Methods of Artificial Intelligence

1. Introduction to Local Search

Kai-Uwe Kühnberger, Nohayr Muhammad

Winter Term 2022/2023

November 4th, 2022

Today

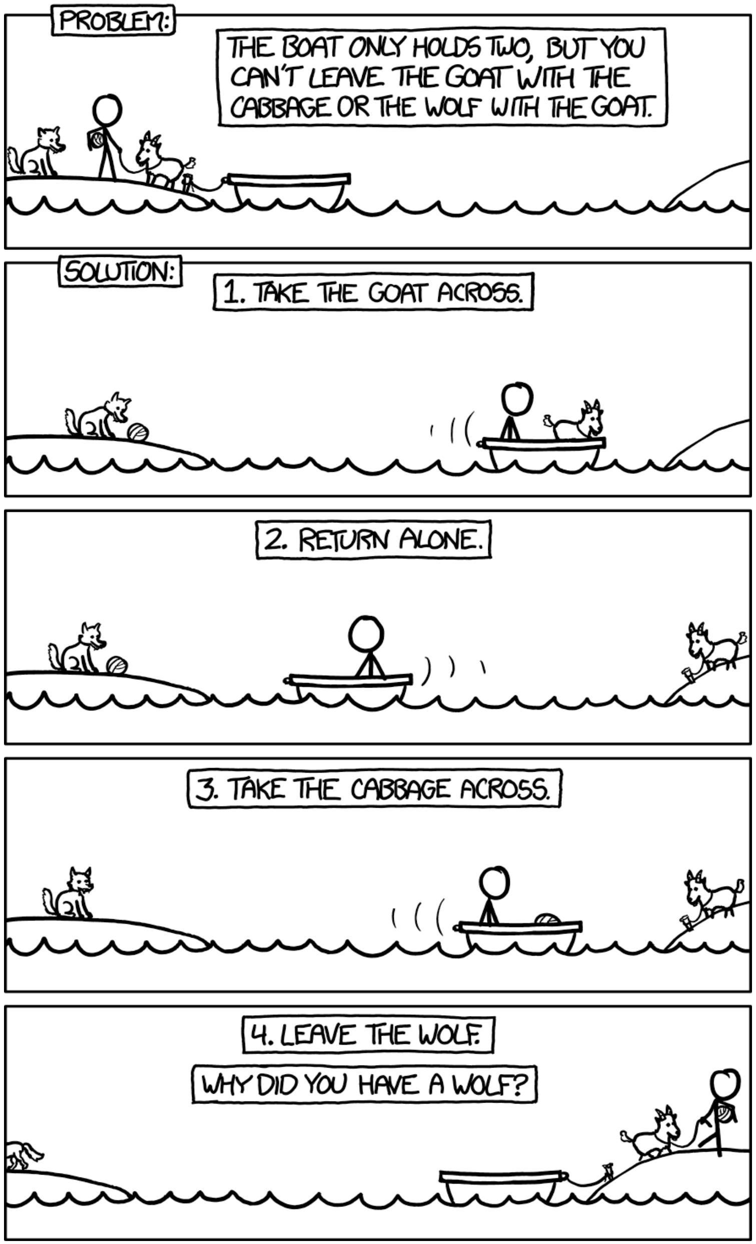
- Solving problems by searching
- Search problems
- (Classical) Search algorithms
- Searching in complex environments
- Local search main ideas

Solving Problems by Searching

A quick recap!

Problem-Solving agent

An agent that can look ahead to find a sequence of actions that will eventually achieve its goal.



https://xkcd.com/1134/

A four-phase problem-solving process:

- **Goal formulation**
  Goals organize behaviour by limiting the objectives and hence the actions to be considered.
- **Problem formulation**
  A description of the states and actions necessary to reach the goal—an abstract model of the relevant part of the world.
- **Search**
  The simulation of sequences of actions in the model, searching until a sequence of actions that reaches the goal is found, if any.
- **Execution**
  The agent can now execute the actions in the solution, one at a time.

