Heuristics, local search

AI3

Mária Markošová

What we know

1. Uninformed search, BFS, DFS, uniform cost search, complexity, properties.
2. Informed search, heuristics.
3. Greedy search, A*, complexity, properties.
4. Graphsearch, treesearch implementation, differences.

Poznámka ku dvom spôsobom implementácie Graphsearch algoritmu.

Pôvodné **8)** Rozvinieme bod *n* , vytvoríme množinu nasledovníkov M, ktorí ale nie sú **v zozname *CLOSED***. Vytvoríme reprezentáciu hrán medzi *n* a členmi M.

Vo chvíli, **keď pridávam uzol do Open, kontrolujem**, či už nie je v Closed zozname. Ak je, preskočím ho.

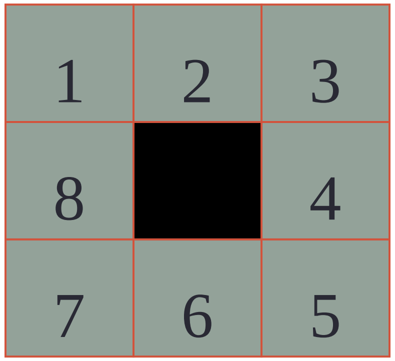
Zjednodušené: Nekontrolujem či je uzol v Closed pri pridávaní do Open, ale pridám tam všetky uzly. Tak sa napr. uzol G môže vyskytnúť v Open viackrát. Keď ho spracujem prvýkrát, pridám ho do Closed. Keď sa ho pokúšam spracovať znovu (pri ďalšom výskyte v Open), skontrolujem, či nie je v Closed. Ak je, preskočím ho. **Closed kontrolujem pri vyberaní** uzla z Open.

Zložitosť algoritmu sa nemení, ale pri druhej implementácii menej krát kontrolujem zoznam Closed. Prečo? Pretože pri druhej implementácii sa k mnohým bodom z Open algoritmus ani nedostane, lebo nájde riešenie skôr.

Outline

1. Heuristics in more details.
2. Local search - principle
3. Local search algorithms (hill climbing, genetic algorithm, simulated annealing)

Heuristic functions: **(8 puzzle example)**



Typical solution for randomly chosen initial state : 22 steps, $d \approx 22$

Typical branching: $b \approx 3$

Searching all states till the depth 22:

goes through $n \approx 3^{22}$ states

Number of different **solvable** states for 8 puzzle: 181 440= 9!/2

Number of different states for 15 puzzle: 10^{13}

$\downarrow$

Good heuristics is important

Possible heuristics h always reflects the problem, but the same way relaxes some constraints. Admissibility is a necessary property.

h: 1. - number of tiles in a bad positions h_1

2. - sum of horizontal and vertical distances from the good position of the tile (manhattan distance)- h_2

$\downarrow$

Both are admissible, that means they do not overestimate the real distance to the goal

Quality of heuristics

Effective branching factor ***b****

$$N + 1 = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d$$

Equation for the b calculation *b**

N – number of nodes expanded by *A** with a given heuristics

d – depth of the tree

Good heuristic gives: $b^* \approx 1$

8 puzzle comparison

	Cena cesty			**Efektívny faktor vetvenia**		
d	**IDS**	**A*** (h_1)	**A*** (h_2)	**IDS**	**A*** (h_1)	**A*** (h_2)
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	3644035	227	73	2.78	1.42	1.24

IDS: iterative deepening search, variant of DFS

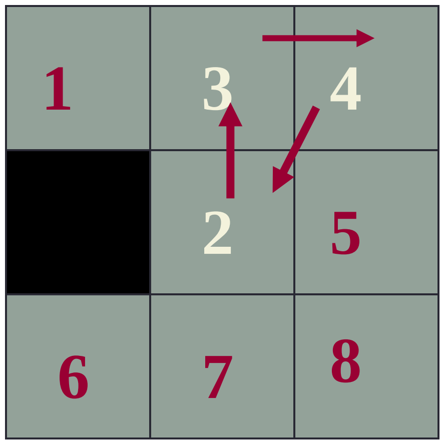
How to find an admissible heuristics by the problem relaxation

-problem with less constraints $\longrightarrow$ ***relaxed problem***

Example:

Let us admit, that the tile in the 8-puzzle can "jump" at an arbitrary position. h_1, the number of tiles in a bad position, gives therefore an exact cost for this relaxed variant, gives an exact cost of the most effective solution.

How we relaxed the problem in the case of the Romanian map (strait line distance heuristics)?



Number of tiles in a bad position : 3

And what about ***h(2)*** ? How did we relaxed the problem here?

How to choose heuristics if we have more then one admissible ones?

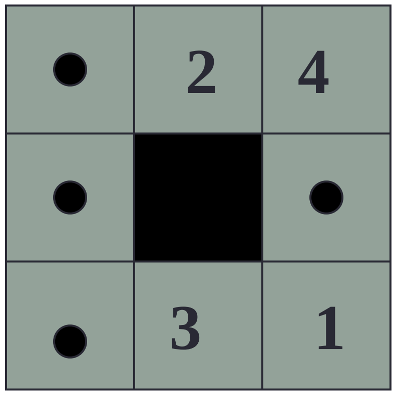
1. Choose always the **dominating** one (h_2, in our case), the one for which $h_2(n) \geq h_1(n)$ holds for all nodes n.
2. If there is no such, create a composed heuristic

$$h(n) = \max\{h_1(n),\ h_2(n), \ldots h_m(n)\}$$

This uses always the best one from the set for each node.

How to find heuristics by solving a subproblem?

(subproblem – put 1,2,3,4 tiles on a correct positions)



-subproblem to 8 puzzle

a) find an optimal solution
b) it's cost is an estimate from below to the total solution cost
c) put it into the database
d) use as a heuristics, in a case if the subproblem appears during the real problem solution

Algorithms of the cyclic improvement

They solve optimization problems. In these problems the state itself is important, the path is not a part of the solution.

Examples: *n* queen problem- how to place *n* queens on a *n x n* chessboard, how to place integrated circuits on a board etc.

Fitness or objective, evaluation function of the intermediate state is evaluated, the goal is to find a state having the optimum of the fitness, objective function.

The basic principle of the local search (cyclic improvement of the state)

Local search:

1. We start from an initial state, which is evaluated with a help of some **evaluation function**.
2. Perturbing the current state, neighbor states are generated and evaluated. The best one of them is then chosen according the evaluation. The path to the best state is forgotten. The best state is then either accepted according some known criteria, or:
3. Current best state is perturbed and the algorithm continues from 2.

These algorithms are appropriate for the optimization problems solution.

Optimization problem

To solve an optimization problem means to find an optimum (maximum or minimum of some function). If this function is complex, unknown analytically, we can do that numericaly with a help of optimization algorithms.

In our case we seek an optimum of the evaluation function in a space of states.

Example: *n*-queen problem

- Put *n* queens on $n \times n$ chessboard. None of them can attack themselves.

Let ***h*** be the number of pairs of queens that can attack each other. Evaluation function gives each state its value *h*. We look for the state with *h=0*, minimum.

