Evoluční algoritmy a příbuzné techniky

doc.Ing. Jiří Krejsa, PhD

krejsa@fme.vutbr.cz, A2/710

Genetické algoritmy (genetic algorithms)

- Základní princip evolučních algoritmů
- Operátory vyhodnocení, selekce, křížení, mutace
- Úlohy s omezením

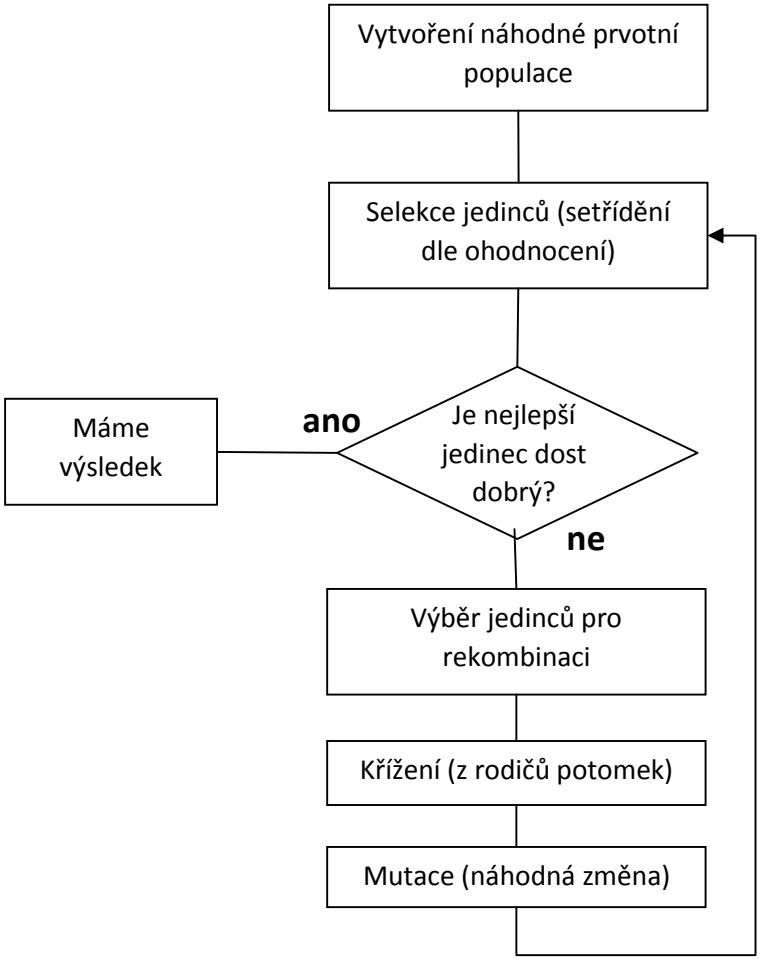
Algoritmy hejna (swarm algorithms)

- Mravenčí kolonie / roj (ant colony optimization)
- Hejno částic (particle swarm optimization)
- Včelí roj (artificial bee colony algorithms)

Základní principy evolučních algoritmů

- Klasická optimalizace: počáteční odhad řešení, který postupně vylepšuji (např. gradientní metodou)
- Evoluční optimalizace: celá množina kandidátů řešení úlohy – populace. Ti mezi sebou „soupeří“, ti lepší „přežívají“ a „množí se“ v dalších generacích, které jsou lepší a lepší.
- Základní termíny
  - Jedinec (individual)– představuje nějaké řešení zadané úlohy
  - Populace (population) – množina jedinců
  - Ohodnocení (fitness) – funkce, která určí jak „dobrý“ je jedinec
  - Selekce (selection) – výběr jedinců v populaci, kteří „přežijí“
  - Rekombinace – způsob tvorby nových jedinců do příští generace
    - Křížení (cross-over) – kombinace dvou či více jedinců do nového jedince
    - Mutace (mutation) – náhodná změna jednoho či více parametrů jednoho jedince

Základní algoritmus evoluční optimalizace



Co musíme vyřešit?

- Jak vypadá jedinec?
- Vytvoření náhodné populace – jak velké?
- Setřídění jedinců – ohodnocení
- Výběr jedinců – jen nejlepší? Půlka? …
- Křížení – ze dvou, z více, .. ze kterých?
- Mutace – jak moc?

Jedinec

- Obsahuje zakódované řešení úlohy
- Řešení úlohy – fenotyp
- Zakódované řešení - genotyp
- Nejjednodušší varianta: řetězec fixní délky (řetězcům se někdy říká chromozomy, jednotlivým prvkům řetězce geny)
- Příklady:
  - Úloha hledání maxima funkce jedné proměnné $y=f(x)$
    - Fenotyp = x
    - Genotyp – například binární kódování o určité délce [1 0 0 1 0 0 1 0 0 0 1 0 1 0 1]
    - Konečná přesnost (délka řetězce), hledání pouze na určitém intervalu
  - Problém obchodního cestujícího
    - Fenotyp = Genotyp (sekvence měst A,B,C,D,….)

