

# Search

Inteligencia Artificial en los Sistemas de Control Autónomo  
Máster en Ciencia y Tecnología desde el Espacio

Departamento de Automática

## Objectives

1. Understand the role of search in AI
2. Describe the importance of trees in search
3. Express AI problems in terms of search
4. Apply classical search algorithms (uninformed, informed and local)

## Bibliography

- S. Russell and P. Norvig. Chapter 3, Solving Problems by Searching. Artificial Intelligence: A Modern Approach. Pearson. 2017
- S. Russell and P. Norvig. Chapter 4, Beyond Classical Search. Artificial Intelligence: A Modern Approach. Pearson. 2017

# Table of Contents

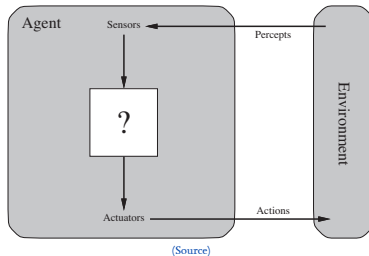
1. Introduction
2. Problem formulation
  - Types of problems
  - Problem components
  - Toy problems
  - Travel Salesman Problem
3. Search strategy
4. Uninformed search
  - Breadth-first search
  - Uniform-cost search
  - Depth-first search
  - Depth-limited search
  - Iterative deepening depth-first search
  - Comparison of uninformed search algorithms
5. Informed search
  - Introduction
  - Best-first search
  - Greedy best-first search
  - A\*
  - Local search
  - Hill-climbing
6. Case studies
  - Case study I: Robot arm with two DOF
  - Case study II: 9<sup>th</sup> GTOC
  - Case study III: Mars orbital insertion
  - Case study IV: Transonic wing shape optimization

# Introduction

## Intelligent agent

### Agent

An agent is anything that can be viewed as perceiving its environment through sensors and acting through actuators



- Agents is a research field in AI by its own
  - ... with its own definition of agent (caution!)
- We use this term to abstract the implementation

# Introduction

## Motivation

Early AI works were directed to

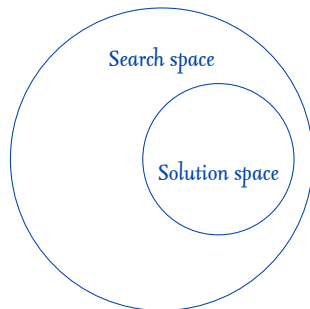
- Proof of theorems, crosswords, games, ...

All in AI is search ...

- ... not entirely true (obviously) but more than we may imagine
- Find a good/best solution (**solution space**) to a problem among several potential solutions (**search space**)

**Enhaustive search** (or brute-force search)

- Iterate over all the potential solutions
- Unsuitable for most real-world problems



# Problem formulation

## Types of problems

Types of problems depending on ...

- Knowledge
  - Observable or Non-observable or Partially observable
- Outcome
  - Deterministic or Stochastic
- Actions
  - Discrete or Continuous
- Time-variance
  - Static or Dynamic

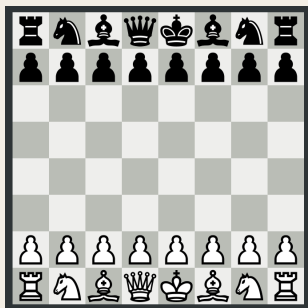
We assume static, observable, discrete and deterministic problems

## Problem formulation

## Types of problems (II)

## Determine problem type

## Chess



## League of Legends



Observable or non-observable, deterministic or stochastic, discrete or continuous, static or dynamic?