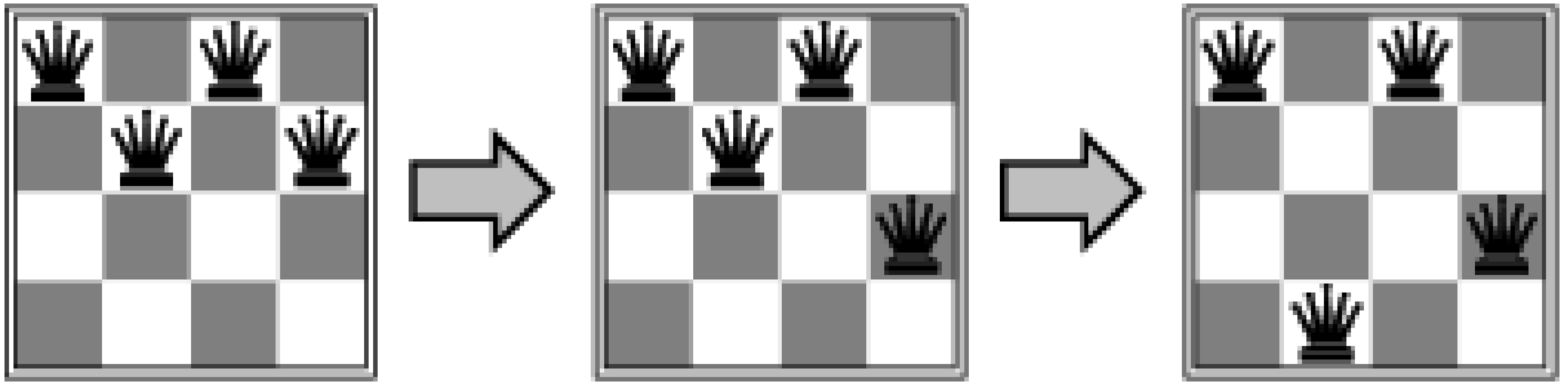
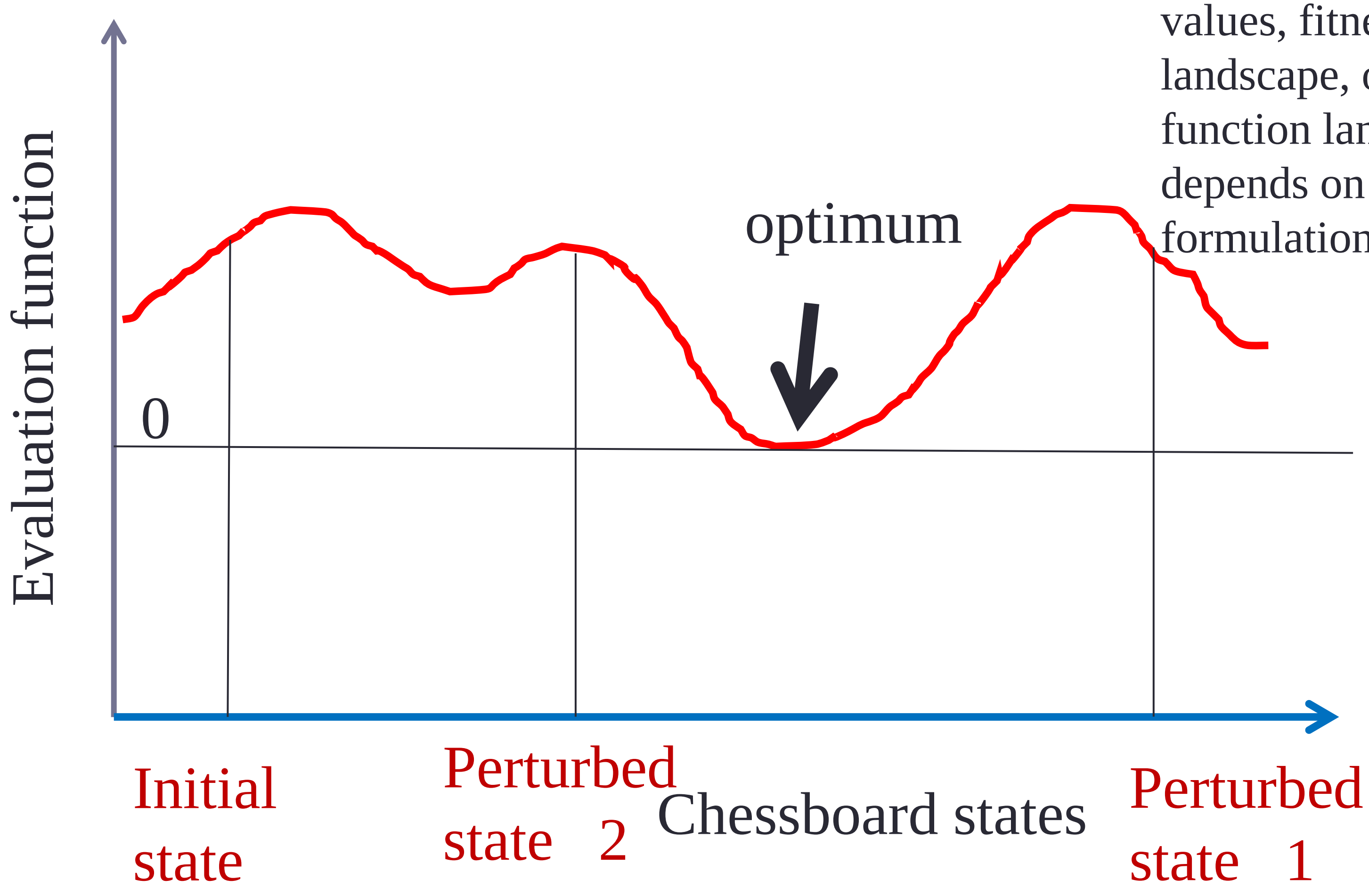
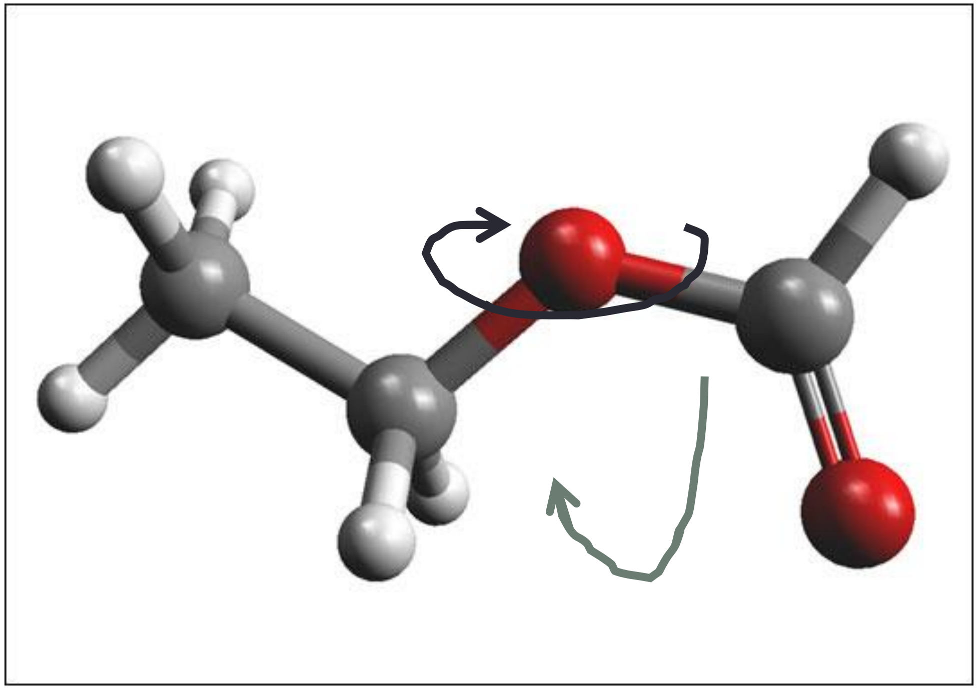


Illustration of the problem (1d)

Red curve is an evaluation function, it is called landscape of values, fitness landscape, objective function landscape. It depends on the problem formulation.