Ohodnocení

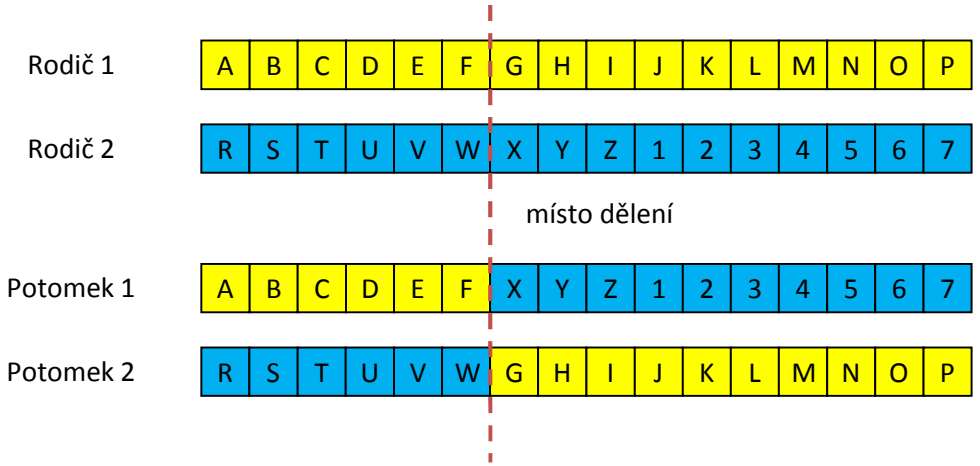
- Musí existovat způsob, jak ohodnotit, zda daný jedinec je lepší, než jiný jedinec
- Někdy je to lehké (maximum funkce, obchodní cestující)
- Někdy je to složitější (hledáme parametry nějakého procesu, k ohodnocení musíme ten proces nechat běžet)
    - cesta mobilního robotu – jedinec = sled klíčových bodů
    - úloha – projet z místa A na místo B co nejdále od zdí
    - ohodnocení – simulační projetí cesty
- Ohodnocení všech jedinců v populaci – nezávislé, ideální pro paralelní zpracování (v Matlabu jednoduše cyklem **parfor**)

Selekce

- Selekční tlak – jen ti nejlepší přežijí, respektive budou použiti pro další generaci
- Nejjednodušší řešení: setřídím podle ohodnocení, vezmu tu lepší polovinu
- Spousta dalších variant
  - Vyberu polovinu (třetinu, …) náhodně, ale pravděpodobnost výběru je tím vyšší, čím lepší je ohodnocení (mechanismus ruletového kola)
  - Výběr jedinců k „množení“ je buď jen z těch vybraných, nebo ze všech, ale část z těch lepších přežije do další generace

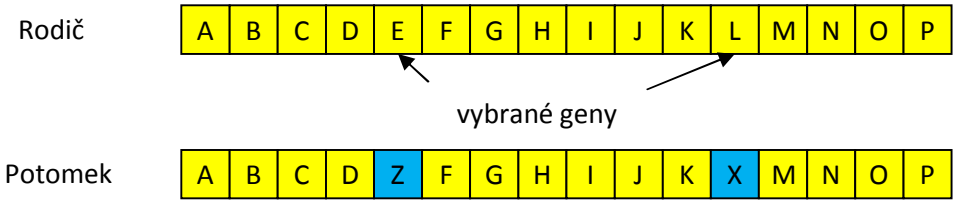
Rekombinační operátory – křížení

- Způsob, jak udělat z původních jedinců jedince nové
- „kombinace“ rodičů do nového potomka
- Nejjednodušší varianta:
  - rozdělení chromozomu na náhodném místě
  - kombinace levé a pravé části chromozomu
  - může vzniknout jeden nebo více potomků



Rekombinační operátory – mutace

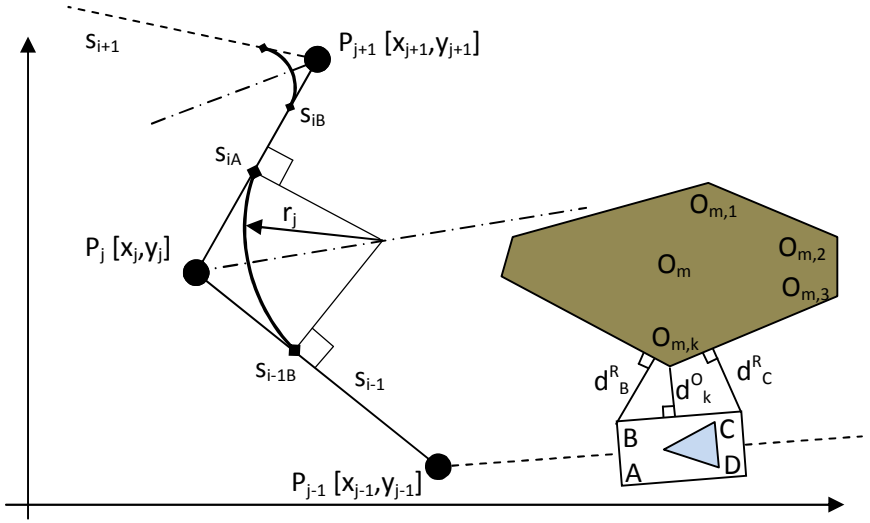
- Jediný rodič
- Náhodná změna jednoho nebo více genů
- Postupné zmenšování změn



Příklad s hledáním optimální cesty pro mobilní robot

Úloha: projet cestu mezi překážkami tak, aby vzdálenost od překážek byla co největší