Search problems examples

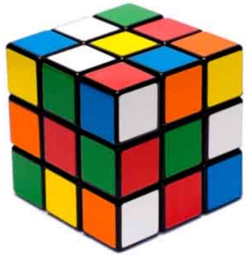
Route-planning

- Goal formulation
  Go from starting point A to point B. Possibly with (shortest, fastest, gas saving, …) path.
- Problem formulation
  - *States: Locations (e.g. cities), time, ...*
  - *Actions: Drive left, right, …*
  - *Possibly: A cost assigned to actions (time, distance, fuel consumption, …)*

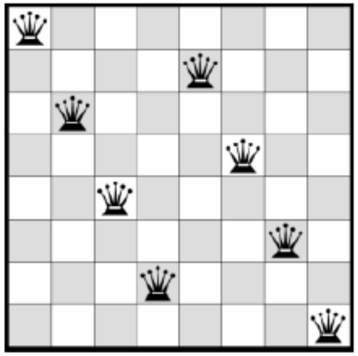
Solving puzzles

- Goal formulation
  Starting from certain, possibly random, configuration to reach the desired configuration.
  Possibly with least number of moves.
- Problem formulation
  - *States: Puzzle configuration*
  - *Actions: Move tile, add number,…*
  - *Possibly: A cost assigned to actions (time, number of moves, …)*

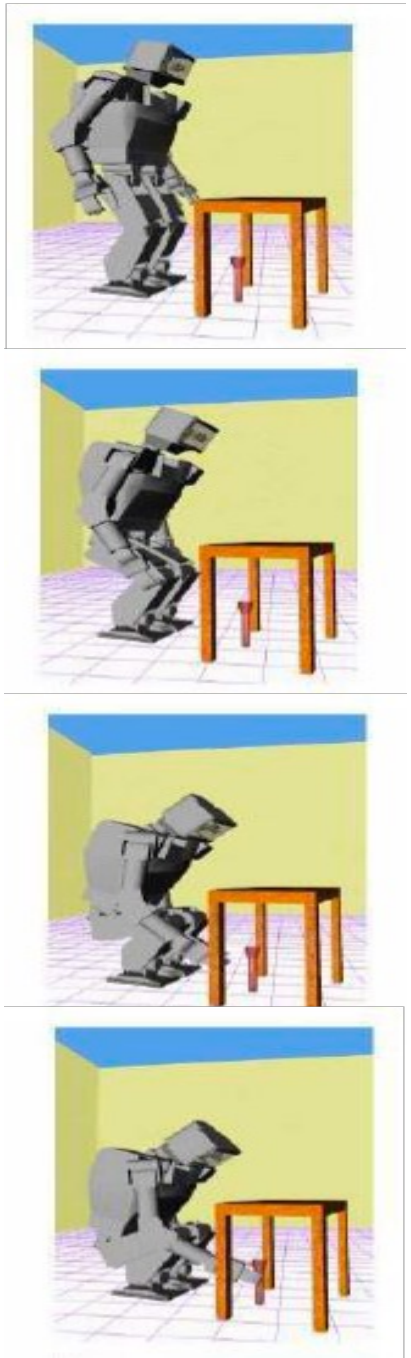
5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9



8		6
5	4	7
2	3	1



Robot motion planning



- Goal formulation
  Perform desired task, e.g. pick up something. Possibly in the (fastest, safest, energy saving, …) manner.
- Problem formulation
  - *States: Position of robot parts*
  - *Actions: Translate, rotate parts*
  - *Possibly: A cost assigned to actions (time, risk factor, energy consumption, …)*

Definition: Search Problem

- A search problem is defined as a 4-tuple $< S, R, s_0, G >$ with
  - The search space (state space) S
  - A successor relation (transition relation) $R \subset S \times S$
  - An initial state $s_0 \in S$
  - A goal predicate $G : S \rightarrow \{true, false\}$ which terminates the search

Also, a possible transition cost.

