**
  - *fringe* = LIFO queue, i.e., put successors at front





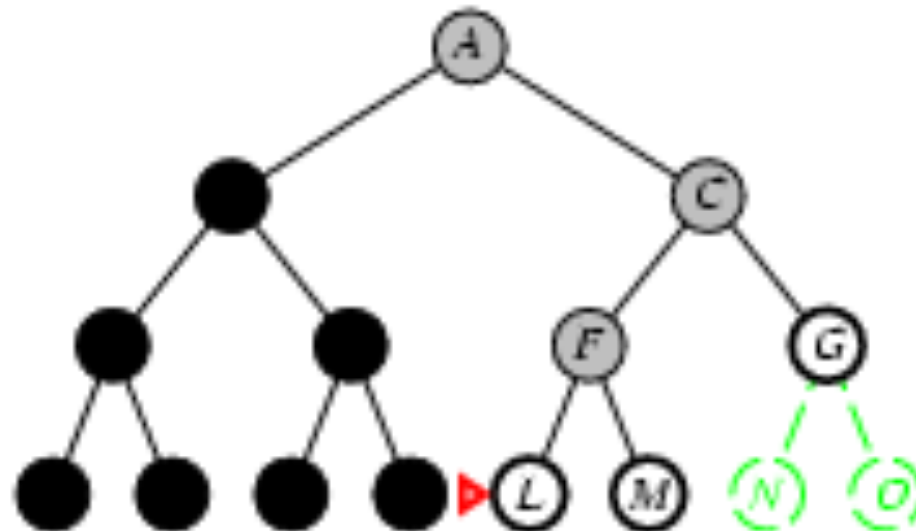
# Depth-first search

- Expand deepest unexpanded node
- **Implementation:**
  - *fringe* = LIFO queue, i.e., put successors at front



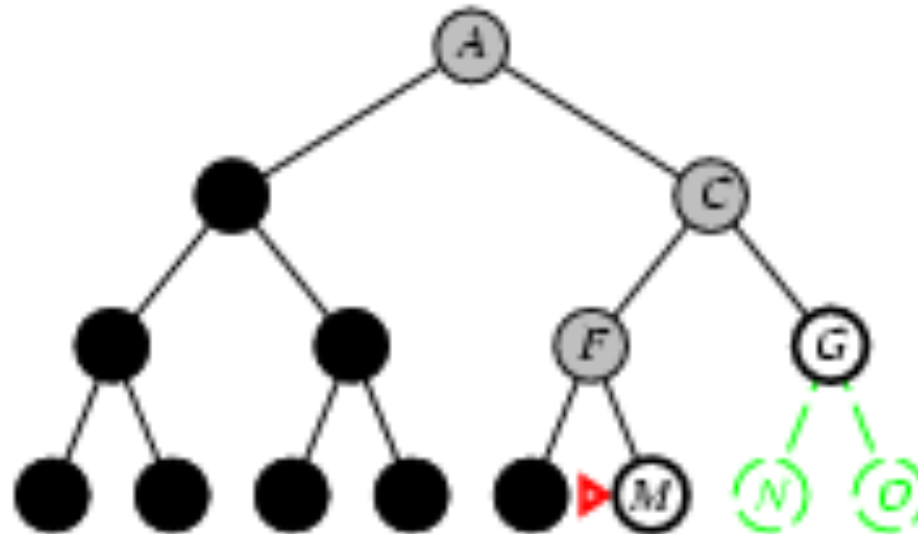
# Depth-first search

- Expand deepest unexpanded node
- **Implementation:**
  - *fringe* = LIFO queue, i.e., put successors at front



# Depth-first search

- Expand deepest unexpanded node
- **Implementation:**
  - *fringe* = LIFO queue, i.e., put successors at front



# Properties of depth-first search

- Complete? No: fails in infinite-depth spaces, spaces with loops
  - Modify to avoid repeated states along path
    - complete in finite spaces
- Time?  $O(b^m)$ : terrible if  $m$  is much larger than  $d$ 
  - but if solutions are dense, may be much faster than breadth-first
- Space?  $O(bm)$
- Optimal? No

# Depth-limited search

- = depth-first search with depth limit  $L$
- i.e., nodes at depth  $L$  have no successor