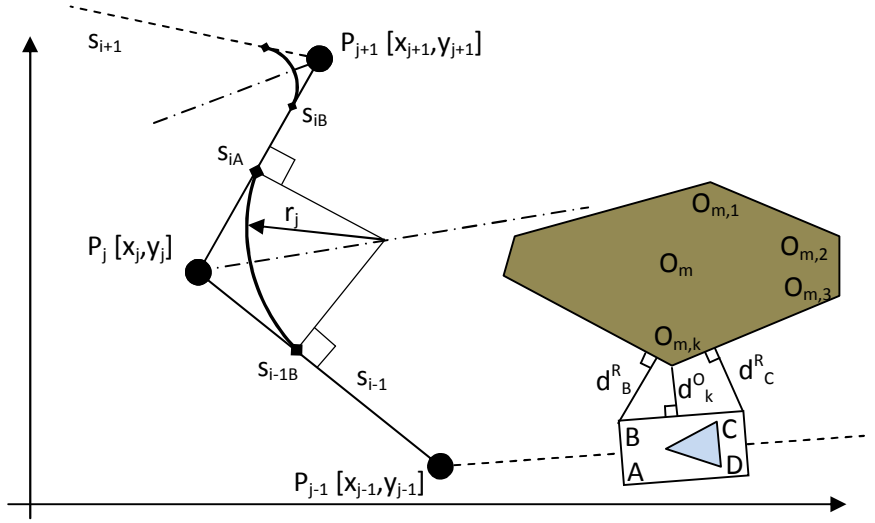
Reprezentace cesty



- Cesta je reprezentována uspořádanou množinou bodů P o souřadnicích [x,y]
- Každý bod P má navíc přiřazený poloměr zatáčení r
- Cesta se tak skládá z rovných úseků a oblouků
- Když dám r=0, můžu použít holonomní podvozek (otočí se na místě)

Ohodnocení

- Robot projede simulačně celou trasu
- Během simulace (diskrétní) počítá vzdálenosti od překážek
- Vzdálenosti jsou
  - Od vrcholů robota k překážkám, od vrcholů překážek je stěnám robotu
  - Nejjednodušší ohodnocení: nejmenší vzdálenost překážky na celé trase
- Výpočet jen v okolí robota (úspora výpočetního výkonu)
- Složitější ohodnocení (průměrná vzdálenost od překážek, délka cesty, …)

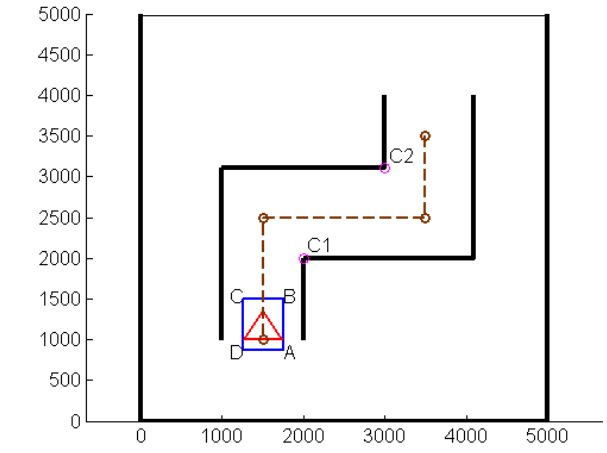


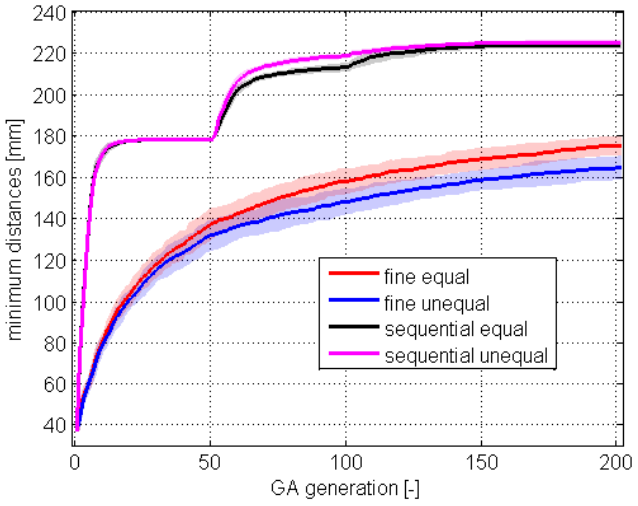
Optimalizace

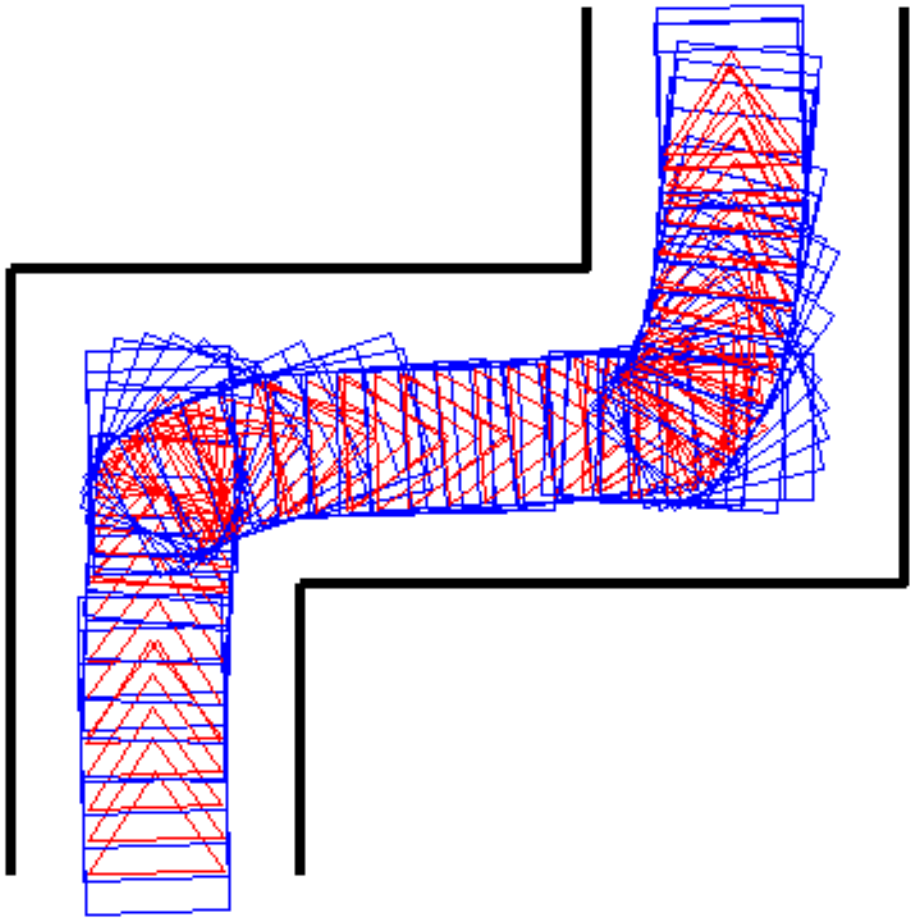
- Kontrola cesty na validitu (poloměr může být tak velký, že nezůstane žádný rovný úsek cesty)
- Invalidní jedinci jsou ohodnoceni jako velmi špatní
- Finta – postupné zjemňování reprezentace cesty
- Necháme algoritmus běžet, až se přestane zlepšovat, přidáme další klíčové body cesty