# Problem formulation

## Problem components (I)

We represent the environment as **states**

- Contain the information about the world

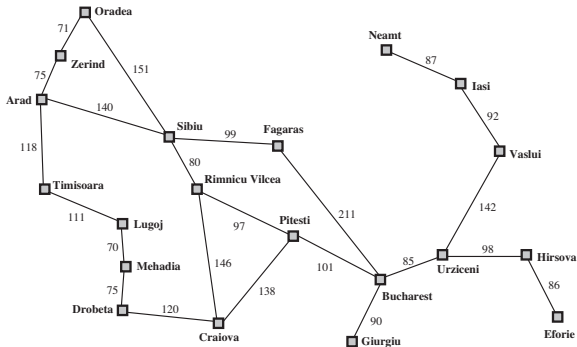
Any problem formulation requires the following components

- **Initial state:** State where the search begins
- **Actions:** Behaviour that the agent may exhibit
- **Transition model:** Which states follow an action in a state (**graph**)
- **Goal test** (*metas*): How to determine if a state is a goal
- **Path cost:** Cost of a path to a state



# Problem formulation

## Problem components: Example (I)



(Source)

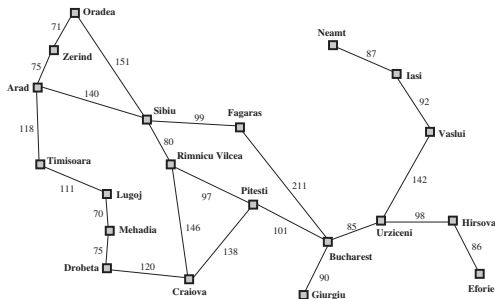
Problem: Move from Arad to Bucharest

On holiday in Romania; currently in Arad, flight leaves tomorrow from Bucharest

Determine: Initial state, goal, states, actions, transition model, goal test and path cost

# Problem formulation

## Problem components: Example (II)



(Source)

## Solution

- Initial state: Arad
- Goal: Bucharest
- States: Multiple cities
- Actions: Drive between cities
- Goal test: In Bucharest?
- Path cost: Distance

# Problem formulation

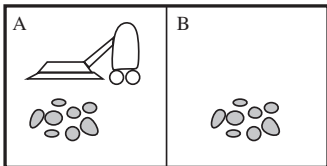
## Problem components: Search

**Search** is the process of finding a solution

- A **solution** is a sequence of actions leading from the initial state to a goal state
- **Optimal solution** is a solution with the lowest cost
- Example of solution: Arad  $\rightarrow$  Sibiu  $\rightarrow$  Fagaras  $\rightarrow$  Bucharest
  - That solution is optimal?

# Search problems

## Toy problems (I): Vacuum world



(Source)

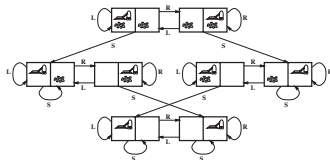
### Problem: Clean rooms

- State? →
- Initial state? →
- Goal? →
- Actions? →
- Transition model? →
- Goal test? →
- Path cost? →

# Search problems

## Toy problems (I): Vacuum world

### Problem: Clean rooms



(Source)

- State? → Dirt and location
- Initial state? → All dirt, Left
- Goal? → No dirt, any location
- Actions? → Left, Right, Suck
- Transition model? → See figure
- Goal test? → No dirt, any location
- Path cost? → 1 per action

# Search problems

## Toy problems (II): 8-puzzle

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

(Source)

### Problem: Solve 8-puzzle

- State? →
- Initial state? →
- Goal? →
- Actions? →
- Transition model? →
- Goal test? →
- Path cost? →

# Search problems

## Toy problems (II): 8-puzzle

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

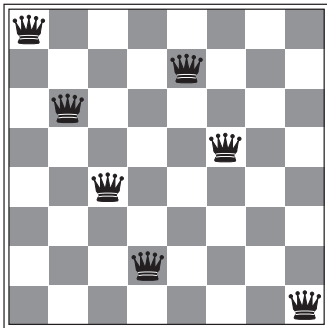
(Source)

### Problem: Solve 8-puzzle

- State? → Location of tiles  
 $9!/2 = 181,440$  states
- Initial state? → Any
- Goal? → See figure
- Actions? → Left, Right, Up, Down
- Transition model? → Complex graph
- Goal test? → Goal state
- Path cost? → 1 per move

# Search problems

## Toy problems (III): 8-queens



(Source)

Problem: Place 8 queens no queen attacks any other

State?

Initial state?

Goal?