# Iterative deepening search $L=0$

Limit = 0

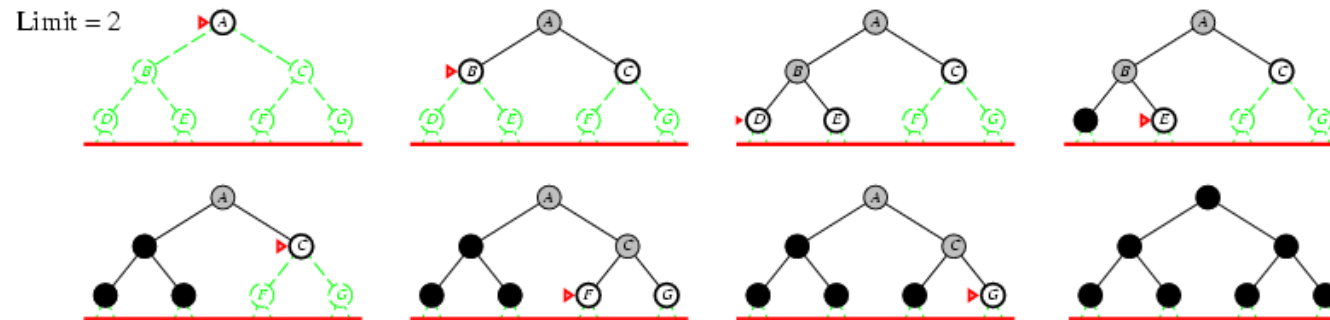


# Iterative deepening search $L=1$

Limit = 1



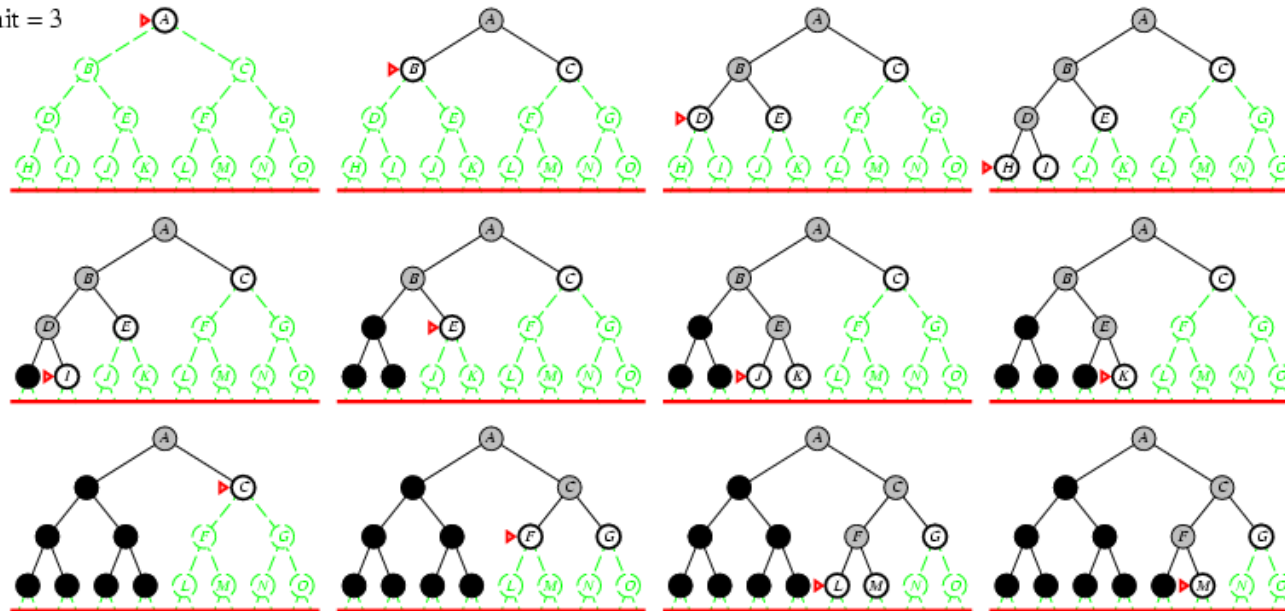
# Iterative deepening search $L=2$





# Iterative deepening search $L=3$

Limit = 3

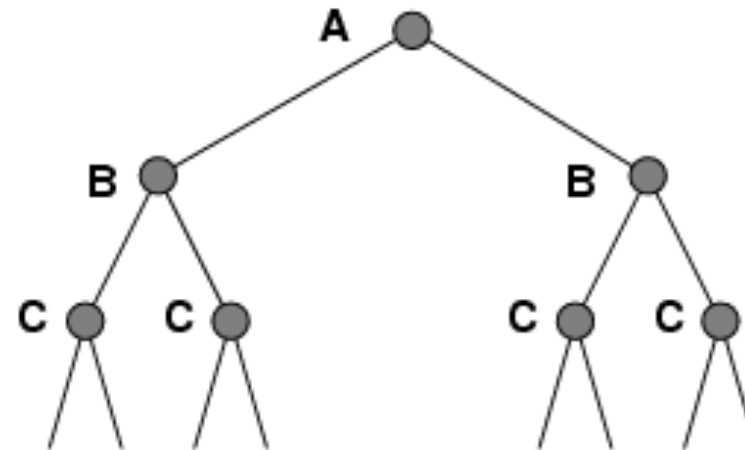
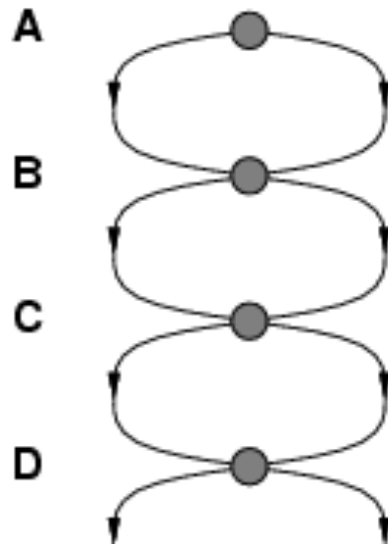


# Properties of iterative deepening search

- Complete? Yes
- Time?  $(d+1)b^0 + d b^1 + (d-1)b^2 + \dots + b^d = O(b^d)$
- Space?  $O(bd)$
- Optimal? Yes, if step cost = 1

# Repeated states

- Failure to detect repeated states can turn a linear problem into an exponential one!



## Summary of algorithms

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	Yes	No	No	Yes

# Outline

- Introduction
- Problem formulation
- Problem types
- Basic search algorithms
  - Uninformed search
  - **Informed search**
- Conclusions

# Informed search strategies

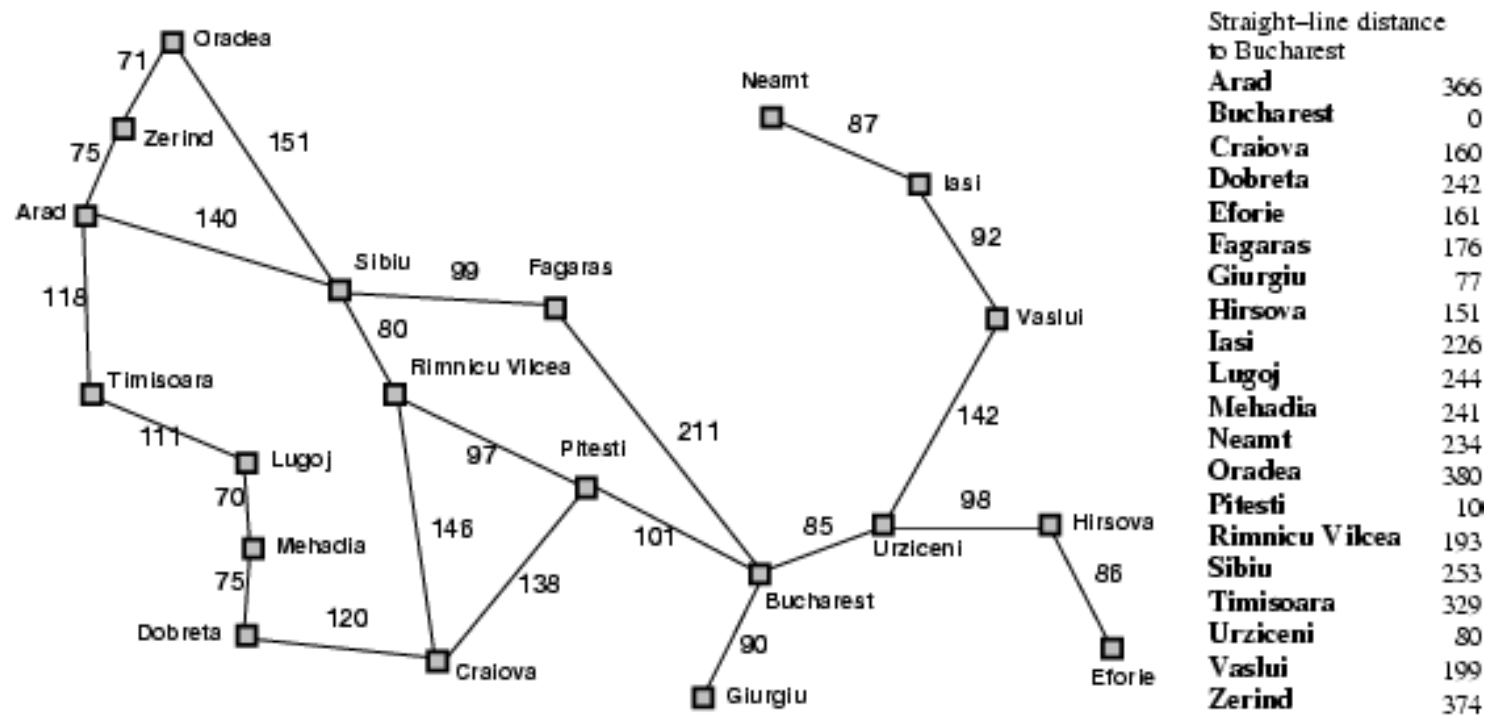
- Use the problem-specific knowledge beyond the definition of the problem itself to find more efficient solutions than uninformed strategies
  - Best-first search
    - Greedy best-first search/Búsqueda voraz primero el mejor
    - $A^*$
  - Heuristics
  - Local search algorithms
    - Hill-climbing search/Búsqueda de escalada
    - Simulated annealing search/Búsqueda de temple simulado
    - Local beam search/Búsqueda de haz local
    - Genetic algorithms/Algoritmos genéticos