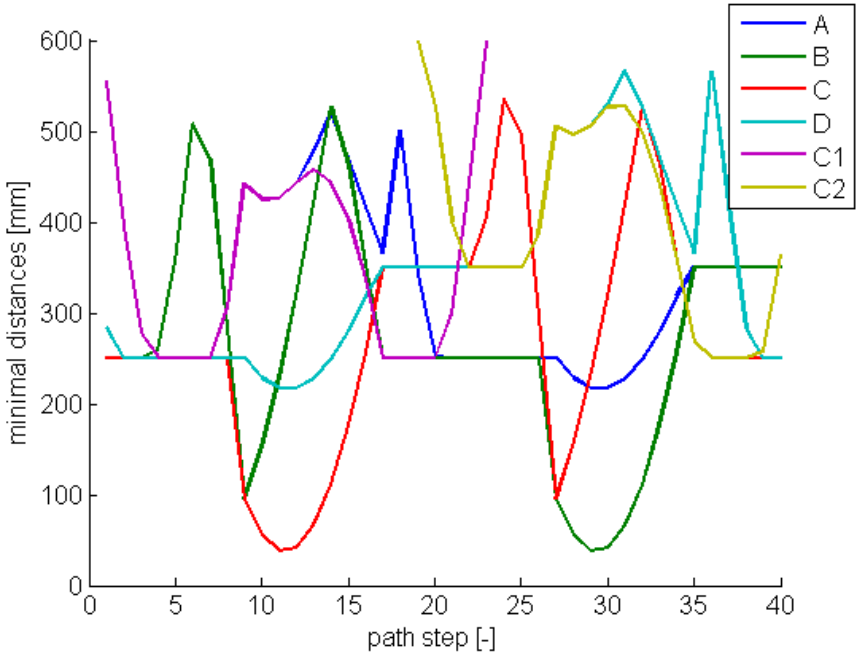
```
Algorithm OptimizePath(KeyPath, World, Parameters)
For K = 1 to RefinementSteps
    Population = CreatePopulation(KeyPath, PopSize)
    For L = 1 to OptimizationSteps
    // Population.CalculateFitness(World)
        ForEach individual in Population
            Path = SamplePath(individual)
            Distances = GetDistances(Path,World)
            Fitness = GetFitness(Distances)
        EndForEach
        Population.SortAndKeepFittest
        Repeat
            Population.PerformMutation
            Population.CheckPathConsistency
        Until Population.HasSufficientSize
        Repeat
            Population.PerformCrossOver
            Population.CheckPathConsistency
        Until Population.HasSufficientSize
    EndFor
    KeyPath = Population.BestSolution
    KeyPath = RefineKeyPath(KeyPath)
EndFor
```