Actions?

Transition model?

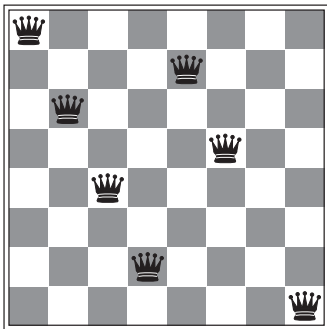
Goal test?

Path cost?



# Search problems

## Toy problems (III): 8-queens



(Source)

Problem: Place 8 queens no queen attacks any other

- State? → Any arrangement of 0 to 8 queens
- Initial state? → Empty board
- Goal? → See figure
- Actions? → Add queen to empty square
- Transition model? → Complex graph
- Goal test? → 8 queens on board, none attacked
- Path cost? → 1 per move

# Search problems

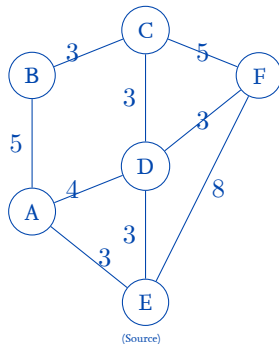
## Travelling Salesman Problem (TSP)

### TSP formulation

A travelling salesman must visit a set of cities only one time each. Find the shortest route.

TSP is a very big problem in AI!

- First formulated in 1930 and still a hot research topic!
- NP-hard problem
- Many real world applications



# Search strategy (I)

In general ...

- Each problem has a search graph, or **state space**
- Searching means finding a path from the initial state to a goal state

Basic idea

- Explore search space
- Generate a **search tree** (i.e., **expanding nodes**)

A search strategy is defined by picking the order of node expansion

- Uninformed search: Only uses the problem definition
- Informed search: Uses problem-specific knowledge

## Search strategy (II)

Search strategies are evaluated along the following dimensions

- Completeness
- Time complexity
- Space complexity
- Optimality

Time and space are measured in terms of

- $b$ : Maximum branching factor
- $d$ : Depth of the least-cost solution
- $m$ : Maximum depth of the state space

# Uninformed search

## Uninformed search algorithms

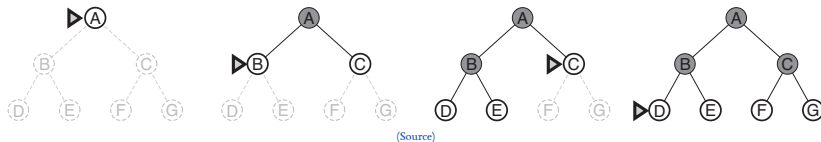
- Breadth-first search (búsqueda en anchura)
- Uniform-cost search (búsqueda de coste uniforme)
- Depth-first search (búsqueda en profundidad)
- Depth-limited search (búsqueda en profundidad limitada)
- Iterative deepening search (búsqueda de profundización iterativa)

# Uninformed search

## Breadth-first search (I)

Expand shallowest unexpanded node

- Implemented with a FIFO queue (First-In First-Out)



# Uninformed search

## Breadth-first search (II)

Depth	Nodes	Time	Memory
2	110	1.1 milliseconds	107 kilobytes
4	11,110	111 milliseconds	10.6 megabytes
6	$10^6$	11 seconds	1 gigabytes
8	$10^8$	19 minutes	103 gigabytes
10	$10^{10}$	31 hours	10 terabytes
12	$10^{12}$	129 days	1 petabytes
14	$10^{14}$	35 years	99 petabytes
16	$10^{16}$	3,500 years	10 exabytes

**Figure 3.13** Time and memory requirements for breadth-first search. The numbers shown assume branching factor  $b = 10$ ; 100,000 nodes/second; 1000 bytes/node.