# Best-first search

- Idea: use an **evaluation function**  $f(n)$  for each node
  - estimate of "desirability"
  - Expand most desirable unexpanded node
- Implementation:

Order the nodes in fringe in decreasing order of desirability
- Special cases:
  - greedy best-first search
  - A\* search

# Romania with step costs in km





# Informed search strategies

- Best-first search
  - Greedy best-first search
  - A\* search
- Heuristics
- Local search algorithms
  - Hill-climbing search
  - Simulated annealing search
  - Local beam search
  - Genetic algorithms

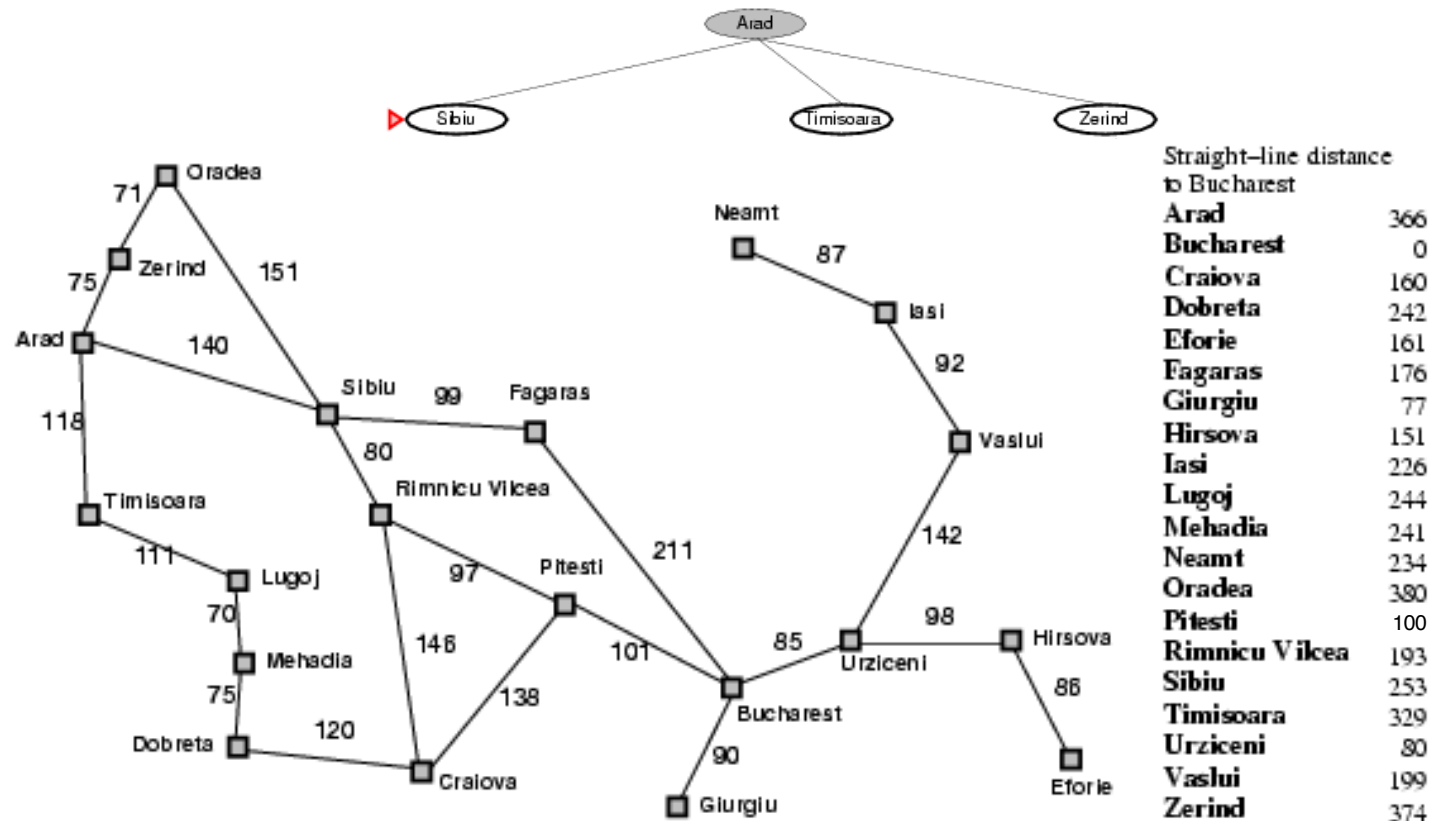
# Greedy best-first search

- Evaluation function  $f(n) = h(n)$  (**h**euristic) = estimate of cost from  $n$  to *goal*
- Greedy best-first search expands the node that **appears** to be closest to the goal
- Implementation: as a priority queue to keep the fringe in ascending order of  $f$ -values
- e.g.,  $h_{SLD}(n)$  = straight-line distance from  $n$  to Bucharest