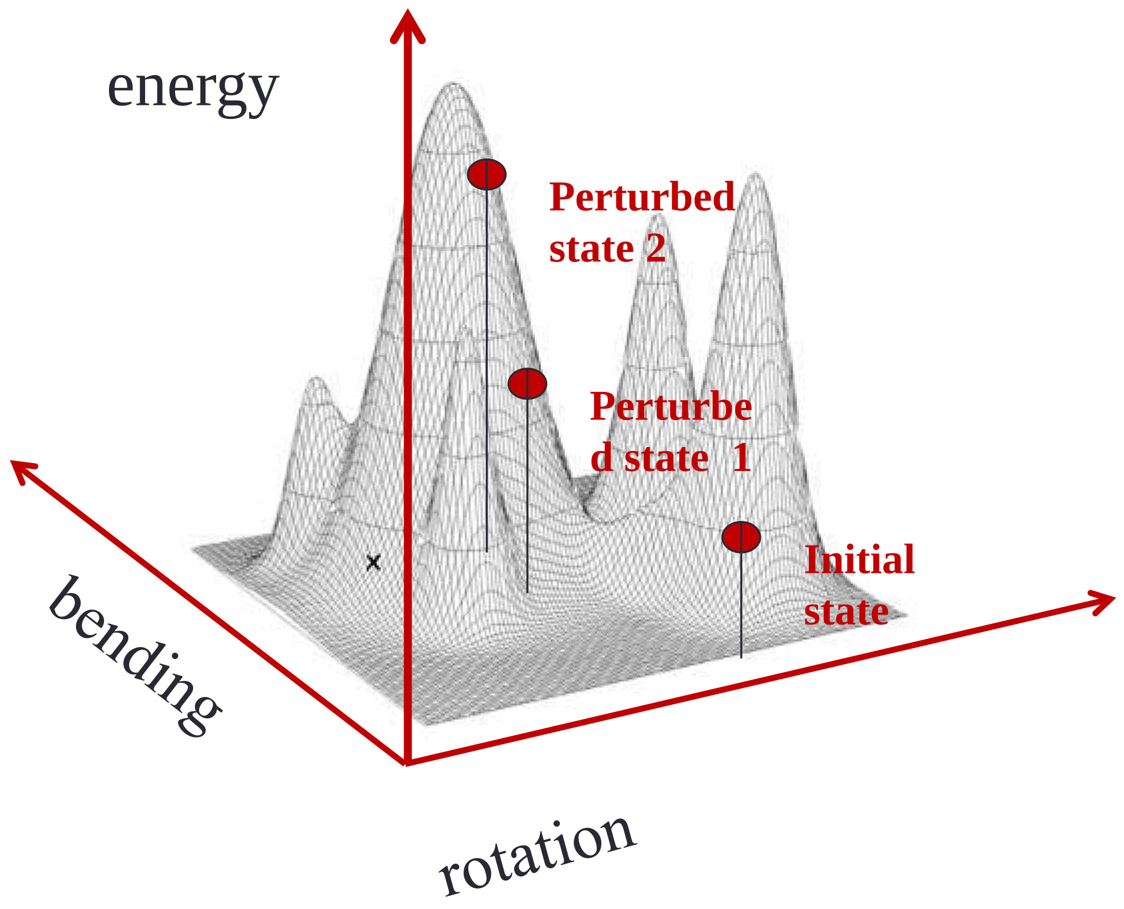


Example: Seeking for an optimal structure of an organic chemical. Optimal means, a structure for which a maximum energy is necessary to disintegrate it.

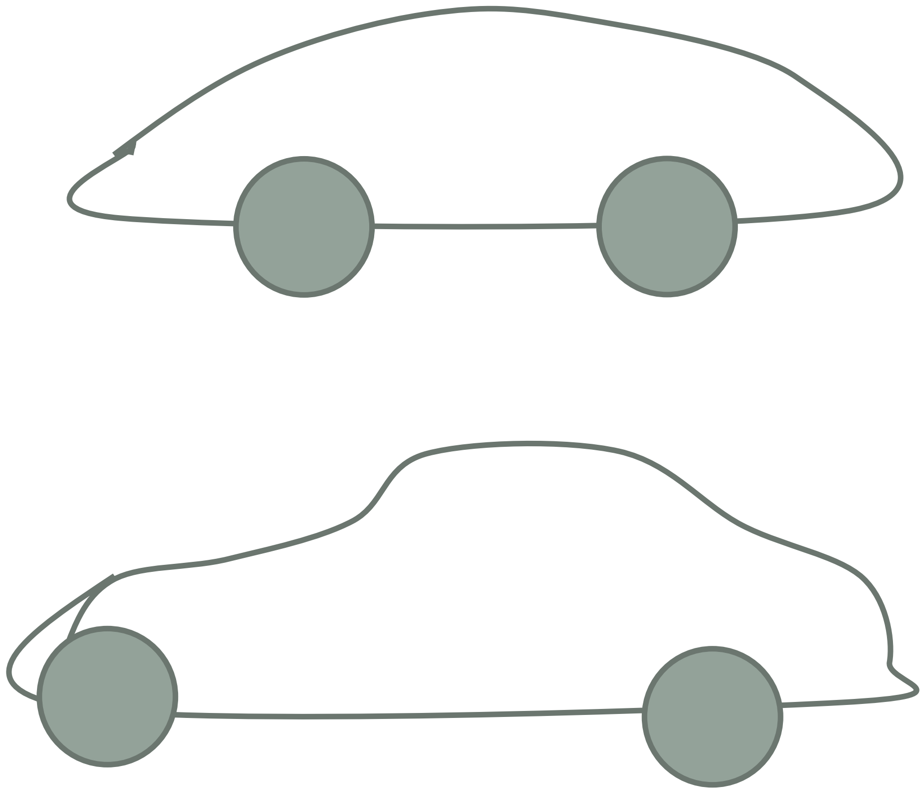


Constrains: Rotating and bending the bonds is possible in certain interval.

2d illustration of the problem



Example: Seeking for the best shape of the car. Best in a sense that it causes the smallest air resistance.



Evaluation of the shape (state) quality is due to the fact what air resistance and friction gives the shape in question. Another properties can influence that as well. Changing shape or other properties, such as lacquer one can change the air resistance and friction. This can be measured in the air chamber.

Algoritmhms of local search

Paradigmatic example: *n* queens.

Generate and test algorithm

1. Generate several random solutions (placements of *n* queens) on *n* x *n* chessboard.
2. Evaluate each state with an evaluation function (sum all pairs of the attacking queens)
3. Forget all solutions except of the best one until now (the one with the least number of the attacking queens).
4. Repeat from 1 until the stop condition is met (for example number of maximal iterations 1.
5. The solution is the overall best state.

Hill climbing algorithmus

On the state surface we always go in a direction which makes our local state better:

Traps: 1. local maxima

2. plateau

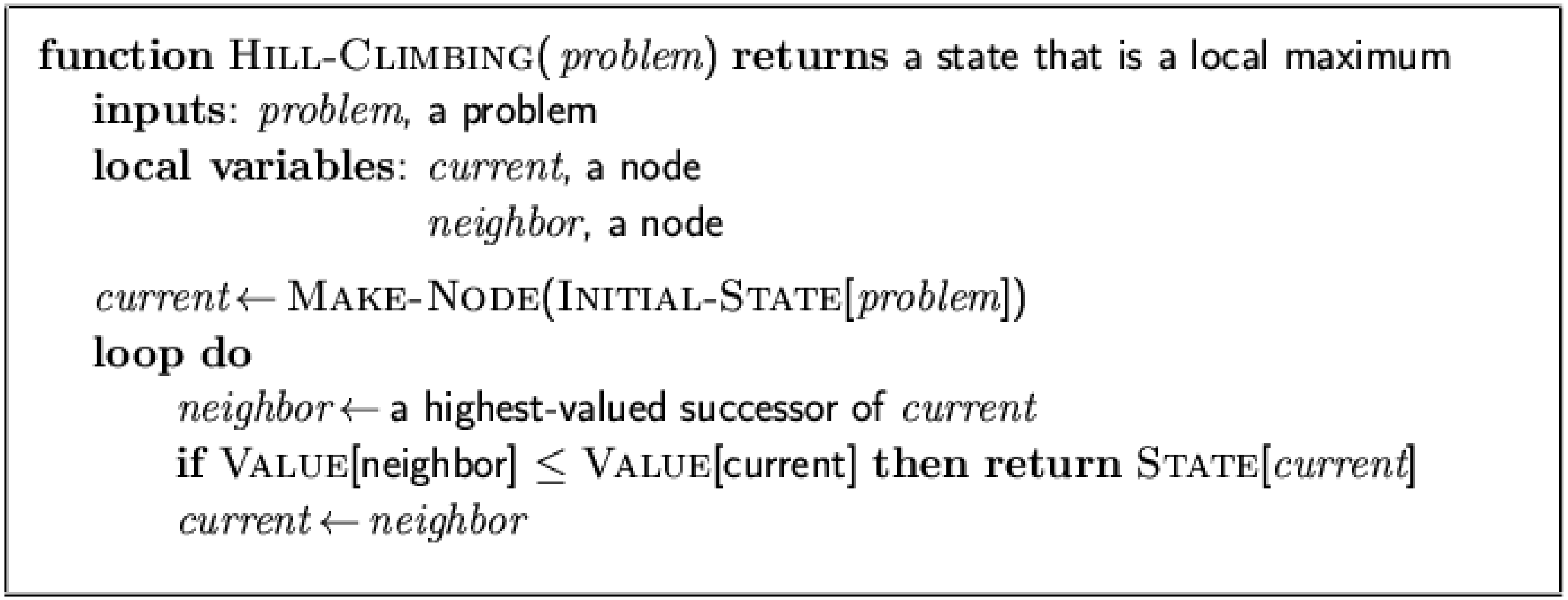
3. ridge

Algorithm likes to cycle; good for state surfaces with small amount of extrema.

How to awoid cycling? Multiple starts from different initial states

Hill-climbing search (Russell, Norwig)

- "Like climbing Everest in thick fog with amnesia„

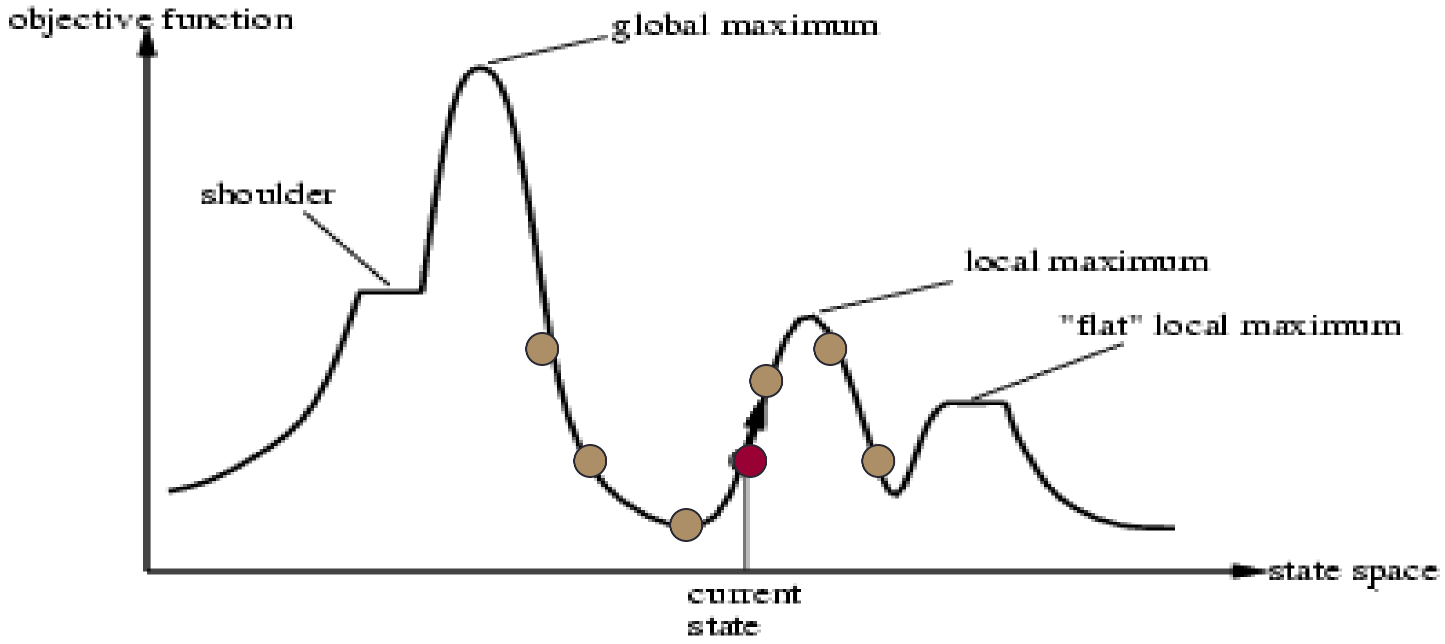


```
function HILL-CLIMBING(problem) returns a state that is a local maximum
    inputs: problem, a problem
    local variables: current, a node
                     neighbor, a node

    current ← MAKE-NODE(INITIAL-STATE[problem])
    loop do
        neighbor ← a highest-valued successor of current
        if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
        current ← neighbor
```

Hill-climbing search

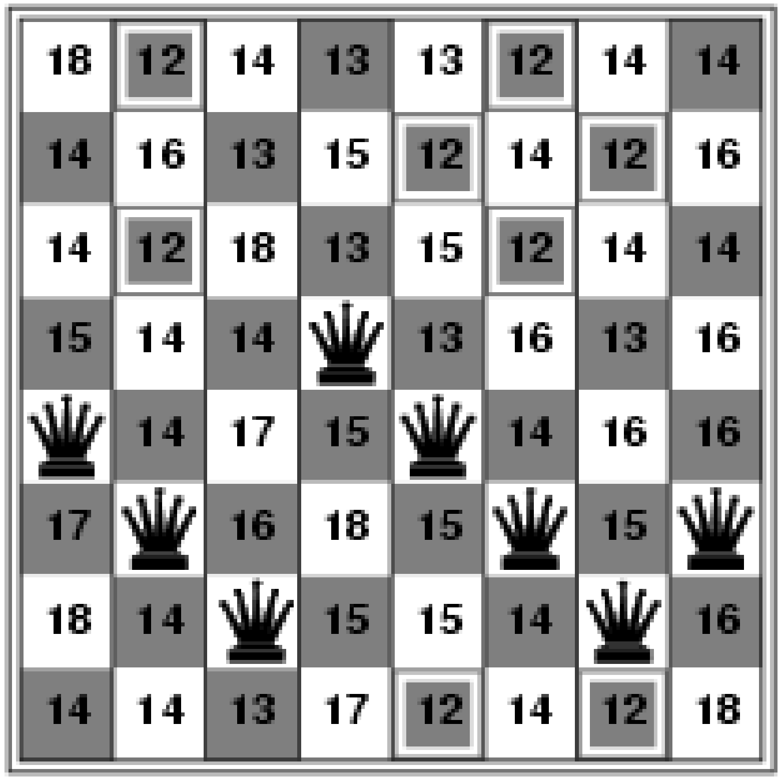
- Problem: solution depends on the initial state, gets often stuck in a local optimum.