Konkrétní úloha – dvojitá zatáčka ve tvaru Z – holonomní robot

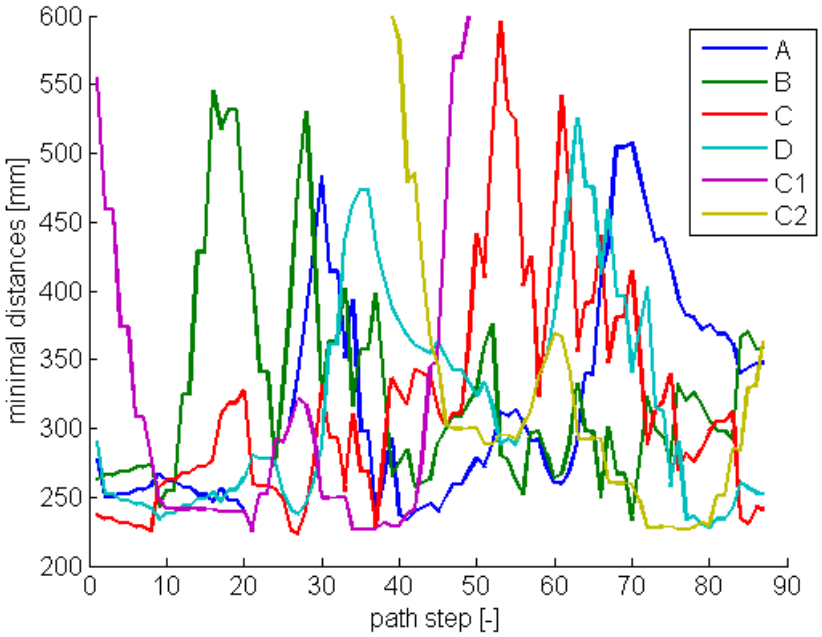






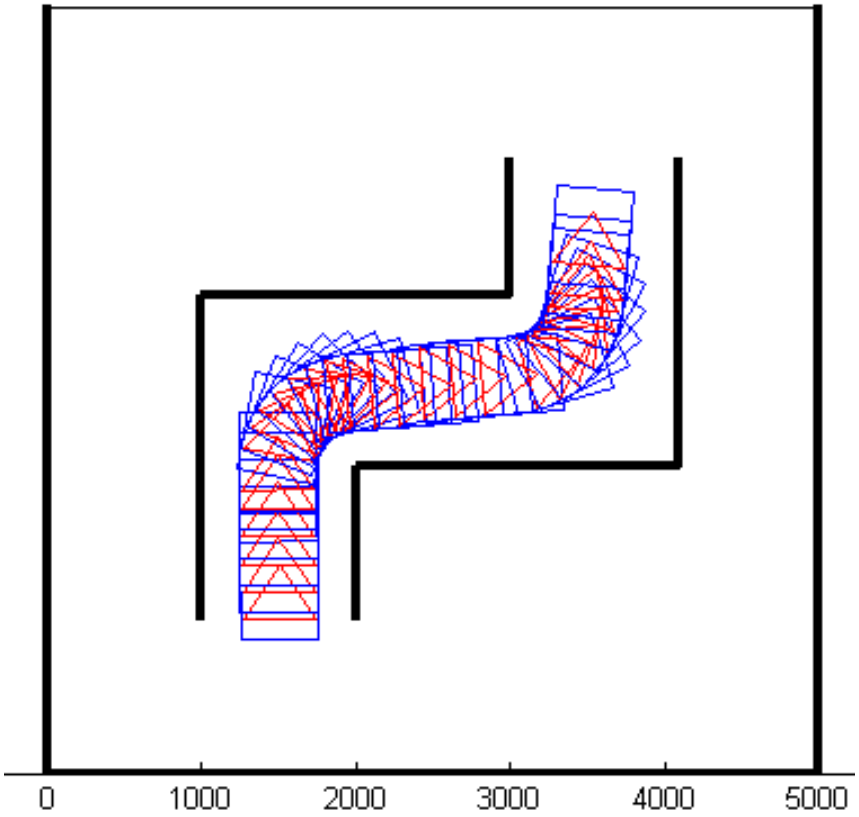


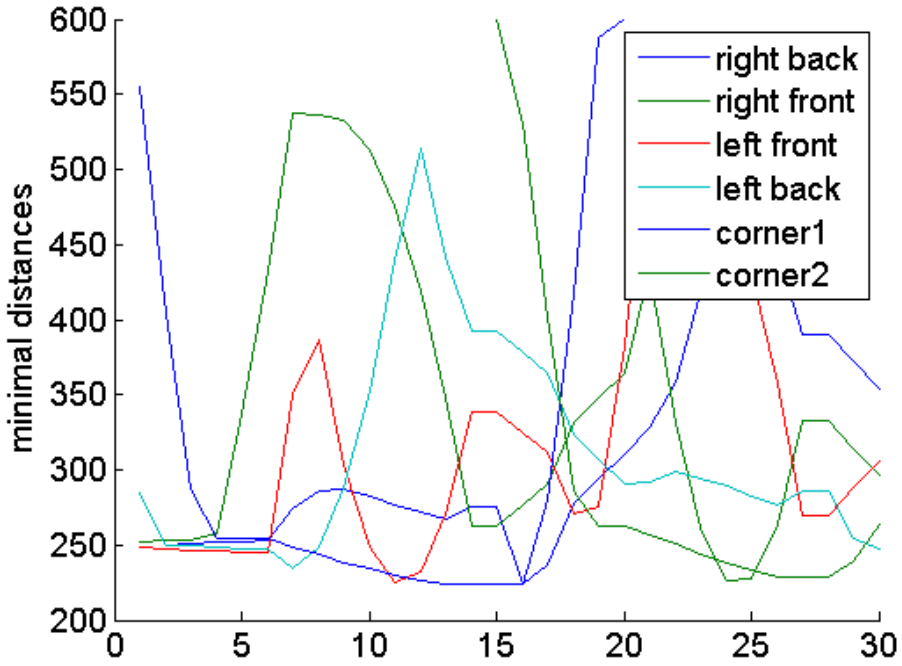
Minimální vzdálenosti na počátku



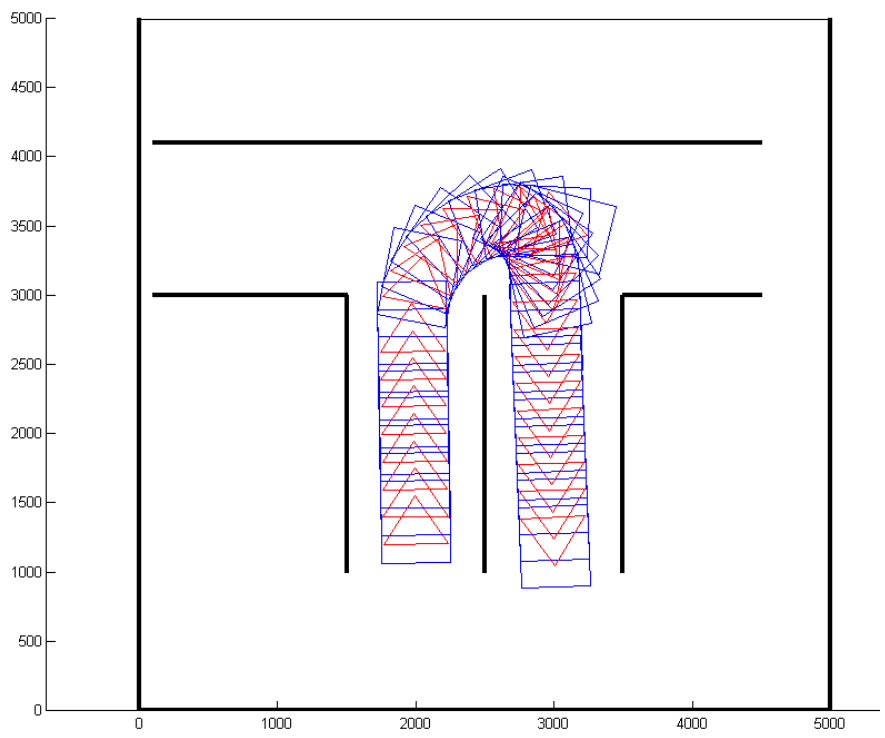