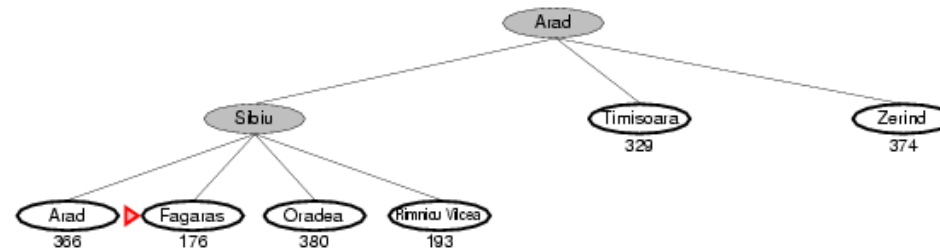
# Greedy best-first search example



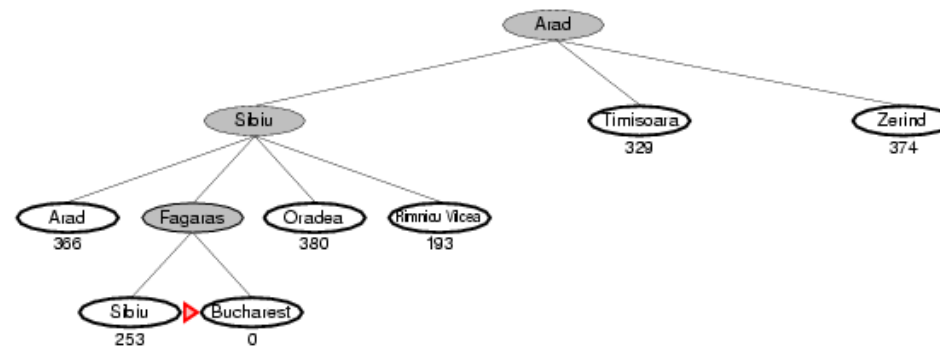
# Greedy best-first search example



# Greedy best-first search example



# Greedy best-first search example



# Properties of greedy best-first search

- Complete? No – can get stuck in loops, e.g., Iasi  $\rightarrow$  Neamt  $\rightarrow$  Iasi  $\rightarrow$  Neamt  $\rightarrow$
- Time?  $O(b^m)$ , but a good heuristic can give dramatic improvement
- Space?  $O(b^m)$  -- keeps all nodes in memory
- Optimal? No

Similar to depth-first search

Each state has  $b$  successors (branching factor)

$d$  is the depth of the shallowest solution

# Informed search strategies

- Best-first search
  - Greedy best-first search
  - A\* search
- Heuristics
- Local search algorithms
  - Hill-climbing search
  - Simulated annealing search
  - Local beam search
  - Genetic algorithms



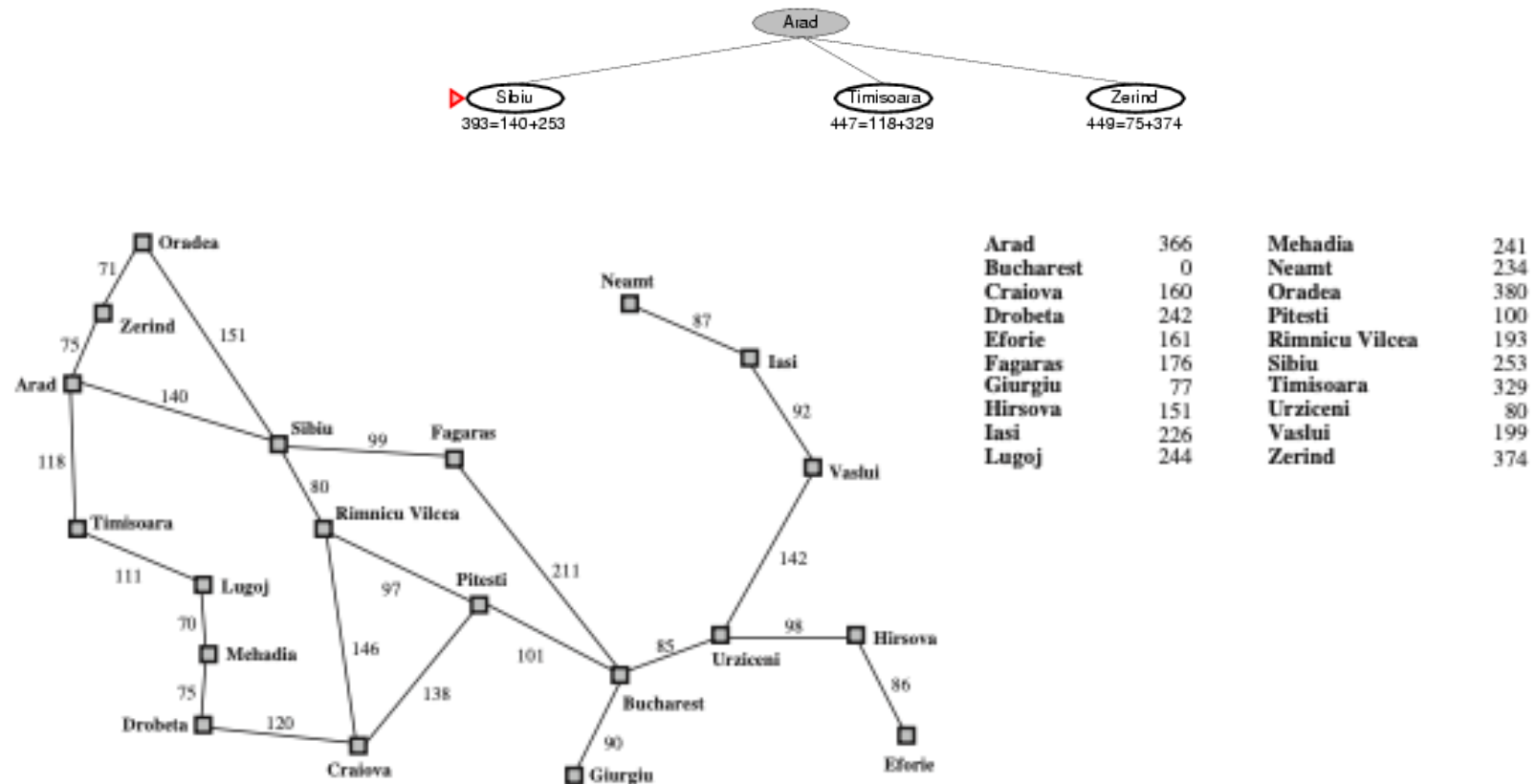
# A\* search

- Idea: avoid expanding paths that are already expensive
- Evaluation function  $f(n) = g(n) + h(n)$ 
  - $g(n)$  = 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
- A\* is optimal if  $h(n)$  is an admissible heuristic such that  $h(n)$  never overestimates the cost to reach the goal

