Solving Problems by Searching

Inteligencia Artificial en los Sistemas de Control Autónomo
Máster Universitario en Ingeniería Industrial

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

Bibliography

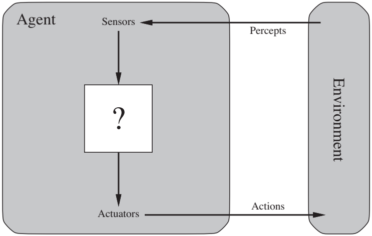
- S. Russell and P. Norvig. Chapter 3, Solving Problems by Searching. *Artificial Intelligence: A Modern Approach*. Pearson. 2017

Table of Contents

Introduction

Agent

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



(Source)

- 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

Search problems

Types of problems

Search is a cornerstone in AI

- Almost any problem in AI is formulated as a search problem

Types of problems depending on ...

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

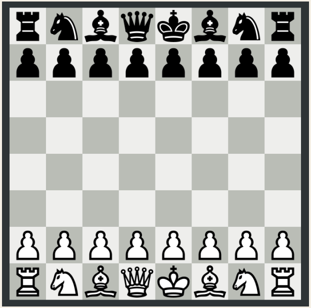
We assume static, observable, discrete and deterministic problems

Search problems

Types of problems (II)

Determine problem type

Chess



League of Legends

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

Search problems

Problem components (I)

We represent the environment as states

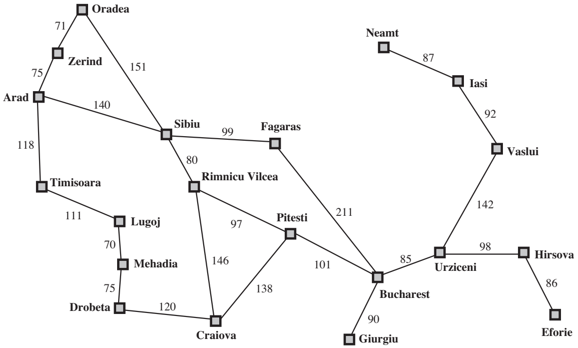
- Contain the information about the world

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
- Goal test (*metas*): How to determine if a state is a goal
- Path cost: Cost of a path to a state

Search problems

Problem components (II)



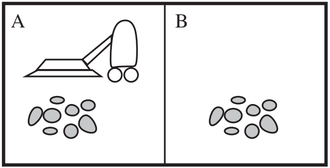
(Source)

Problem: Move from Arad to Bucharest

Initial state? Goal? Actions? Transition model? Goal test? Path cost?

Search problems

Toy problems (I): Vacuum world



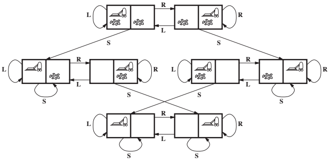
(Source)

Problem: Clean rooms

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

Search problems

Toy problems (I): Vacuum world



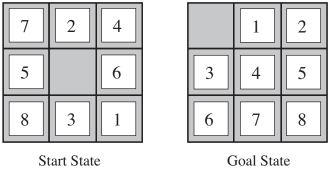
(Source)

Problem: Clean rooms

- 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



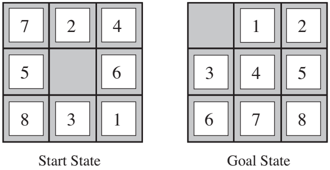
(Source)

Problem: Solve 8-puzzle

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

Search problems

Toy problems (II): 8-puzzle



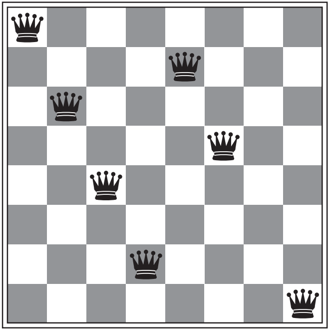
(Source)

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



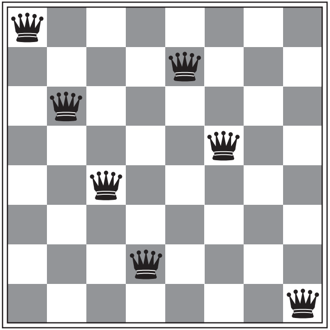
(Source)