Minimální vzdálenosti na konci

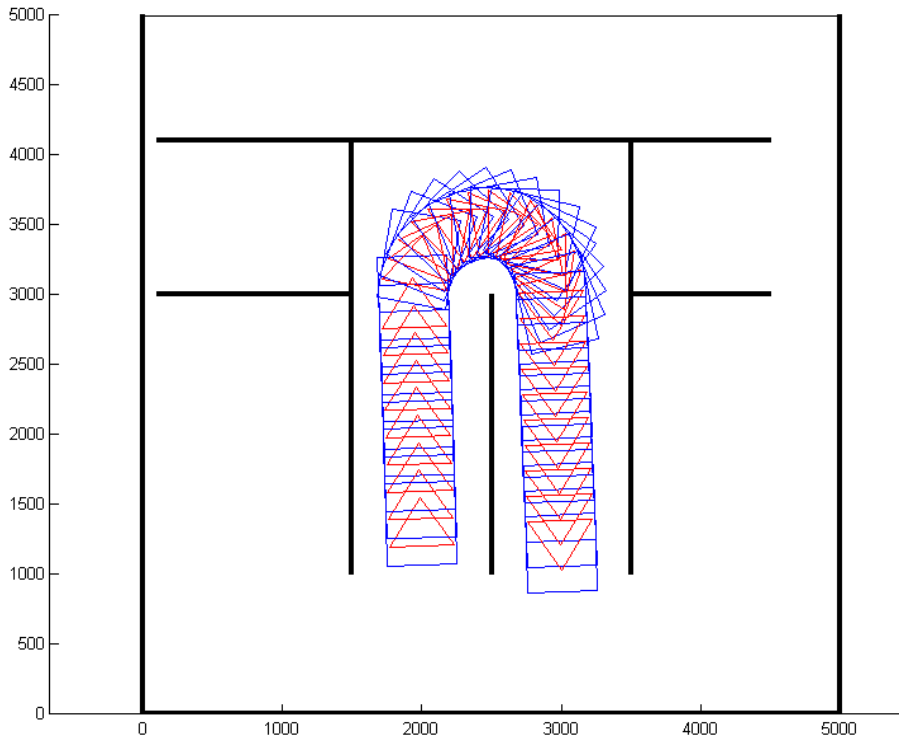
Nenulový rádius otočení (neholonomní robot)





Obrátka – různý prostor kolem robotu





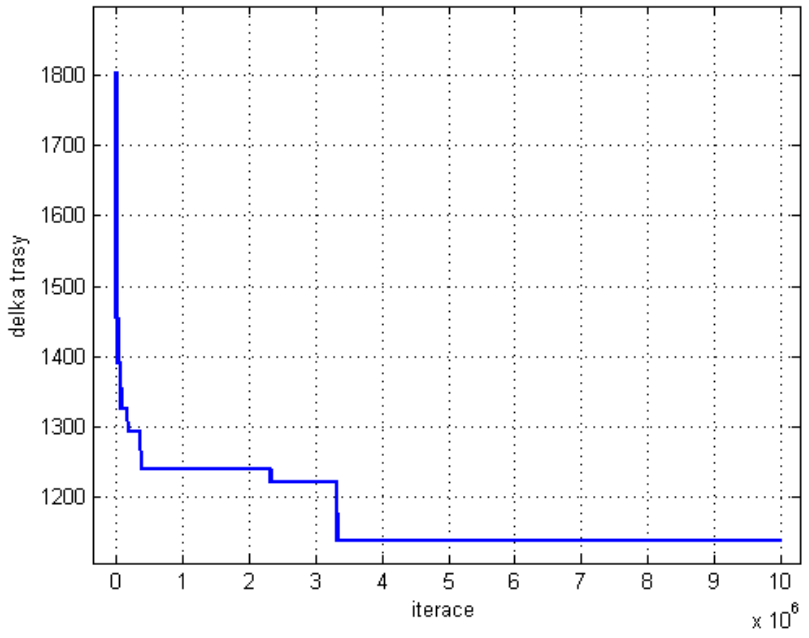
Úlohy s omezením – problém obchodního cestujícího