(Source)

### Properties of breadth-first search

- Completeness: Yes
- Time complexity:  $O(b^{d+1})$
- Space complexity:  $O(b^{d+1})$
- Optimality: Yes (if cost = 1 per step)

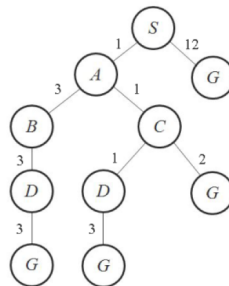
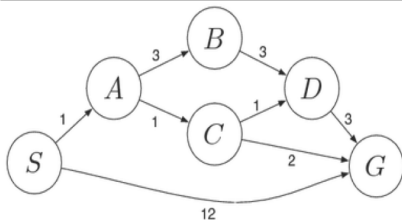
Space is the biggest problem (more than time)

# Uninformed search

## Uniform-cost search (I)

Special case of breadth-first search

- Expand least-cost unexpanded node
- The queue is sorted by cost





# Uninformed search

## Uniform-cost search (II)

### Uniform-cost example

Initialization:  $\{[S, 0]\}$

Iter. 1:  $\{[S \rightarrow A, 1], [S \rightarrow G, 12]\}; V = S$

Iter. 2:  $\{[S \rightarrow A \rightarrow C, 2], [S \rightarrow A \rightarrow B, 4], [S \rightarrow G, 12]\}; V = S, A$

Iter. 3:  $\{[S \rightarrow A \rightarrow C \rightarrow D, 3], [S \rightarrow A \rightarrow C \rightarrow G, 4], [S \rightarrow A \rightarrow B \rightarrow D, 7], [S \rightarrow G, 12]\}; V = S, A, C$

Iter. 4:  $\{[S \rightarrow A \rightarrow C \rightarrow D \rightarrow G, 6], [S \rightarrow A \rightarrow C \rightarrow G, 4], [S \rightarrow A \rightarrow B \rightarrow D, 7], [S \rightarrow G, 12]\}; V = S, A, C, D$

Iter. 5:

$\{[S \rightarrow A \rightarrow C \rightarrow G, 4], [S \rightarrow A \rightarrow C \rightarrow D \rightarrow G, 10], [S \rightarrow G, 12]\}; V = S, A, C, D, B$

Solution:  $S \rightarrow A \rightarrow C \rightarrow G$

# Uninformed search

## Uniform-cost search (III)

### Properties

- Completeness: Yes, if step cost  $\geq \epsilon$
- Time complexity:  $O(b^{\lceil C^*/\epsilon \rceil})$ , where  $C^*$  is the cost of the optimal solution
- Space complexity:  $O(b^{\lceil C^*/\epsilon \rceil})$
- Optimality: Yes

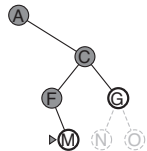
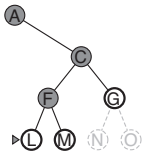
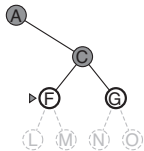
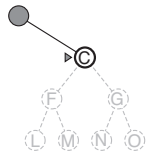
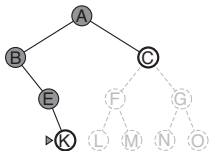
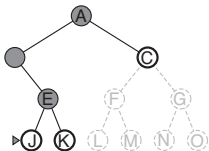
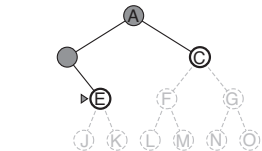
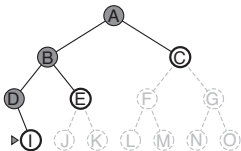
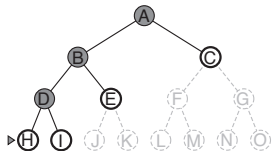
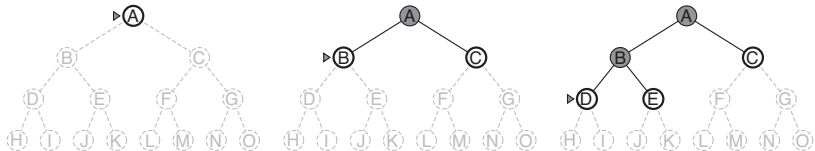
Space is the biggest problem (more than time)