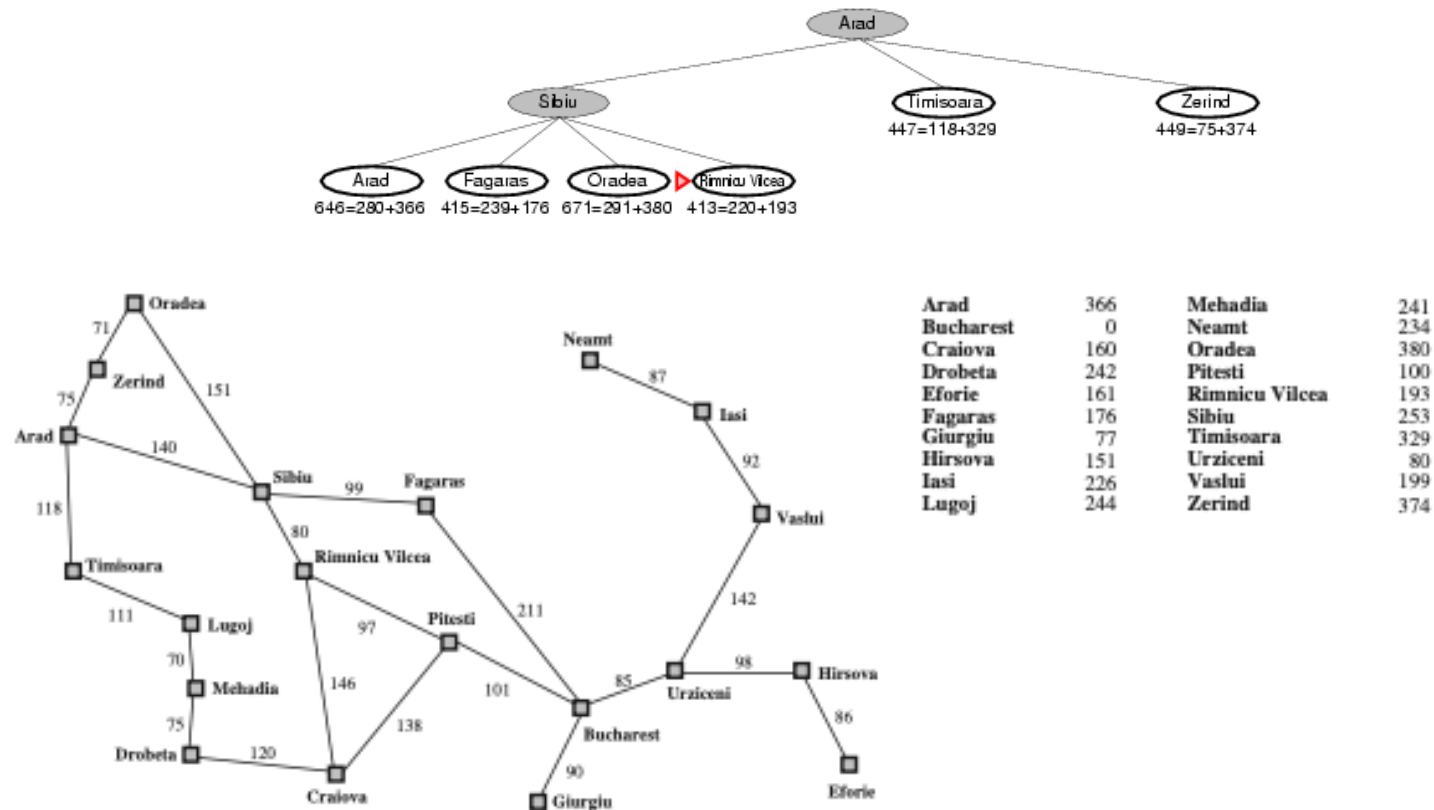
# A\* search example



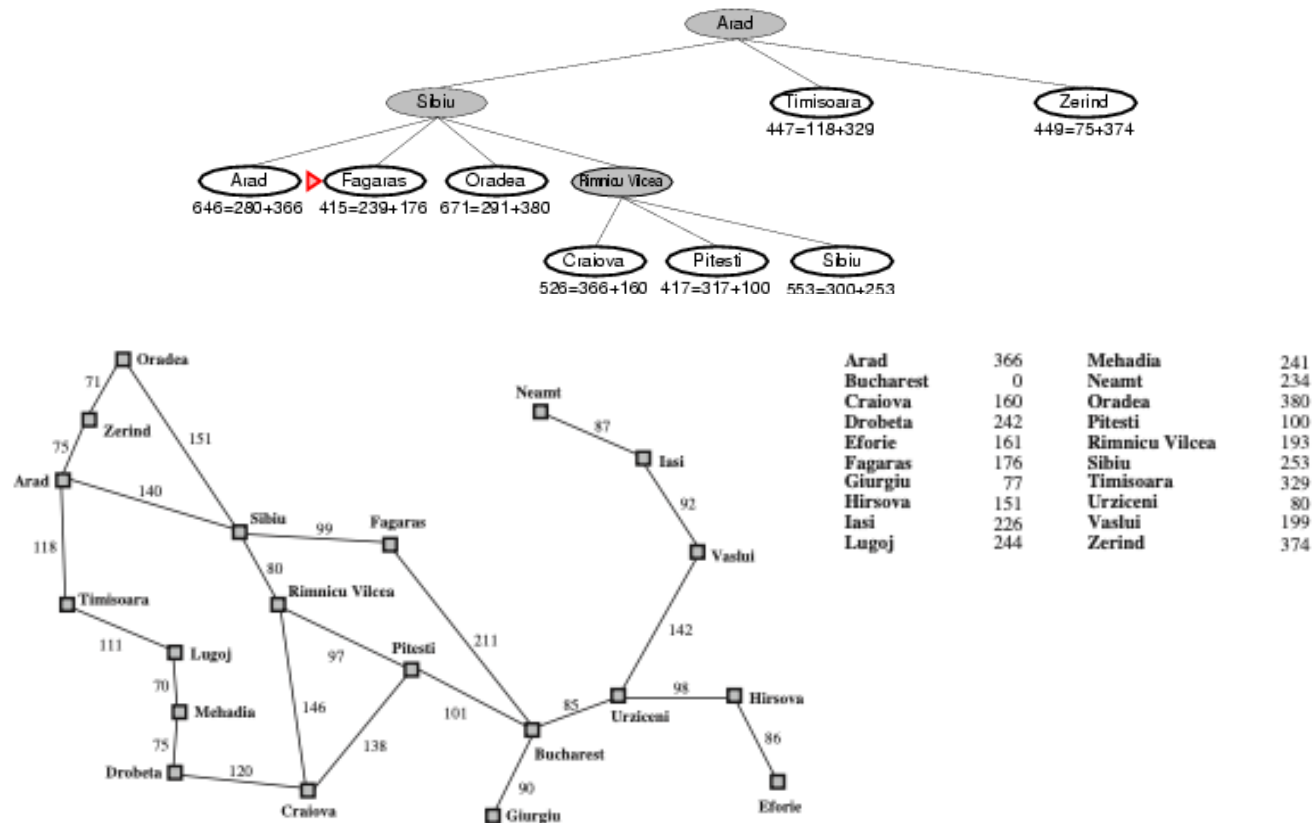
# A\* search example



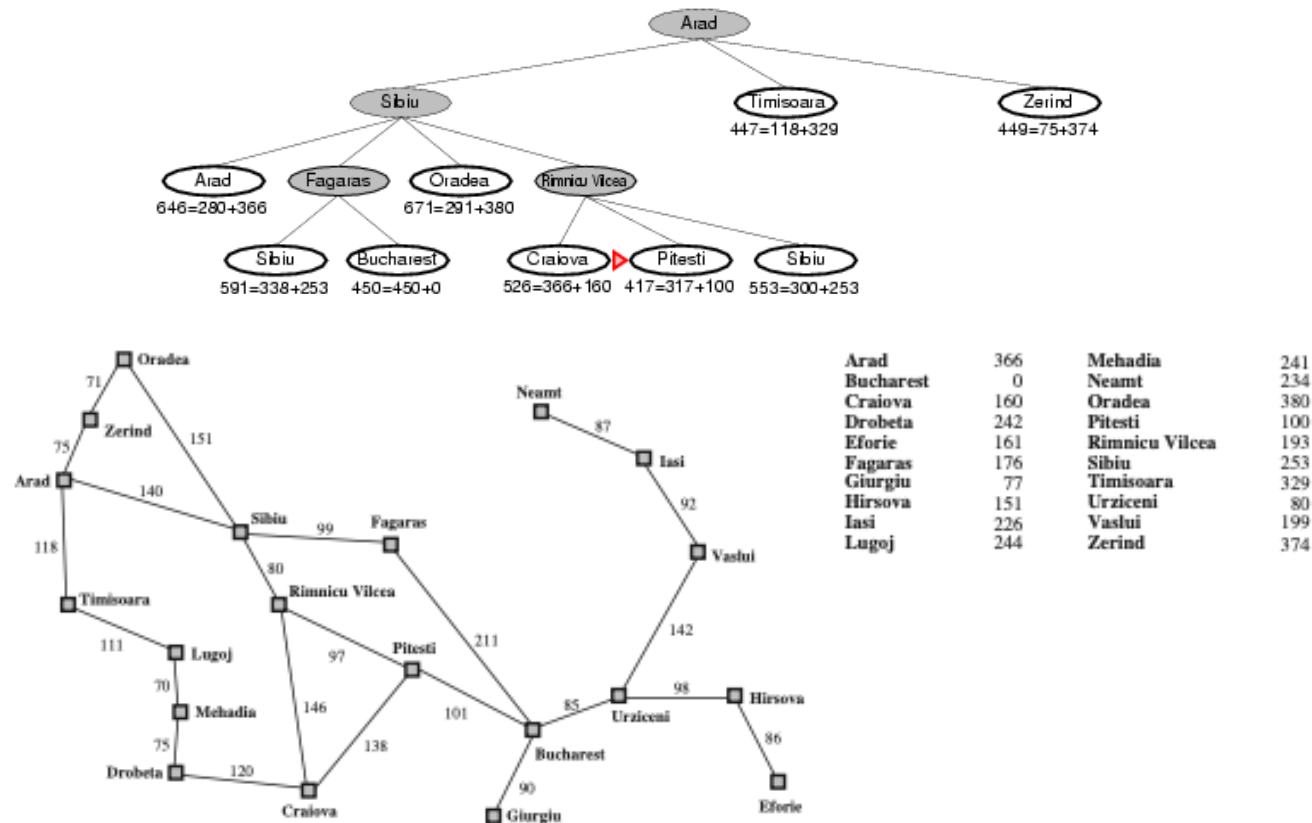
# A\* search example



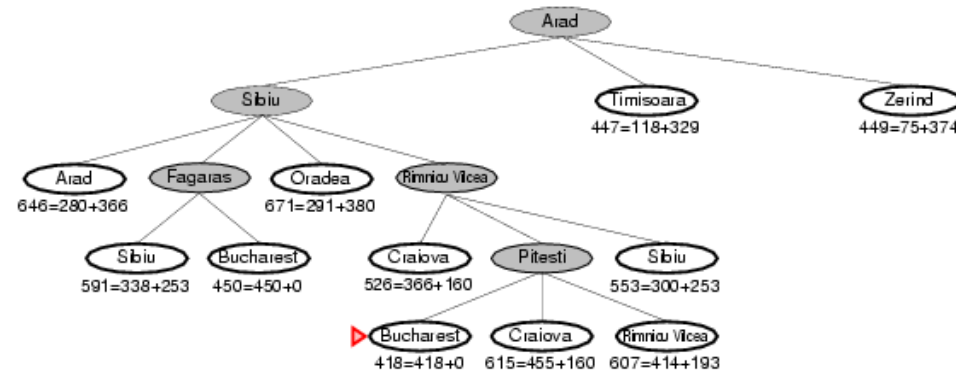
# A\* search example



# A\* search example



# A\* search example



# Informed search strategies

- Best-first search
- Greedy best-first search
- A\* search
- **Heuristics**
- Local search algorithms
  - Hill-climbing search
  - Simulated annealing search
  - Local beam search
  - Genetic algorithms



# Admissible heuristics

- Objective:
  - produce a solution that is good enough for solving the problem at hand
  - This solution may not be the best but approximate the exact solution
- $h(n)$  is **admissible** if for node  $n$ ,  $h(n) \leq h^*(n)$ , where  $h^*(n)$  is the **true** cost to reach the goal state from  $n$
- An admissible heuristic **never overestimates** the cost to reach the goal, i.e., it is **optimistic**
- Example:  $h_{SLD}(n)$  (never overestimates the actual road distance)
- **Theorem:** If  $h(n)$  is admissible,  $A^*$  using TREE-SEARCH is optimal