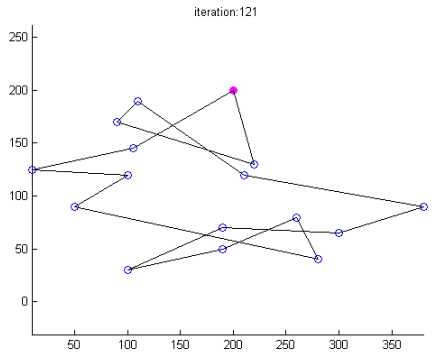
- Problémy:
  - Nemůžu mít stejný gen 2x
  - Cyklická úloha ([a b c] je totéž jako [c a b]) – výběr jednoho města jako počátečního
  - tradiční **křížení** nedává smysl (nevylepším cestu)
    - operátor křížení se zachování pořadí (OX – order crossover)
    - operátor křížení s částečným zobrazením (PMX - partially mapped crossover)
    - operátor křížení s rekombinací hran (ERX – edge recombination crossover)
  - mutace
    - přehození měst, nebezpečí uváznutí v lokálním extrému při malé populaci
    - inverze kousku cesty (města v opačném pořadí)
    - přesun kousku cesty (na náhodné místo)

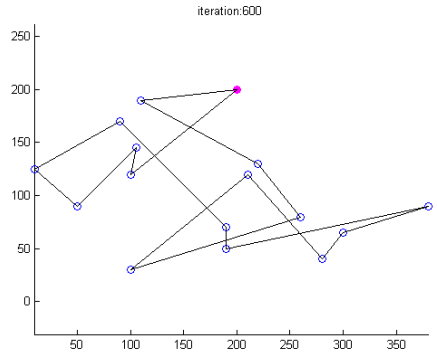
Příklad, implementace v Matlabu

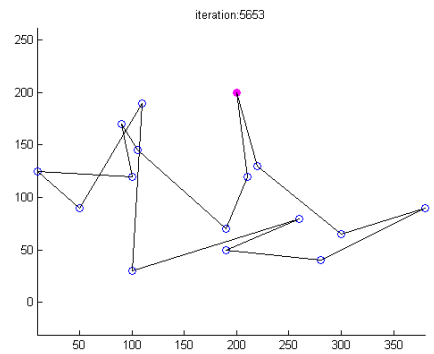
- mapa ČR (z úlohy o prohledávání stavového prostoru)
- počet měst: 15 + jedno extra
- počet permutací: 15! (cca 1,3 x 10^{12})
- řešení hrubou silou: permutace v cyklu, 10^7 pokusů, nejlepší řešení uchovávám

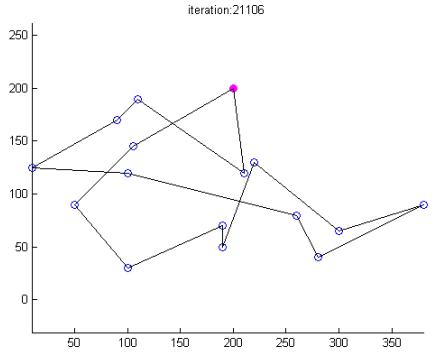
```
map = mapCreateCR();

startCity.pos = [200,200];

bestChain = randperm(mapGetCityCount(map));
val = evaluateIndividual(map,startCity,bestChain);

bestCourse = val;

for i= 1:10000000
    chain = randperm(mapGetCityCount(map));
    val2 = evaluateIndividual(map,startCity,chain);
    if val2 < val
        bestChain = chain;
        val = val2;
        mapDraw(map,startCity,bestChain,i);
    end
    sprintf('current %f, best so far: %f',val2,val)
    bestCourse(i+1) = val;
end
```