# Uninformed search

## Depth-first search (I)

Expand deepest unexpanded node

- Implemented with a LIFO stack



# Uninformed search

## Depth-first search (III)

### Properties of depth-first search

- Completeness: No, fail in infinite-depth spaces or spaces with loops
- Time complexity:  $O(b^m)$ , (terrible if  $m \gg d$ )
- Space complexity:  $O(bm)$
- Optimality: No

# Uninformed search

## Depth-limited search

Depth-first search with depth limit  $L$

- Nodes at depth  $L$  are not expanded

# Uninformed search

## Iterative deepening depth-first search (I)

Depth-limited search where gradually increases  $L$

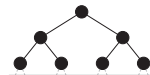
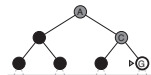
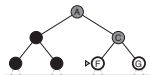
Limit = 0



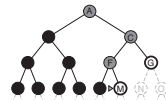
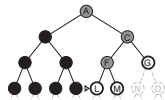
Limit = 1



Limit = 2



Limit = 3



(Source)



# Uninformed search

## Iterative deepening depth-first search (III)

### Properties

- Completeness: Yes
- Time complexity:  $O(b^d)$
- Space complexity:  $O(bd)$
- Optimality: Yes if step cost = 1

# Uninformed search

## Comparison of uninformed search algorithms

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

# Informed search

## Introduction (I)

Use problem-specific knowledge beyond problem definition

- Best-first search (búsqueda primero el mejor)
  - Greedy best-first search (Búsqueda voraz)
  - A\* search
- Local search algorithms
  - Hill-climbing search (búsqueda en escalada)
  - Simulated annealing search (búsqueda de temple simulado)
  - Local beam search (búsqueda de haz local)
  - Genetic Algorithms

# Informed search

## Beast-first search

Use an evaluation function  $f(n)$  for each node

- $f$  is a cost estimate, i.t, its “desirability”
- Expand most desirable unexpanded nodes
- $n$  is a node

Most algorithms use a **heuristic function** or just **heuristic** ( $h(n)$ )

- Estimated cost from a state to the goal
- $h$  only depends on the state (does not consider the path)
- $h$  is a negative, nonnegative problem-specific function
- If  $n$  is a goal node, then  $h(n) = 0$

The choice of  $f$  determines the search strategy

- Greedy best-first search
- $A^*$

# Informed search

## Greedy best-first search (I)

It only considers the heuristic

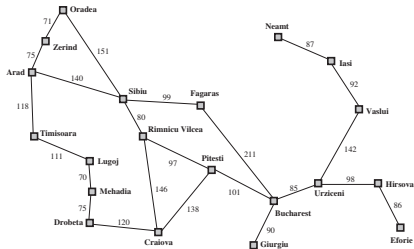
- $f(n) = h(n)$
- Remember:  $h$  is the estimate cost from a state to the goal

### Greedy search

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

Example: Find a path between Arad and Bucharest

- Heuristic: Straight-line distance



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

(Source)

Search

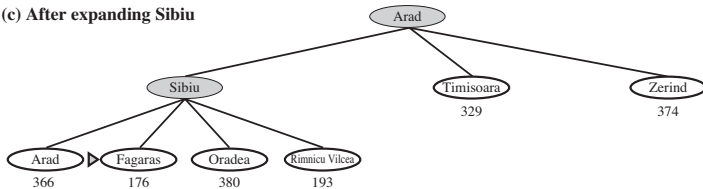
(a) The initial state



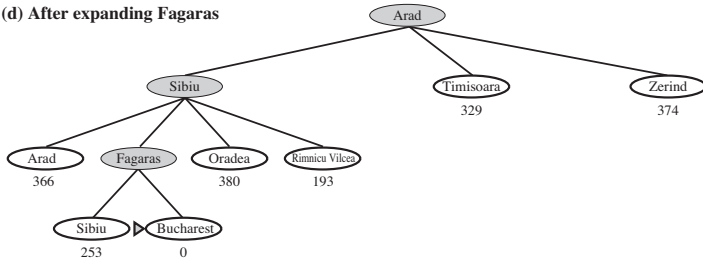
(b) After expanding Arad



(c) After expanding Sibiu



(d) After expanding Fagaras



# Informed search

## A\* (I)

Idea: avoid expanding paths that are already expensive

- It consider the path to a state (past), and its estimated cost to goal (future)