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

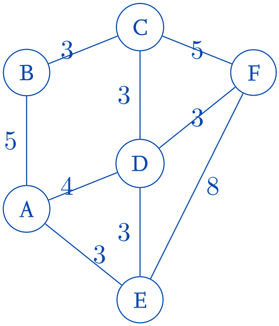
Travelling Salesman Problem (TSP)

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



(Source)

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 exansion

- 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

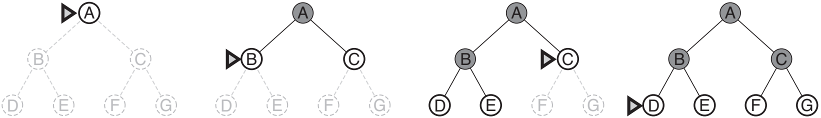
- b: Maximum branching factor
- d: Depth of the least-cost solution
- m: Maximum depth of the state space

Uninformed search

Breadth-first search (I)

Expand shallowest unexpanded node

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



(Source)

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)

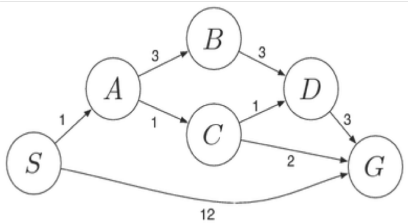
Space is the biggest problem (more than time)

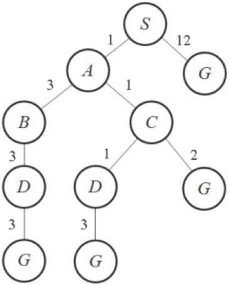
Uninformed search

Uniform-cost search (I)

Expand least-cost unexpanded node

- Implemented with a queue ordered by path cost





Uninformed search

Uniform-cost search (II)

Uniform-cost example

Initialization: $\{[S, 0]\}$
Iter. 1: $\{[S \to A, 1], [S \to G, 12]\}$
Iter. 2: $\{[S \to A \to C, 2], [S \to A \to B, 4], [S \to G, 12]\}$
Iter. 3:
$\{[S \to A \to C \to D, 3], [S \to A \to C \to G, 4], [S \to A \to B \to D, 7], [S \to G, 12]\}$
Iter. 4: $\{[S \to A \to C \to D \to G, 6], [S \to A \to C \to G, 4], [S \to A \to B \to D, 7], [S \to G, 12]\}$
Iter. 5: $\{[S \to A \to C \to G, 4], [S \to A \to C \to D \to G, 10], [S \to G, 12]\}$
Solution: $S \to A \to C \to G$

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

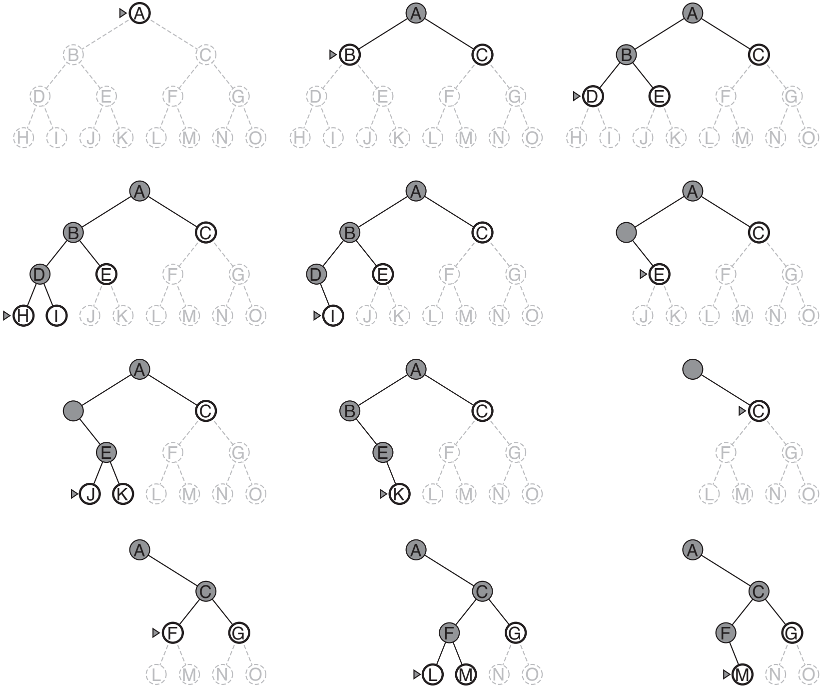
Space is the biggest problem (more than time)