# Properties of A\*

- Complete? Yes (unless there are infinitely many nodes with  $f \leq f(G)$ )
- Time? Exponential
- Space? Keeps all nodes in memory
- Optimal? Yes

# Admissible heuristics for 8-puzzle

- Heuristic: produce a solution in a reasonable time frame, good enough to solve the problem
- The average cost for the 8-puzzle are approx. 22 steps. Here are 26 steps.
- Branching factor is approx. 3
  - Empty in the middle, 4 mov
  - Empty in the corner, 2 mov
  - Rest cases, 3 mov
- Depth-first search will look  $3^{22}$  states
- If we keep track of repeated states, we could reduce to 170.000
- In the 15-puzzle =  $10^{13}$

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

# Admissible heuristics for 8-puzzle

E.g., for the 8-puzzle:

- $h_1(n)$  = number of misplaced tiles
- $h_2(n)$  = total Manhattan distance  
(i.e., + horizontal and vertical distance from desired location of each tile)

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

- $h_1(S) = ?$
- $h_2(S) = ?$

# Admissible heuristics for 8-puzzle

E.g., for the 8-puzzle:

- $h_1(n)$  = number of misplaced tiles
- $h_2(n)$  = total Manhattan distance

(i.e., + horizontal and vertical distance from desired location of each tile)

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

- $h_1(S) = ?$  8
- $h_2(S) = ?$   $3+1+2+2+2+3+3+2 = 18$

# Dominance

- If  $h_2(n) \geq h_1(n)$  for all  $n$  (both admissible) then  $h_2$  **dominates**  $h_1$
- $h_2$  is better for search
- Typical search costs (average number of nodes expanded):
  - $d=12$  IDS = 3,644,035 nodes  
     $A^*(h_1) = 227$  nodes  
     $A^*(h_2) = 73$  nodes
  - $d=24$  IDS = too many nodes  
     $A^*(h_1) = 39,135$  nodes  
     $A^*(h_2) = 1,641$  nodes

# Relaxed problems

- A problem with fewer restrictions on the actions is called a **relaxed problem**
- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem
- If the rules of the 8-puzzle are relaxed so that a tile can move **anywhere**, then  $h_1(n)$  gives the shortest solution
- If the rules are relaxed so that a tile can move to **any adjacent square**, then  $h_2(n)$  gives the shortest solution

# Informed search strategies

- Best-first search
- Greedy best-first search
- A\* search
- Heuristics
- **Local search algorithms**
  - Hill-climbing search
  - Simulated annealing search
  - Local beam search



# Local search algorithms

- In many optimization problems, the **path** to the goal is irrelevant; the goal state itself is the solution
- State space = set of "complete" configurations. Find configuration satisfying constraints, e.g., n-queens
- In such cases, we can use **local search algorithms**
- Keep a single "current" state, try to improve it
- Work with one current state and generally moves to the neighboring state
- The paths followed by the search are not retained
  - They use little memory
  - You can find reasonable solutions in large state spaces or infinite

## Example: $n$ -queens

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



# Informed search strategies

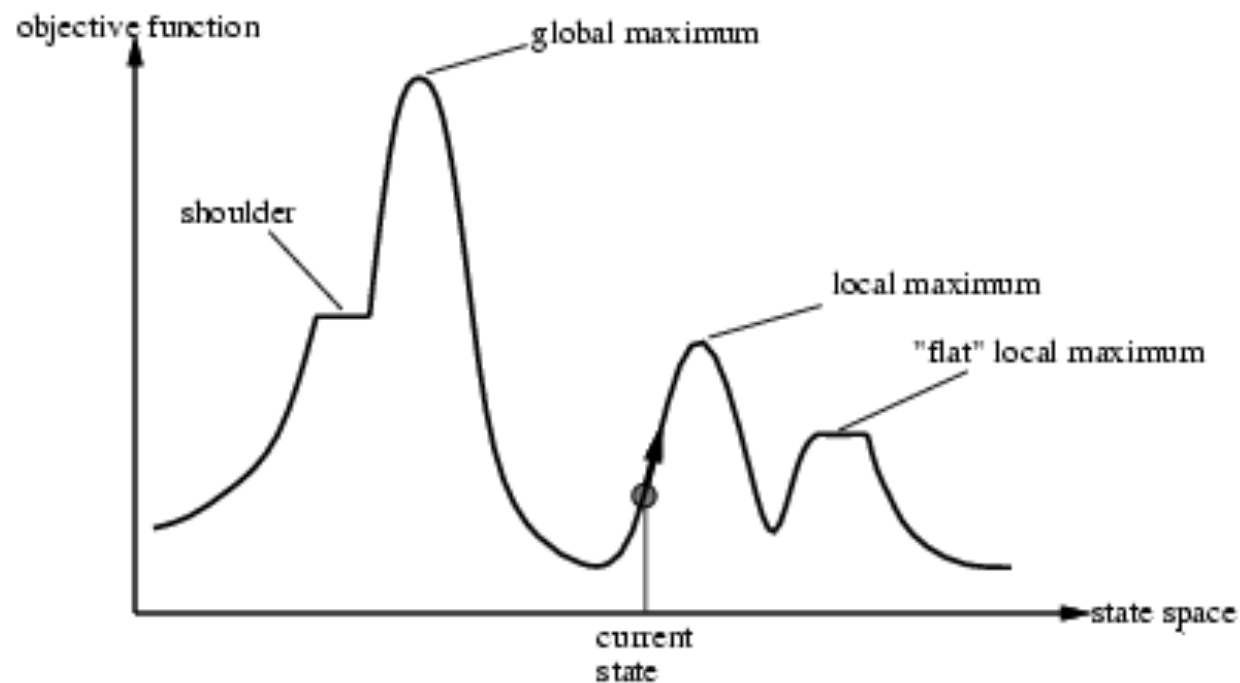
- Best-first search
- Greedy best-first search
- $A^*$  search
- Heuristics
- Local search algorithms
  - **Hill-climbing search**
  - Simulated annealing search
  - Local beam search

# Hill-climbing search

- It's just a loop that moves in the direction of increasing value
  - Ends when it reaches a peak where no neighbor has a higher value
  - The search tree is not kept, just a data structure of the current node to check the goal condition and its objective function value
- "Like climbing Everest in thick fog with amnesia"