Definition: Solution

A solution to such a search problem is a sequence of states

$$(s_0, \ldots, s_n),$$

starting with initial state

$$s_0$$

with adjacent states in the transition relation

$$(s_i, s_{i+1}) \in R \text{ for } i = 0, \ldots, n-1$$

and ending at a goal state or

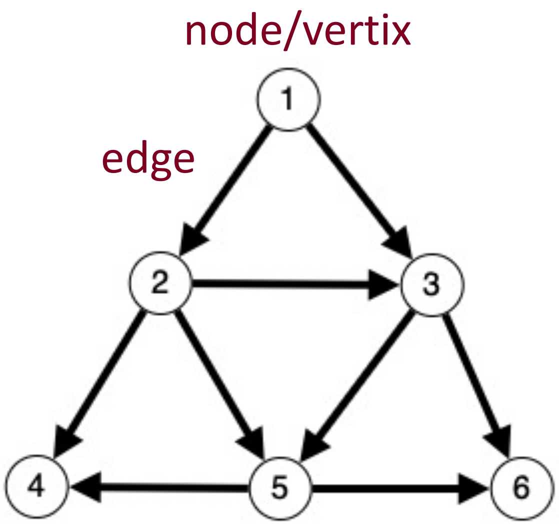
$$G(s_n) = \text{true}$$

A general search "algorithm"

A search algorithm takes a search problem as input and returns a solution, or an indication of failure.
This can be done by creating a search tree over the state-space graph, forming various paths from the initial state, trying to find a path that reaches a goal state.

State-Space Graph

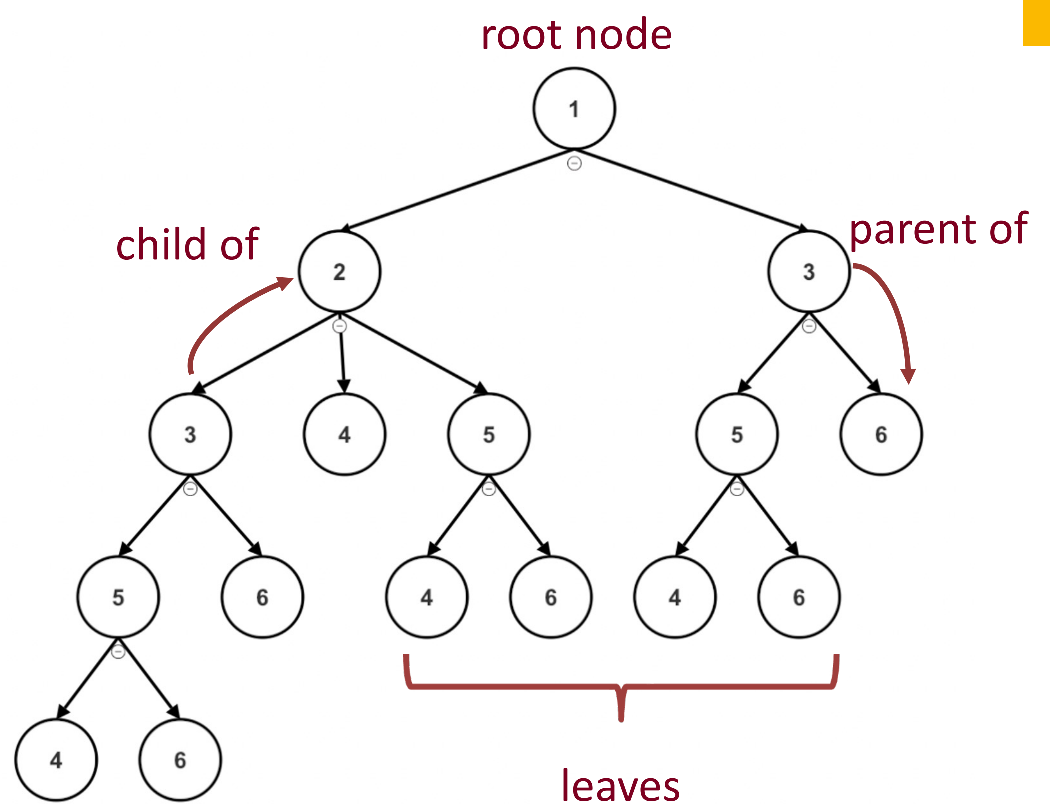
The state space can be represented as a graph in which the vertices are states and the directed edges between them are actions.