Hill climbing

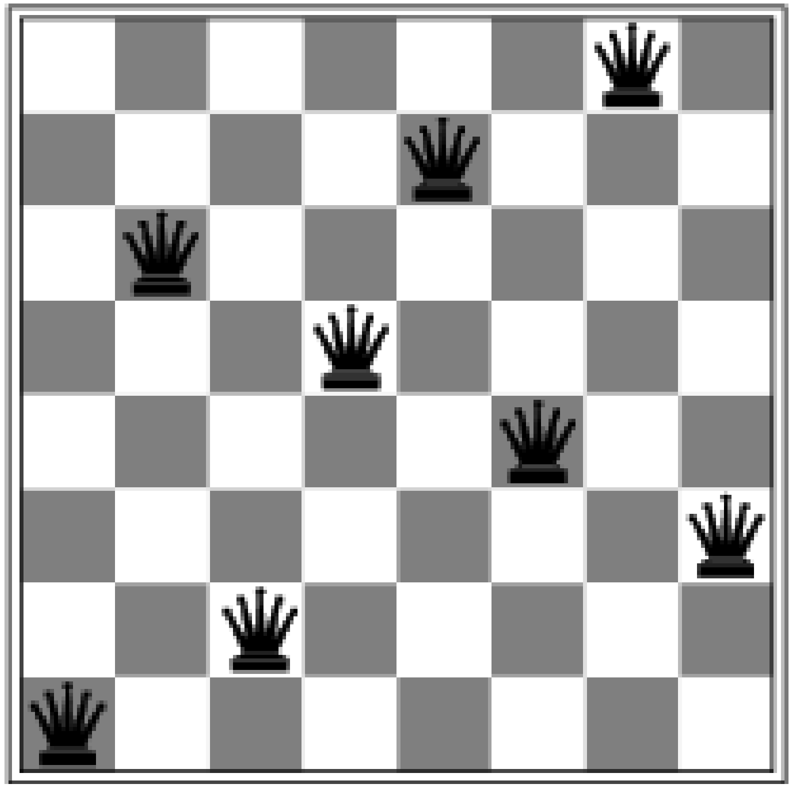
1. Choose an initial state and evaluate it.
2. From the initial state generate *m* new states by perturbation of the initial state. Evaluate all of them
3. In the most simple hill climbing only the best state is chosen. The other ones are forgotten. The best state will be a new initial state.
4. Repeat from 2.
5. Stop according the stopping criterion.

Hill-climbing search: 8-queens problem



- *h* = number of pairs of queens that are attacking each other, either directly or indirectly
- *h = 17* for the above state

Hill-climbing search: 8-queens problem



A global minimum with *h = 0*

Hill climbing algorithm is

-***noncomplete***, does not find the solution always, can get stuck

how to avoid this: generate several initial states and run hill climbing separately from each of them

$\downarrow$

if the number of restarts is infinite, algorithmus

finds a solution with probability close o 1

Genetic algorithmus

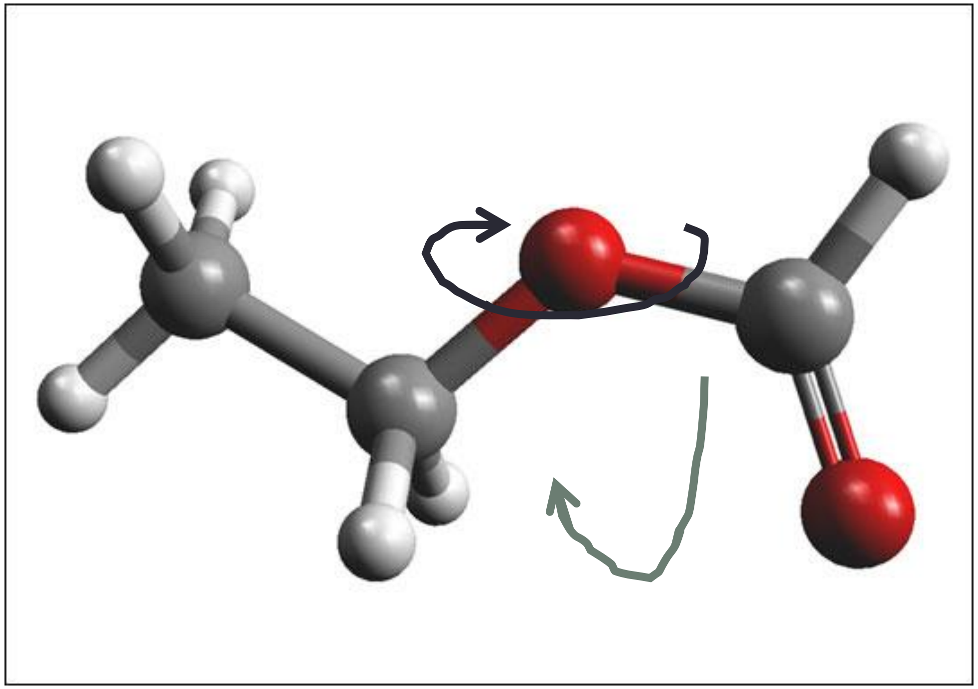
Inspired by the natural evolution, Darwinian evolutionary paradigma

1. Individual in the population is represented by a string of symbols, chromosomes, in which the genetic information is encoded. Each individual has certain ability to survive, according which we can evaluate him. Let us take the most simple case, individual is represented by exactly one chromosome.
2. Population is a set of individuals represented by the vector of symbols / chromosome
3. During the reproduction the parents are selected by a natural selection (the better the individual, the more probable it is to be selected as a parent).
4. Then the parent genetic information is recombined (crossover) and mutated.

Genetic algoritmus

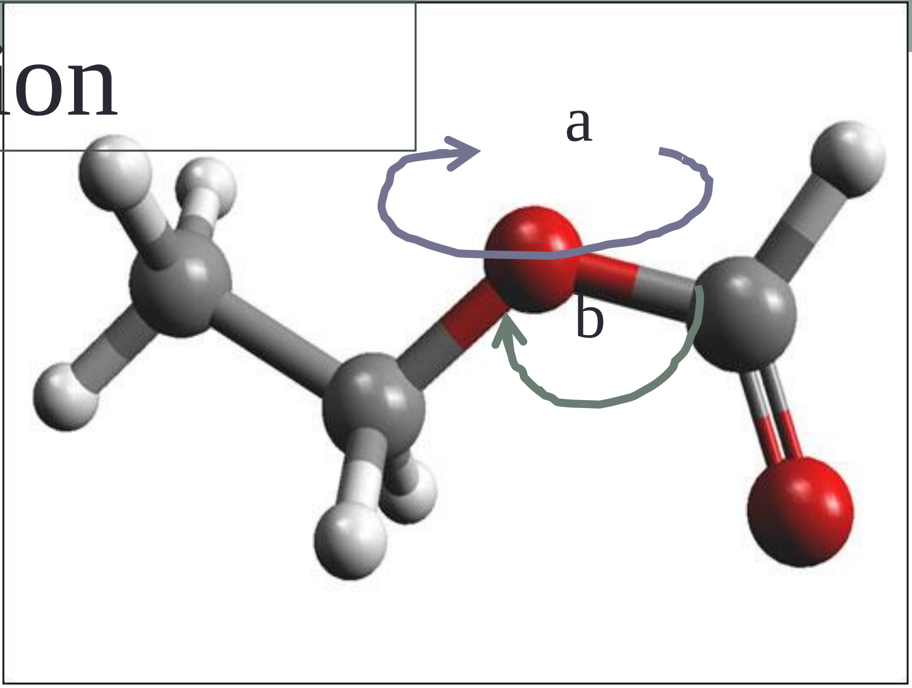
1. Starts with *k* randomly generated states (population).
2. Each state is represented by the “chromosome”.
3. Evaluation of the population individuals.
4. New state is created with a help of the reproduction process. Parent states for the reproduction are chosen with respect of their value. The greater the value, the greater is the probability to be chosen as a parent.
5. Reproduction= crossover +mutation.
6. The reproduction process creates new individuals which replace the old ones either entirely or partly.

Example: ; we are seeking its secondary structure which needs maximal energy to be destroyed. The structure can be changed by bending and rotating some of the bonds.



Constrain: Rotation and bending is possible only in some limits.