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

- $g(n)$ : Cost so far to reach node  $n$
- $h(n)$ : Estimated cost from node  $n$  to goal
- $f(n)$ : Estimated total cost of path through  $n$  to goal

A\*

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

Theorem: A\* is optimal if  $h(n)$  is **admissible**

- A\* is admissible if it never overestimates the cost
- Example: Straight-line distance never overestimates road distance

(a) The initial state



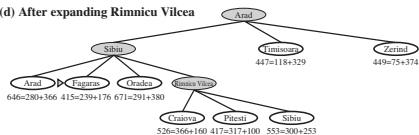
(b) After expanding Arad



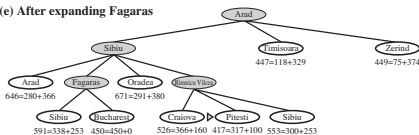
(c) After expanding Sibiu



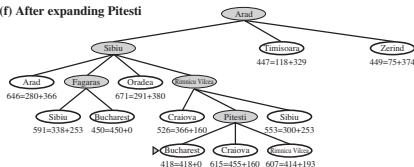
(d) After expanding Rimnicu Vilcea



(e) After expanding Fagaras



(f) After expanding Pitesti





# Informed search

## A\* (III)

### Properties

- Completeness: Yes
- Time complexity: Exponential
- Space complexity: Keeps all nodes in memory
- Optimality: Yes

# Informed search

## Local search: Introduction (I)

In many optimization problems, the path to the goal is irrelevant; the goal itself is the solution

- The path is stored in memory

Solution: Keep a single “current” state and try to improve it

- Generally, moving to the neighboring state

The paths followed by the search are not retained

- They use little memory
- They can find reasonable solutions in large or even infinite state spaces

# Informed search

## Local search: Introduction (II)

### 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

## Hill-climbing search (I)

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
- No search tree is kept, just a datastructure with the current node

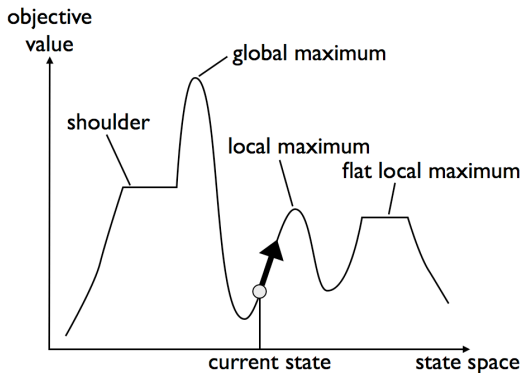
“Like climbing Everest in thick fog with amnesia”

# Informed search

## Hill-climbing search (II)

Good for pure optimization problems

- No obvious cost function for such problems
- **Objective function:** How good a state is



# Informed search

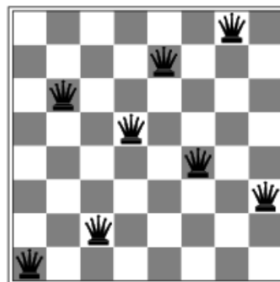
## Hill-climbing search (III)

### 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	14	♚	16	
14	14	13	17	12	14	12	18

$h$  = number of pairs of queens that are attacking each other

- $h = 0$  for the above state
- All successors  $h \geq 12$



A local minimum ( $h = 1$ )

# Informed search

## Hill-climbing search (IV)

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): no ascendant exit

In the 8-queens, it gets stuck in 86 %

- If we allow lateral movements  $\rightarrow$  100 %

Variants

- Stochastically chooses upward movements
- Random restart (random generation of initial state)