# Hill-climbing search

- Problem: depending on initial state, can get stuck in local maxima

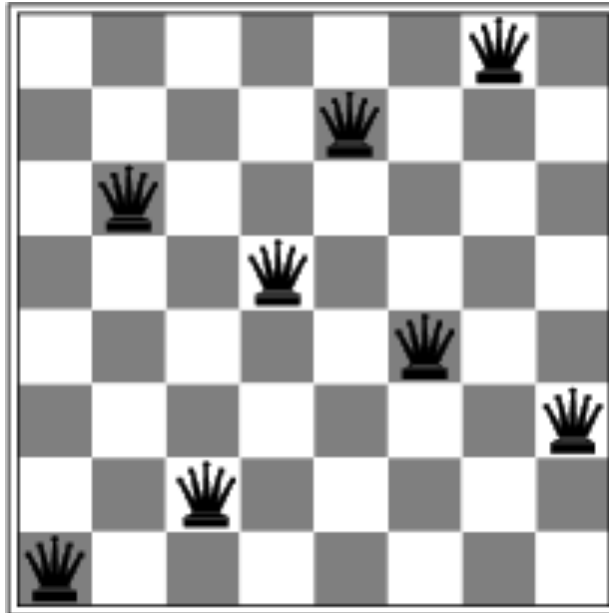


# Hill-climbing search: 8-queens problem

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

- $h$  = number of pairs of queens that are attacking each other, either directly or indirectly
- $h = 17$  for the above state
- The figure also shows the values of all successors, top successors have  $h = 12$

## Hill-climbing search: 8-queens problem



- A local minimum with  $h = 1$  (obtained in 5 steps)

# Hill-climbing search

- The algorithm gets stuck for several reasons:
  - Local Maximum: it is a peak that is higher than each of its neighbours, but lower than the maximum overall
  - Ridges: cause a sequence of local maxima that make navigation difficult
  - Plateau (flat): can lead to a local maximum where there is no ascendant exit or a terrace to advance
- In the 8-queens, it gets stuck in 86% and solve 14% cases
- If we allow lateral movements with the hope that we find a terrace (limiting them if reach a local maximum, e.g.100): → 94% success
- Variants:
  - Stochastically (randomly chooses upward movements)
  - Random restart (the initial states are generated randomly)



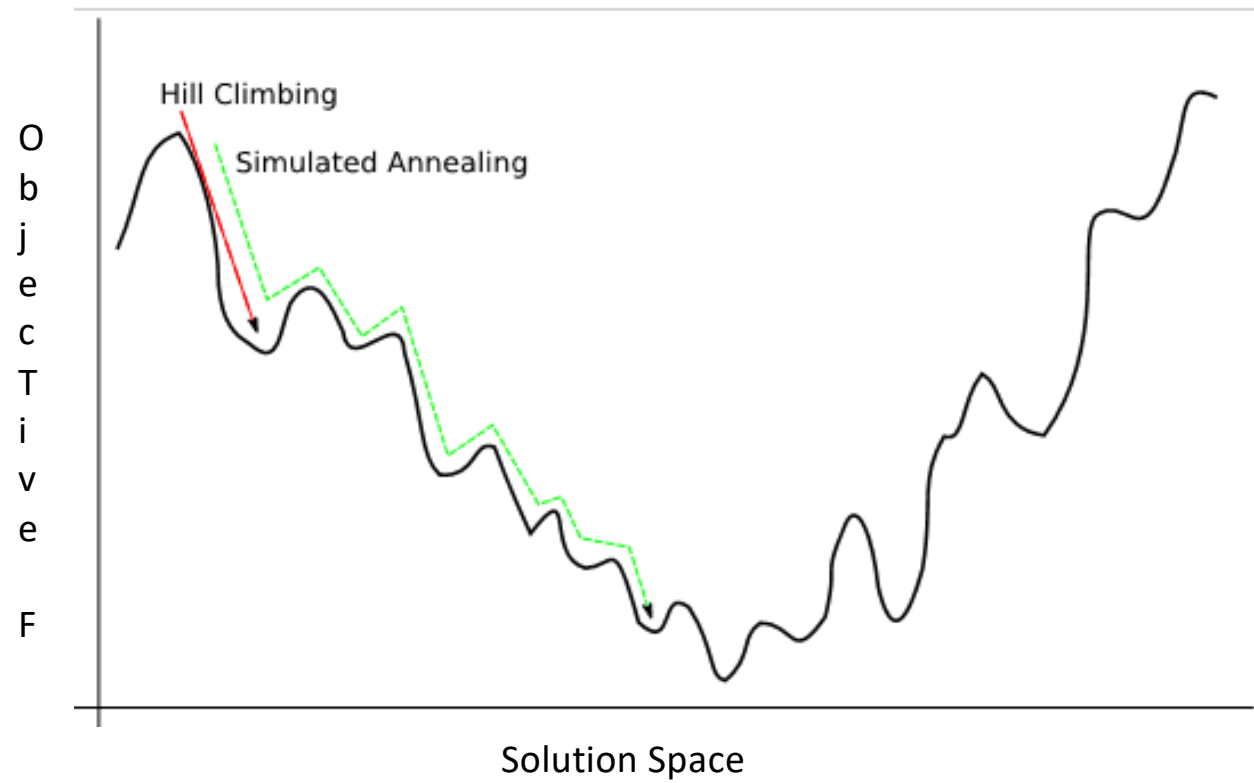
# Informed search strategies

- Best-first search
- Greedy best-first search
- A\* search
- Heuristics
- Local search algorithms
  - Hill-climbing search
  - **Simulated annealing search**
  - Local beam search

# Simulated annealing search

- Process of tempering or hardening metals by heating and then cooling them gradually
- Idea: escape local maxima by allowing some "bad" moves but **gradually decrease** their frequency
- It combines hill-climbing with random generation successor
- Good for problems with a large search space, optimum is surrounded by many local optima
- Problem: determine the values of the parameters, and requires an important work of experimentation that depends on each problem
- Widely used in VLSI layout, airline scheduling, etc

# Simulated annealing search



# Informed search strategies

- Best-first search
- Greedy best-first search
- A\* search
- Heuristics
- Local search algorithms
  - Hill-climbing search
  - Simulated annealing search
  - **Local beam search**

# Local beam search

- Idea: Keep track of  $k$  states rather than just one
  - Start with  $k$  randomly generated states
  - At each iteration, all the successors of all  $k$  states are generated
  - If any one is a goal state, stop; else select the  $k$  best successors from the complete list and repeat
  - Alternatively stochastic LBS randomly choose  $k$  successors, with the probability of choosing a successor as an increasing function of its value

# Outline

- Introduction
- Problem formulation
- Problem types
- Basic search algorithms
- **Conclusions**

# Conclusions

- Problem formulation usually requires abstracting away real-world details to define a state space that can feasibly be explored
- Variety of uninformed and informed search strategies