Population generation



Different structures	energy
x1=(a1,b1)	h1
x2=(a2,b2)	h2
.	
.	
.	
xn=(an,bn)	hn

Reproduction

Selection

Parent 1:	x10=(a10,b10)
Parent 2:	x16=(a16,b16)

Chosen with respect to the value of energy

Crossover

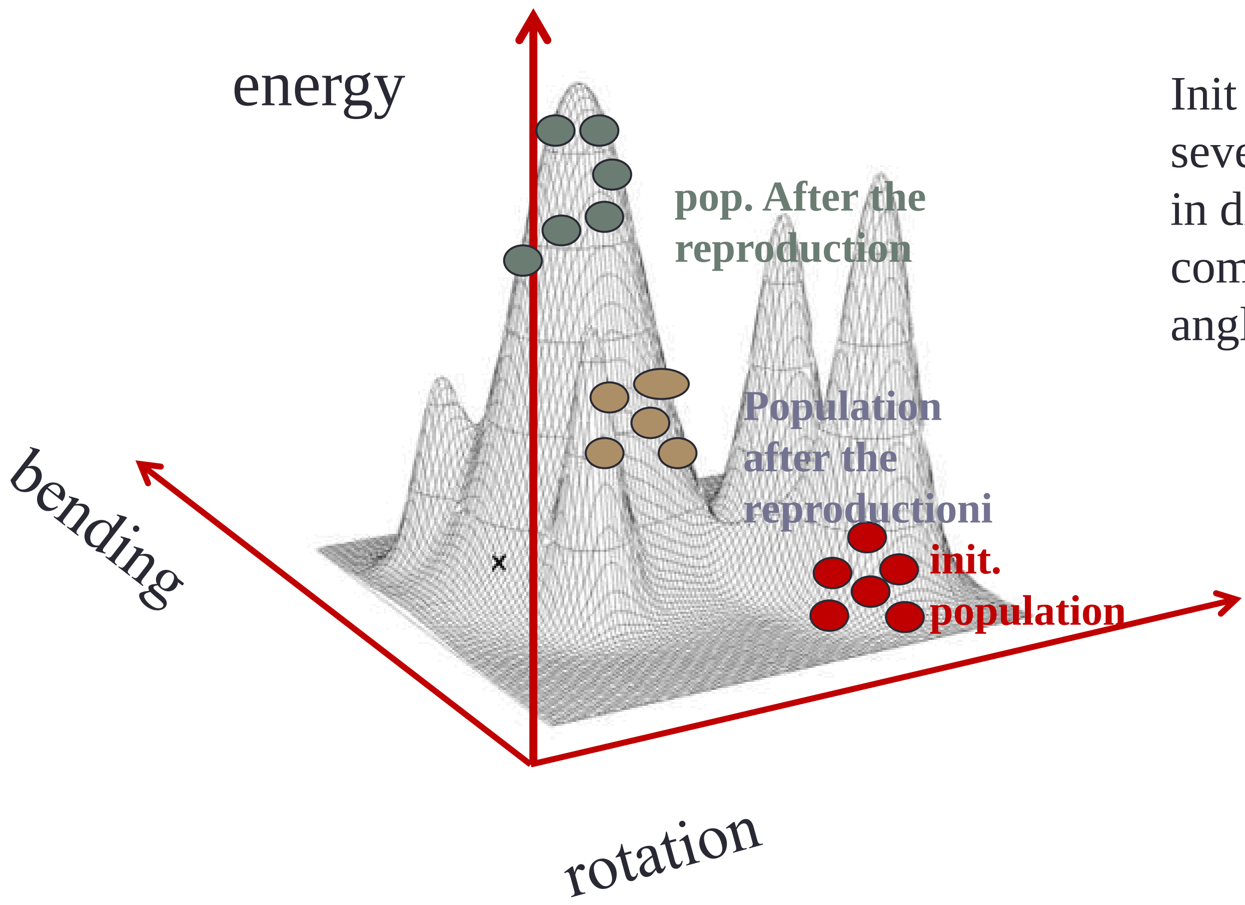
Successors after the crossover: xa1=(a16, b10)
xa2=(a10,b16)

Mutation

Succesor after the mutation xa11=(a16, b10+delta)

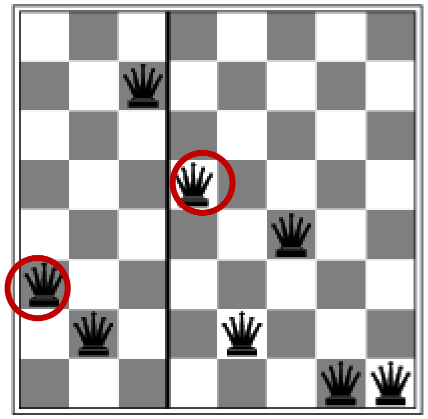
Reproduction process runs as many times as we need for the creation of enough new individuals

Ilustration in 2d


energy
pop. After the reproduction
Population after the reproductioni
init. population
bending
rotation

Init . population: several molecules in different states, combination of angles.

Another example: 8 queen problem, evaluation function
evaluate the number of queen which are attacking themselves



State is represented by the string of symbols (chromosom):

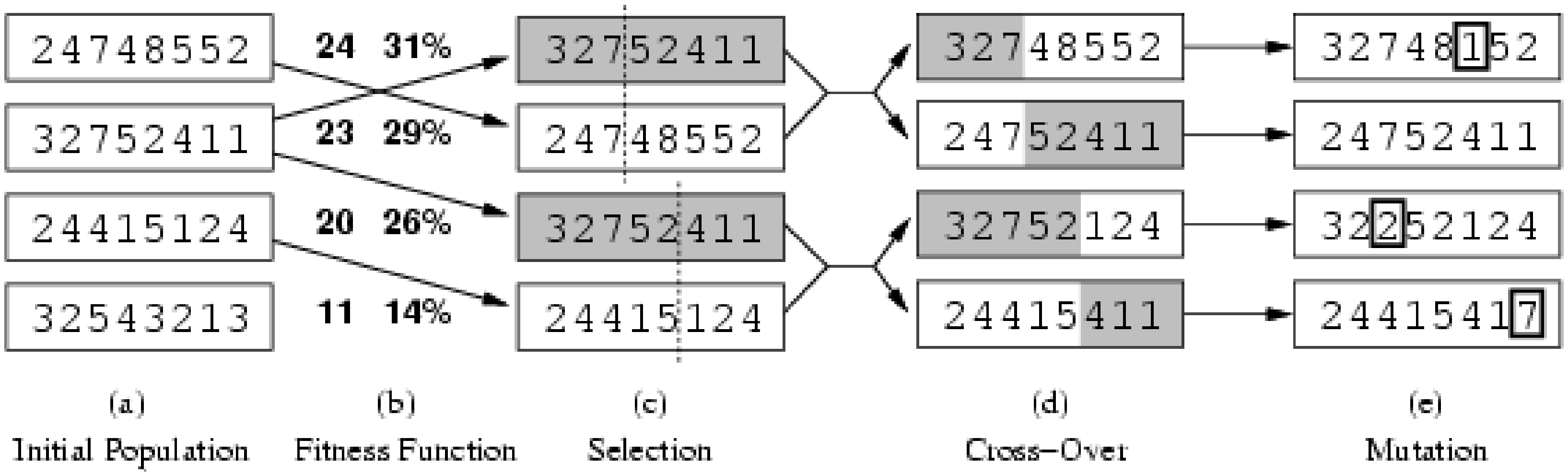
(3,2,7,5,2,4,1,1)

Tell me, looking at the picture, what these numbers mean.