Uninformed search

Depth-first search (I)

Expand deepest unexpanded node

- Implemented with a LIFO stack



(Source)

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 $m >> d$)
- Space complexity: $O(bm)$
- Optimality: No

Uninformed search

Depth-limited search

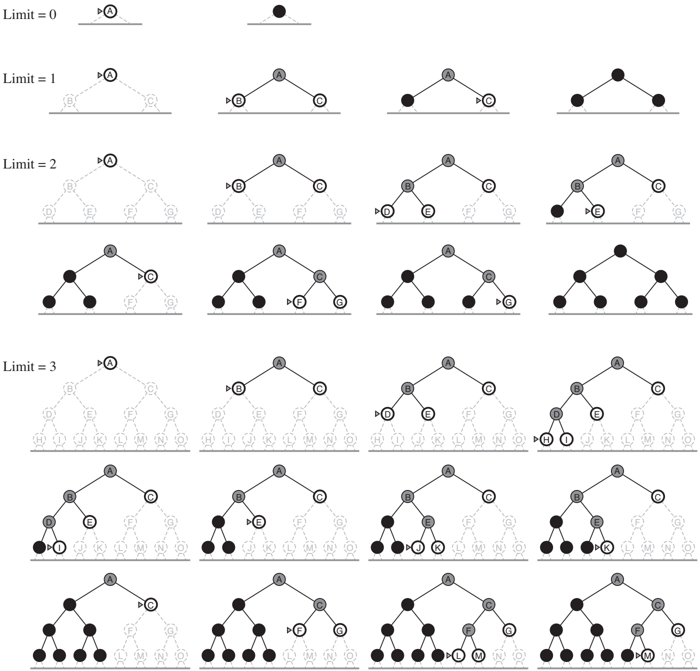
Depth-first search with depth limit L

- Nodes at depth L are not expanded

Uninformed search

Iterative deepening depth-first search (I)

Depth-limited search where gradually increases L



(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

Introduction (II)

Best-first search

- Use an evaluation function $f(n)$ for each node
- Estimate of "desirability"
- Expand most desirable unexpanded nodes

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

- Estimated cost from a state to the goal

Best-first algorithms

- Greedy best-first search
- A*

Informed search

Greedy best-first search (I)

It only considers the heuristic

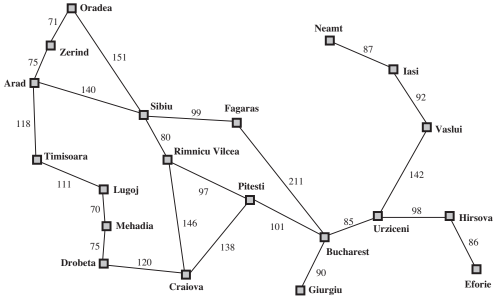
- Greedy search expands the node that *appears* to be closest to the goal

Greedy search

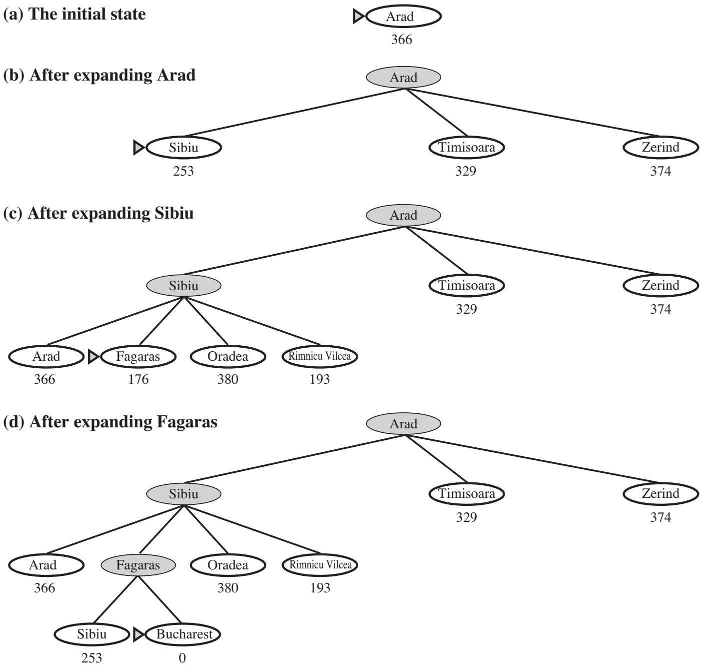
$f(n) = h(n)$

Example: Find a path between Arad and Bucharest

- Heuristic: Straight-line distance



(Source)