Directed graph

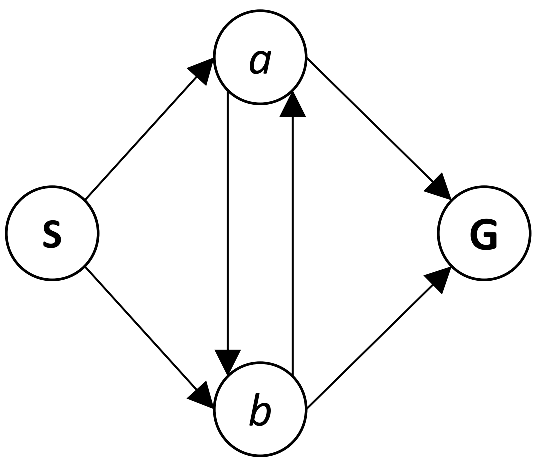
Search Tree

- Each node in the search tree corresponds to a state in the state space.
- The edges in the search tree correspond to actions.
- The root of the tree corresponds to the initial state of the problem.

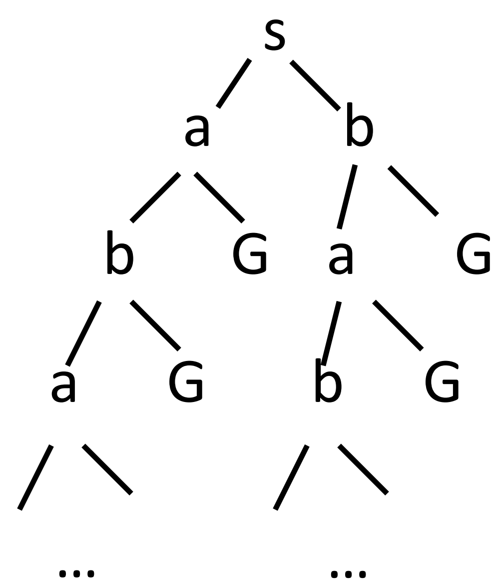


State Graph vs. Search Tree

State space graph



Search tree



UNIVERSITÄT OSNABRÜCK

A rough sketch of a general search algorithm:

- Generate a search tree:
  - Start with s_0 as root.
  - Add successors to a leaf-state s_i for all transitions $(s_i , s_j) \in R$
- A path from the root s_0 to a node s in the tree with G(s) = true is a solution
- Traverse the search tree somehow to collect the solutions

There are different ways to traverse somehow → Different search algorithms.

Classical Search

- Uninformed (blind)
  - Random search
  - Systematic search
    - Depth-first search (DFS)
    - Breadth-first search (BFS)
    - Variations
      - Uniform-cost search
      - Depth-limited search
  - Features
    - no information about the path cost from the current state to the goal state

- Informed (heuristic)
  - Systematic search
    - Greedy best-first search
    - A* search
  - Features
    - heuristic information (estimate) about the path cost from current state to goal state