Position in a string means a column, number denotes the number of row (from the player point of view)

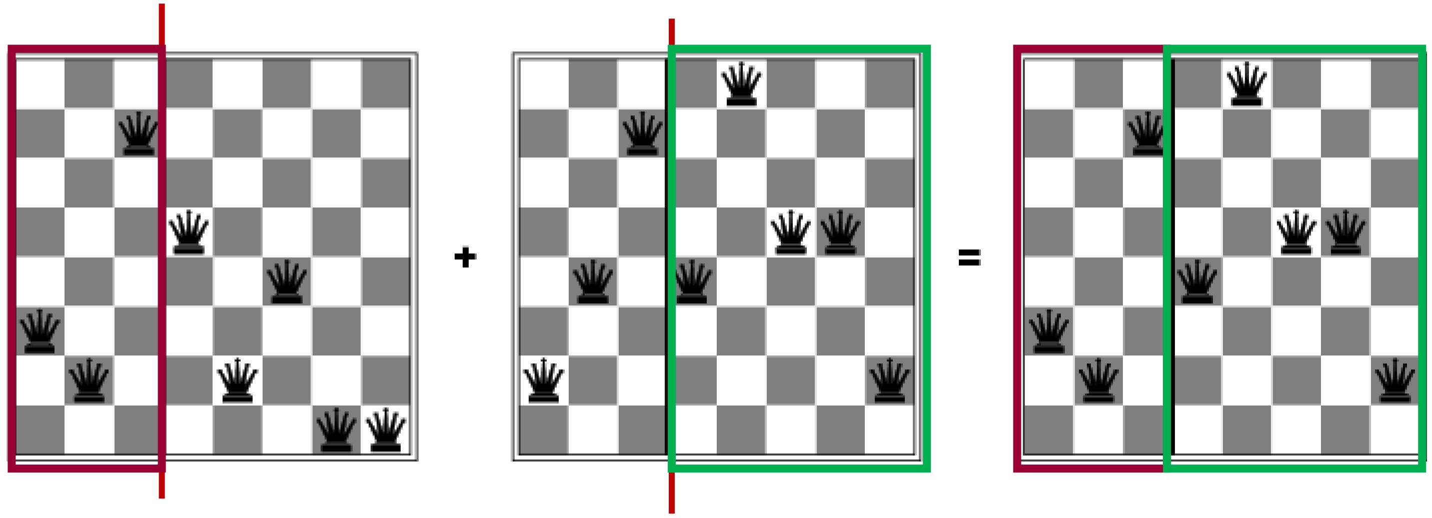
Genetic algoritmus)

8 queen problem, according Russel and   Norwig



Gives
evaluation of
the position

Genetic algorithm- crossover



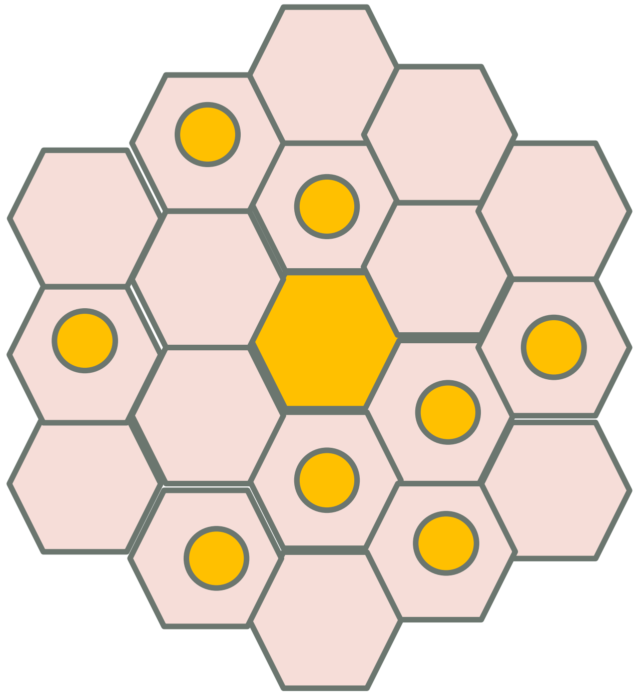
$$(0,3,0,4,6,1,(2,5),(7,8)) + (5,3,0,(6,7),(2,4),0,(1,7),0) = (5,3,0,(6,7),4,1,(2,7),0)$$

Genetic algorithm

- GA is not optimal, often finds local optimum, not a global one

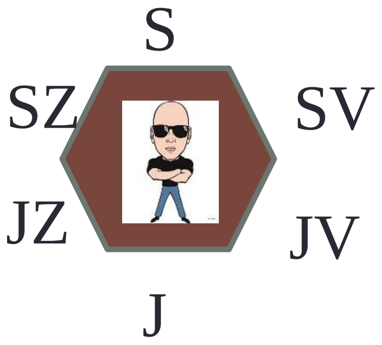
  Problems of GA / next example.

Example



Let us have a network of hexagonal rooms, each room has 6 doors. In some of them is a treasure.

In the middle room is a treasure finder. He can make 10 steps, each in six possible directions.



The finder task is to find such path, which visits as many treasure rooms as possible. Ideally all 10, but he can stop finding six.

Chromosome, represents individual ?

$$\vec{x} = (SZ, SV, J, SV, SZ, JZ, S, J, S, S)$$

Evaluation of the chromosome?

Number of rooms visited by the individual .

Population?

Set of individuals, chromosomes representing movement codes

Initial population

$$\vec{x_1} = (JZ, SV, JZ, S, JV, SZ, JZ, JV, JV, SV)$$
$$\vec{x_2} = (SZ, J, J, SV, SZ, JZ, S, J, S, S)$$
$$\vec{x_3} = (SV, JZ, S, JZ, SZ, JV, JV, JZ, SV, JV)$$

Stopping condition?

If we find an individual, chromosome, path which visits 6 treasure rooms.

Fitness value of individuals (due to the evaluation) is 3, 3, 2. Average fitness in an initial generation is 2.67 .

Selection: The greater the fitness the higher probability to be selected as a parent.

Selection probability in an initial population?

3/8, 3/8, 2/8

Selected parents

$$\overrightarrow{x_1} = (JZ, SV, JZ, S, JV, SZ, JZ, JV, JV, SV)$$
$$\overrightarrow{x_2} = (SZ, J, J, SV, SZ, JZ, S, J, S, S)$$

Crossover :

$$\overrightarrow{x_1} = (JZ, SV, JZ, S, JV, SZ, JZ, \boxed{JV, JV, SV})$$
$$\overrightarrow{x_2} = (SZ, J, J, SV, SZ, JZ, S, \boxed{J, S, S})$$

Successors after the crossover

$$\vec{y}_1 = \left(JZ, SV, JZ, S, JV, SZ, JZ, \boxed{J, S, S}\right)$$
$$\vec{y}_2 = \left(SZ, J, J, SV, SZ, JZ, S, \boxed{JV, JV, SV}\right)$$

Successors after the mutation

$$\vec{y}_1 = \left(JZ, SV, JZ, S, JV, SZ, JZ, S, S, S\right)$$
$$\vec{y}_2 = \left(SZ, J, J, SV, SZ, JZ, S, JV, JV, SV\right)$$

Fitness: 2
4

Replacement of the initial generation: From the initial and new chromosomes choose 3 with the highest evaluation value (fitness). If there is a chromosome with the value 6 (six visited treasure rooms), stop. We have a solution. If not, continue.

Space of chromosomes and searching space.

Space, set of chromosomes: is connected to the encoding of the problem parameters and algorithm able to work with this encoding.
Searching space: reflects the problem we solve

Decoding is sometimes necessary among these two spaces:

Example : Treasure world

Space (set) of chromosomes: Each chromosome is a string of direction symbols. Looking at such string we do not know the value of the string, how many treasure room the path encoded by the string visited. **We do not know the fitness of individual.**

$$\vec{x} = (SZ, SV, J, SV, \underset{\Big\downarrow \text{Decoding}}{SZ}, JZ, S, J, S, S)$$

Searching space: rooms in which the finder moves

Are we able to find such chromosome encoding, that both spaces are identical (we do not need decoding)?

1. Number the rooms.
2. Let the treasure is in rooms number 2, 4, 5, 11, 13, 15, 17, 19.
3. Chromosome is encoded as a vector of room numbers visited by the finder.