(a) The initial state
Arad
366
(b) After expanding Arad
Arad
Sibiu
253
Timisoara
329
Zerind
374
(c) After expanding Sibiu
Arad
Sibiu
Timisoara
329
Zerind
374
Arad
366
Fagaras
176
Oradea
380
Rimnicu Vilcea
193
(d) After expanding Fagaras
Arad
Sibiu
Timisoara
329
Zerind
374
Arad
366
Fagaras
Oradea
380
Rimnicu Vilcea
193
Sibiu
253
Bucharest
0
(Source)

Informed search

A* (I)

It considers the heuristic and the cost

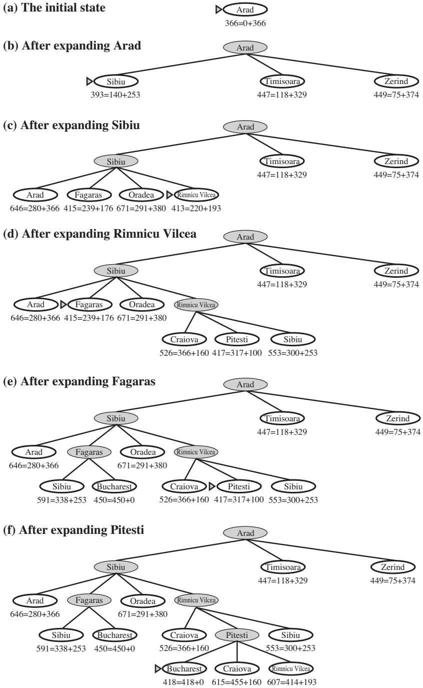
- $h(n)$: Estimated cost to goal from node n
- $g(n)$: Cost to node n

A*

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

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

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


(a) The initial state
Arad
366=0+366
(b) After expanding Arad
Arad
Sibiu
393=140+253
Timisoara
447=118+329
Zerind
449=75+374
(c) After expanding Sibiu
Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
415=239+176
Oradea
671=291+380
Rimnicu Vilcea
413=220+193
(d) After expanding Rimnicu Vilcea
Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
415=239+176
Oradea
671=291+380
Rimnicu Vilcea
Craiova
526=366+160
Pitesti
417=317+100
Sibiu
553=300+253
(e) After expanding Fagaras
Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
417=317+100
Sibiu
553=300+253
(f) After expanding Pitesti
Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
Sibiu
553=300+253
Bucharest
418=418+0
Craiova
615=455+160
Rimnicu Vilcea
607=414+193

(Source)

Informed search

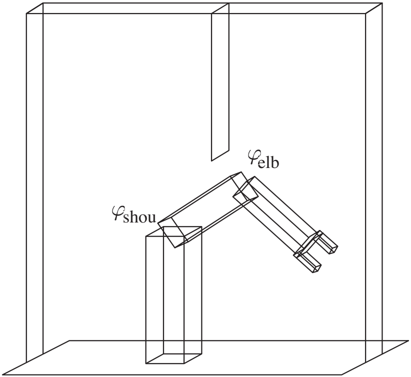
A* (III)

Properties

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

Case studies

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



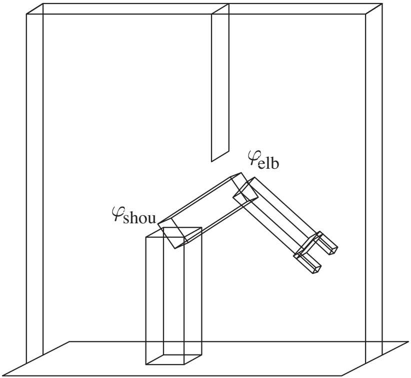
(Source)