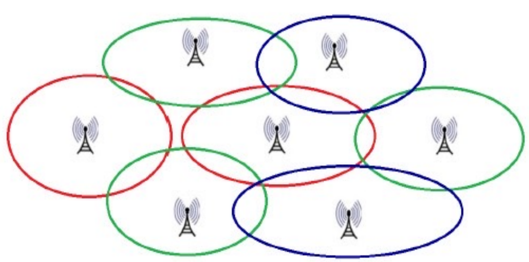
From Classical Search to Local Search

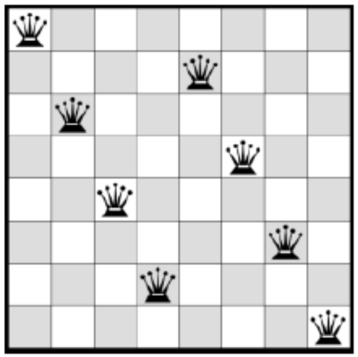
Classical Search

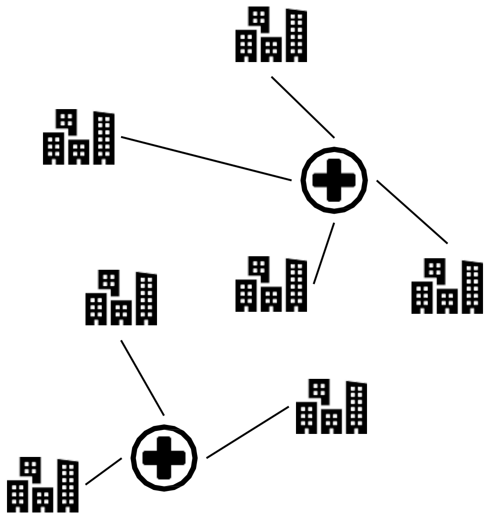
- find (optimal) sequence of actions — Sequence might be irrelevant
- that leads from initial state — Might not exist
- to a goal state — Might be hard to test

Local Search

- find „nearly“ feasible/optimal state







Local Search Applications

Local Search is applied if search spaces are too big for exhaustive search.

Typical problem types are:

- Constraint Satisfaction Problems
- (Discrete) Optimization Problems
  - Production planning (maximize profit)
  - Scheduling (minimize conflicts)
  - Machine Learning (minimize classification error)

States and Neighborhoods

- States often correspond to variable assignments

X1	X2	X3	X4
0	1	1	0

- neighborhood determines which assignments can be reached from given assignment

X1	X2	X3	X4
0	1	1	0

X1	X2	X3	X4
1	1	1	0

X1	X2	X3	X4
0	0	1	0

X1	X2	X3	X4
0	1	0	0

X1	X2	X3	X4
0	1	1	1

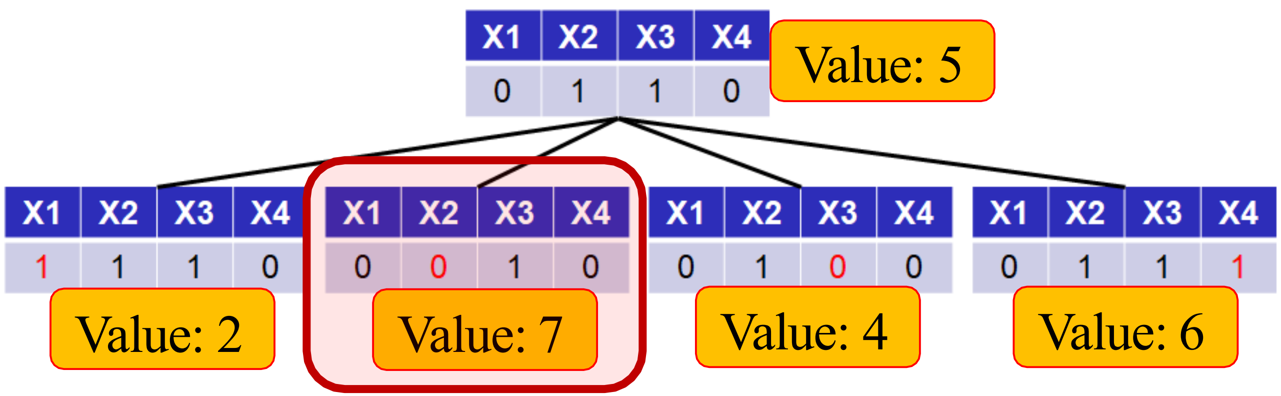
Local Search Main Ideas

Rough Idea

- start from some initial assignment (e.g. random choice)