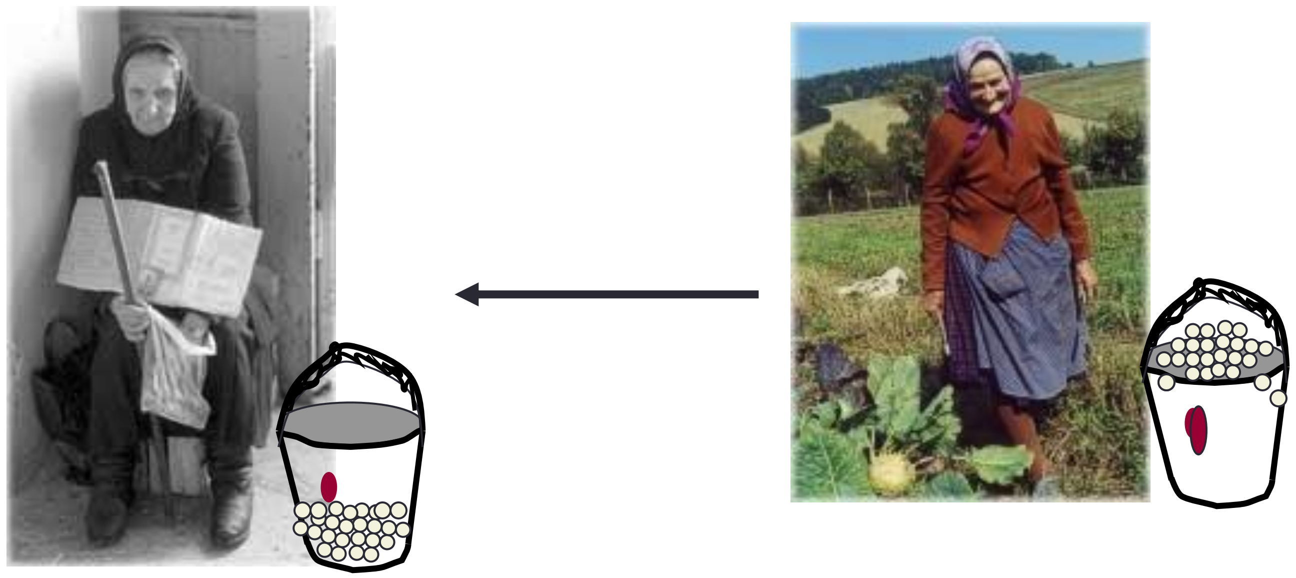
$$\vec{x} = (6,1,6,7,1,7,17,6,5,4)$$

4. In this representation the chromosome value is directly visible =3 (3 rooms with the treasure were visited).

But there is no free lunch! What problems can arise in a reproduction process with this encoding?

Simulated annealing

Idea: The worse state is allowed during the solution of the problem, but only with certain probability.



A bit of physics

How to put sugar cubes into a box? Try this method

1. Pour cubes into a box. They take random positions.
2. Than shake the box, until we get the configuration of the sugar cubes

   positions having the minimal potential energy.

What box shaking means?

If we want to get an appropriate cube positions with a low potential energy, we shake, that means we create transient position configuration, which might have even greater or high potential energy.

This is the principle of the simulated annealing.

Distribution of configurations of the set of distinguishable particles at the temperature T is given as:

$$p(T,s)=\frac{1}{n}\exp\left(\frac{-E(s)}{kT}\right)$$

$$n=\sum_{s}\exp\left(\frac{-E(s)}{kT}\right)$$

E – free energy

T – temperatura

k- Boltzman constant

Metropolis algorithm: Let the system at the time t in the state s s having the free energy $E(s)$ and the temperature T. At $t+1$ a new configuration is created

s_{new} , ith a new energy E_{new}.

Energy difference between the new and old states, configs:

$$\Delta E = E(s_{new}) - E(s)$$

If $\Delta E \prec 0$ accept the new configuration with the probability 1.

If $\Delta E \geq 0$ accept the new configuration with the probability p

$$p = \exp\left(-\frac{\Delta E}{k.T}\right)$$

What is „energy“ in the simulated annealing?

It is a function in which we want to find a minimum (sum of the blackberries potential energies, path between the towns in the traveling salesman problem, evaluation function in the *n*-queen problem etc.)

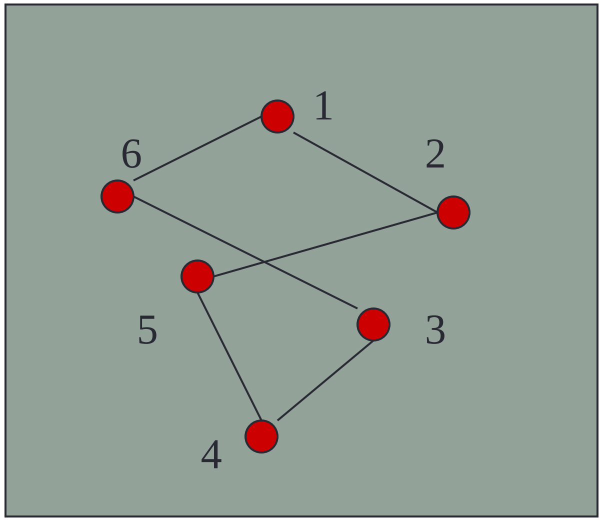
.

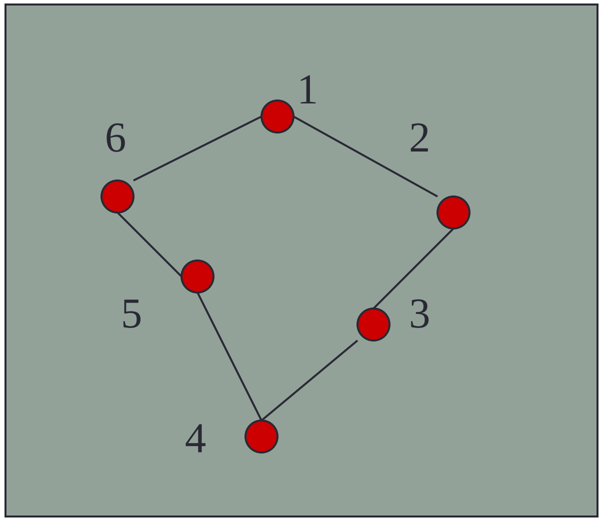
What is „temperature“ in the simulated annealing?

It is a constant governing the acceptance of a “worse” configuration, state.

$$p = exp\left(-\frac{\Delta E}{k.T}\right) = exp\left(-\frac{\Delta E}{T'}\right) = \frac{1}{exp\left(\frac{\Delta E}{T'}\right)}$$

Example – traveling salesman problem.





s=(1,2,5,4,3,6,1)

Initial configuration

Perturbed initial configuration

f(s) – function we want to minimize, distance between the towns

1) Create town ordering (configuration)

2) Calculate the path length for this configuration. This is „energy“ – function, we want to find a minimum of.

3) Disturb initial configuration; if the path among the towns is then longer, regulate the temperature so, that the probability of acceptance of this worse stat is close to one.

4) Repeat *m* times at this temperature.

5) Lower the temperature and thus the acceptance probability of a longer path config and repeat the process at this new temperature.

Algorithm

```
**Algorithm** SIMULATED-ANNEALING
**Begin**
  *temp* = INIT-TEMP;  put an initial temperature
  *konfig* = INIT-konfig;  initial configuration
  **while** (*temp* > FINAL-TEMP) **do**
          **while** (*inner_loop_criterion* = FALSE) **do**
                    *new_konfig* = PERTURB(*konfig*);
                    $\Delta E$ = E(*new_konfig*)  -E(*konfig*);
                    **if** ($\Delta E < 0$) **then**
                              *konfig* = *new_konfig*;
                    **else if** (RANDOM(0,1) > $e^{-(\Delta E/temp)}$) **then**
                              *konfig* = *new_konfig*;
          *temp* = SCHEDULE(*temp*);
**End.**
```

Simulated annealing search properties

- It is possible to prove mathematically , that if the temperature *T* decreases infinitely slowly, then the simulated annealing finds global optimum with the probability close to one .
- Simulated annealing is used in layout problems, planning problems etc.

Summary

1. How to find a good heuristic
2. Local search principle
3. Algorithms (generate and test, hill climbing, genetic, simulated annealing)

Tasks to think about

Let us have a map of Europe. We have three different colors. The goal is to color the states by such a way, that two neighbor states do not have the same color. Formulate as a searching problem.

What is state ? How many different states we have?

What is a goal state?

What is a state space?