Problem: Move arm

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

Case studies

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



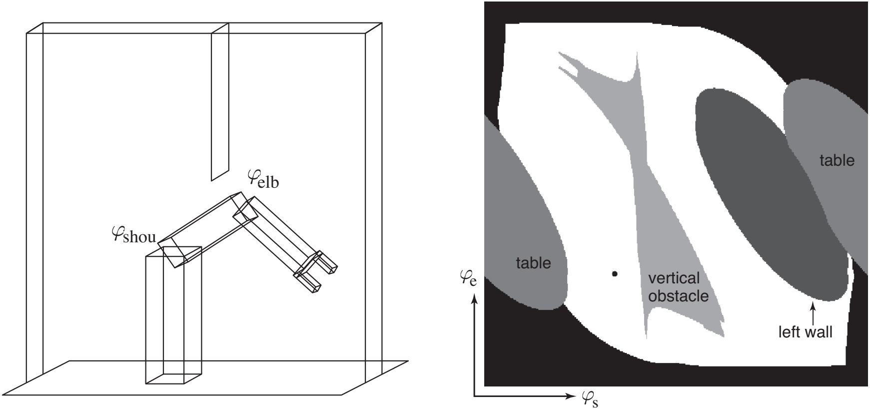
(Source)

Problem: Move arm

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

Case studies

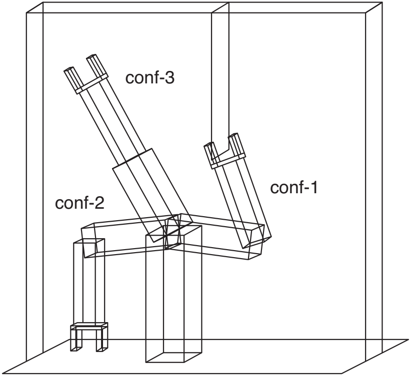
Case study I: Robot arm with two DOF (II)

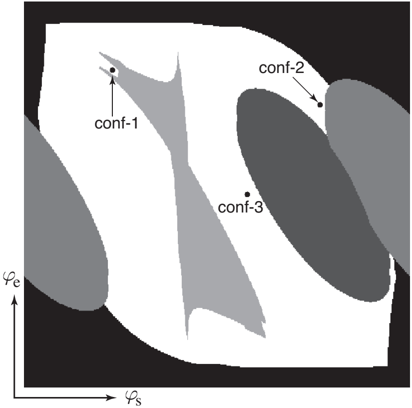


(Source)

Case studies

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

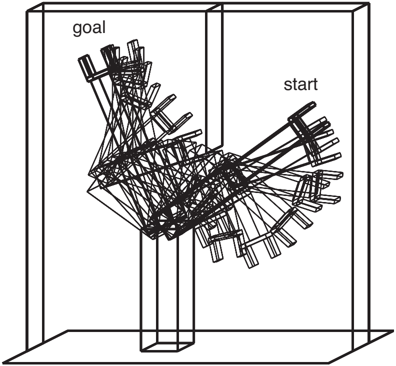


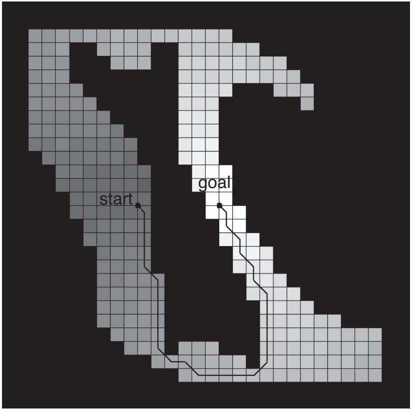


(Source)

Case studies

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





(Source)

Study case

Case study II: 9th Global Trajectory Optimization Competition (I)

GTOC: Global Trajectory Optimization Competition

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

GTOC 9: The Kesser Run

- 123 orbiting debris
- Remove debris
- Design multiple missions

(Video) (Suggested video)

Study case

Case study III: Mars orbital insertion

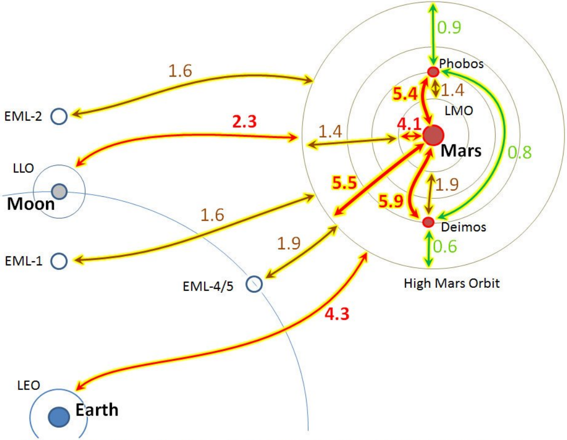


Chart by Richard Penn CC-BY, data from David Hollister hopsblog-hop.blogspot.co.uk