X1	X2	X3	X4
0	1	1	0

- select a "promising" neighbor of the current assignment



- continue selecting neighbors until we cannot improve anymore

Solving local search problems

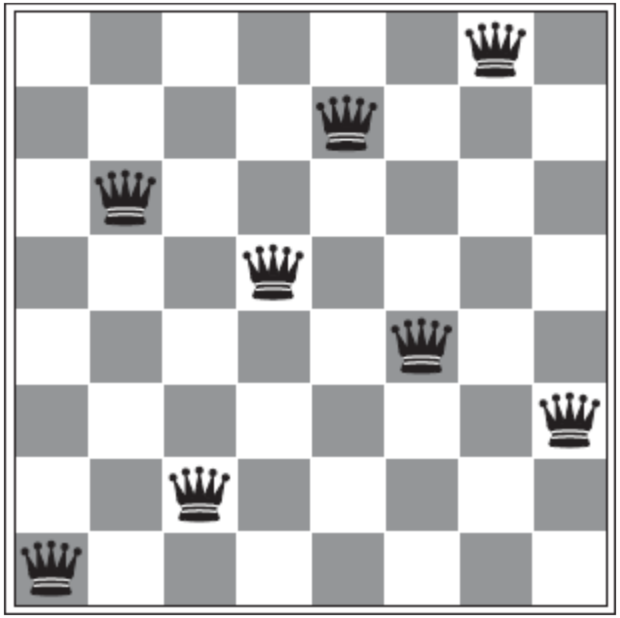
1. Define state space
2. Define neighborhood
3. Define objective function

   (minimize f(x) by maximizing –f(x))
4. Apply local search algorithm

Local Search Example

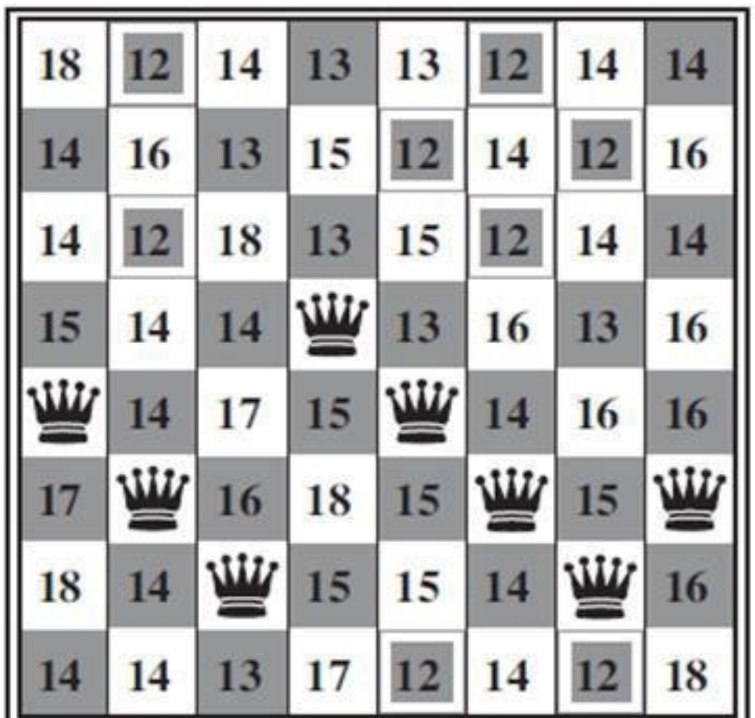
How can we apply local search to typical problems?

N Queens Problem



1. State-Space: board configurations with one queen per column (tuple (8,3,7,4,2,5,1,6) corresponds to board configuration above)
2. Neighborhood: configurations that can be obtained by moving a single queen to another field in the same column

3. Objective Function: number of pairs of queens that can attack each other directly or indirectly *(maximize negative objective function)*



- Queen in column 1 can directly attack queen in column 2
- Queen in column 1 can indirectly attack queen in column 3
- State has value 17
- Numbers show values of neighbors

4. Apply a local search algorithm

Next time…

We will discuss the following local search algorithms:

- Hill-Climbing
- Simulated Annealing
- Local Beam Search
- Genetic Algorithms

Summary

- Problem-solving by searching
- Formulating search problems
- Difference between classical and local search algorithms
- If we are only interested in a solution and not in the solution path, Local Search can be better suited than Classical Search
- this is typically the case for optimization and constraint satisfaction problems

Slides are, mainly, Dr. Tobias Thelen slides.
Some presented ideas are adapted from other courses like CS188 at UC Berkeley.

Further Readings

Lecture is mainly based on:

Russell, S., Norvig, P. Artificial Intelligence - A modern approach. Pearson Education: 2010.

More details on presented (and similar algorithms) can be found in:

Burke, E. K., & Kendall, G. Search methodologies - Introductory Tutorials in Optimization and Decision Support Techniques. Springer US: 2014.