# Informed search

## Simulated annealing search (I)

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

Combination of hill-climbing with random generation successor

- Problem: determine its parameters, need of experimentation

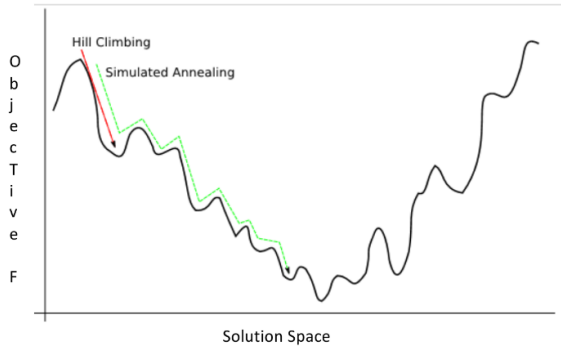
Application

- Good for problems with a large search space, optimum is surrounded by many local optima
- Widely used in VLSI layout, airline scheduling, etc



# Informed search

## Simulated annealing search (II)



# Informed search

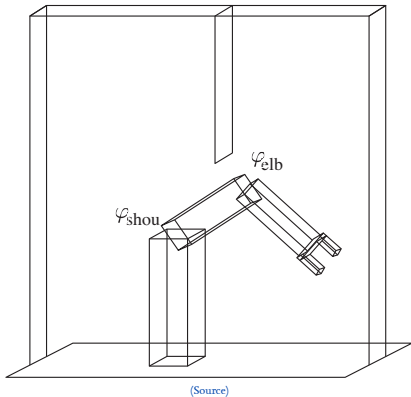
## Local beam search

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

# Case studies

## Case study I: Robot arm with two DOF (I)

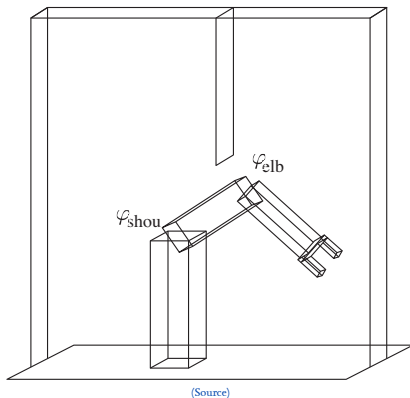


### Problem: Move arm

- State? →
- Actions? →
- Goal test? →
- Path cost? →

# Case studies

## Case study I: Robot arm with two DOF (I)

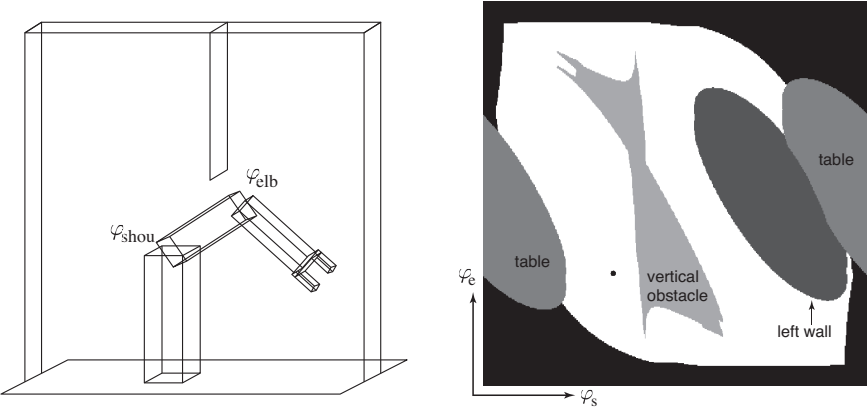


### Problem: Move arm

- State? → Real-valued coordinates of robot joint angles
- Actions? → Continuous motions of joints
- Goal test? → Complete assembly
- Path cost? → Time to complete

# Case studies

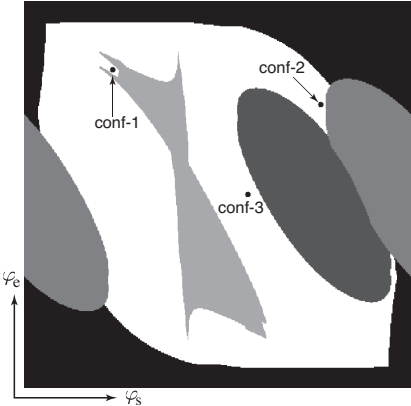
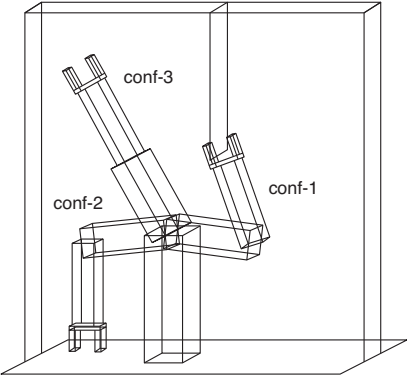
## Case study I: Robot arm with two DOF (II)



(Source)

# Case studies

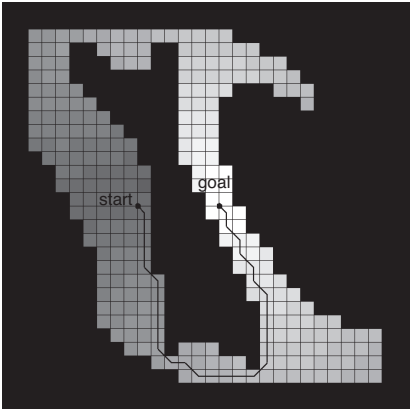
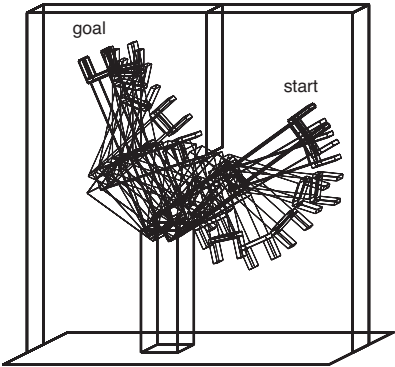
## Case study I: Robot arm with two DOF (III)



(Source)

# Case studies

## Case study I: Robot arm with two DOF (IV)



(Source)

# Study case

## Case study II: 9<sup>th</sup> Global Trajectory Optimization Competition (I)

### GTOC: Global Trajectory Optimization Competition

- Proposed by ESA Advanced Concepts Team
- Difficult trajectory optimization problems

### GTOC 9: The Kessler Run

- 123 orbiting debris
- Remove debris
- Design multiple missions

(Video) (Solution)



# Study case

## Case study III: Mars orbital insertion

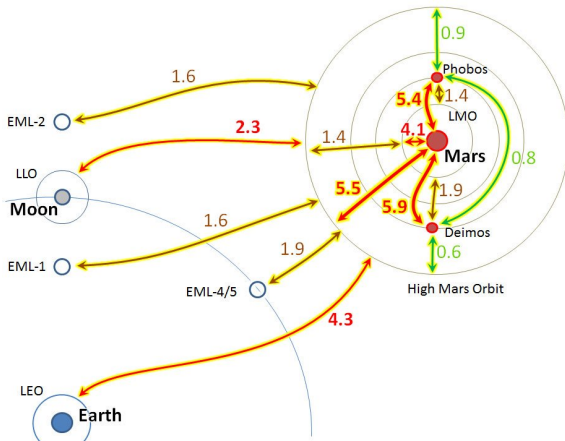


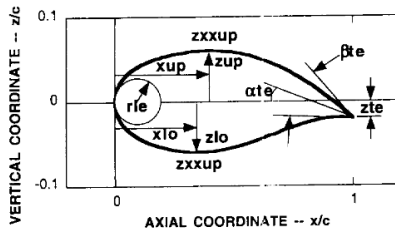
Chart by Richard Penn CC-BY, data from David Hollister hopsblog-hop.blogspot.co.uk

# Study case

## Case study IV: Transonic wing shape optimization

Problem: Design a wing shape for transonic flight

- Maximize lift



Holst T.L., Pulliam T.H. (2003) Transonic Wing Shape Optimization Using a Genetic Algorithm. In: IUTAM Symposium Transsonicum IV. Fluid Mechanics and its Applications, vol 73. Springer.