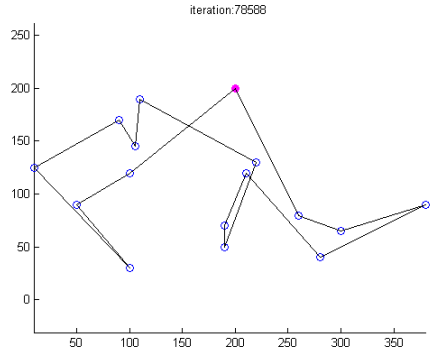


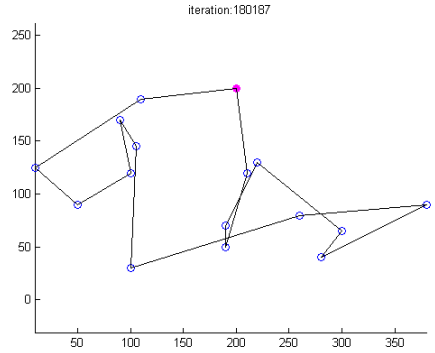

iteration:121

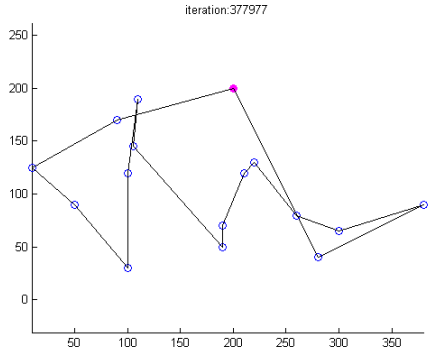

iteration:600

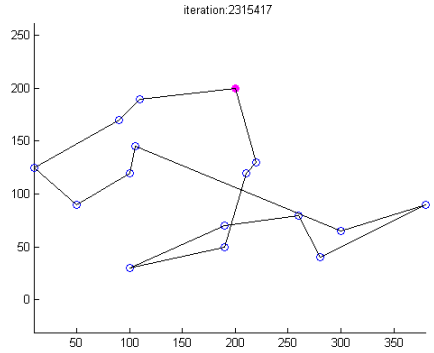

iteration:5653

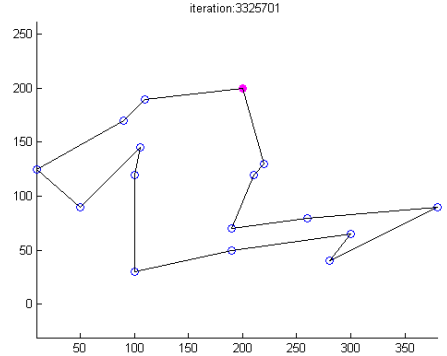

iteration:21106


iteration:78588


iteration:180187

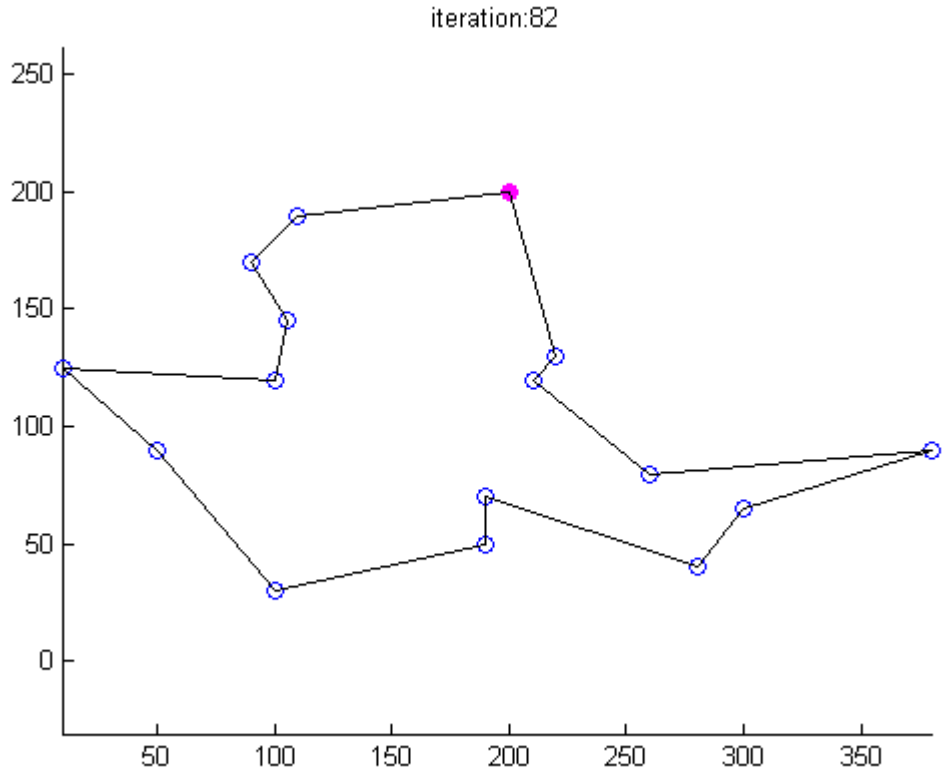

iteration:377977


iteration:2315417


iteration:3325701

Optimální řešení

Délka cesty 989,16 km



Operátor mutace (je jednodušší)

Přehození dvou měst

- Města vybírám náhodně
- Jen přehodím města (zachovám validitu chromozomu)

```
function [mutated] = indMutate(individual,indL)
% mutates single individual
% individual - vector of integers, indL long

% replace two genes
idxA = randi(indL);
idxB = randi(indL);

mutated = individual;
mutated(idxA) = individual(idxB);
mutated(idxB) = individual(idxA);

end
```

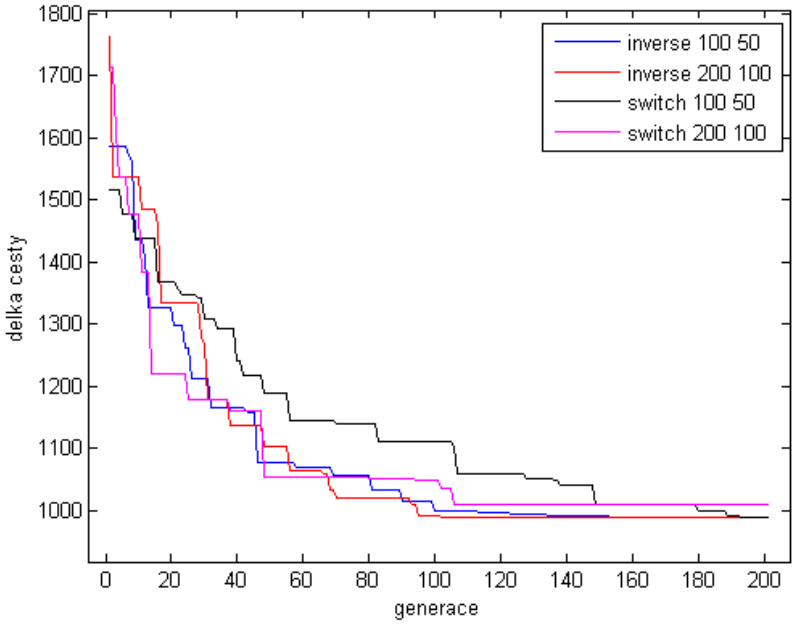
Inverze kousku cesty

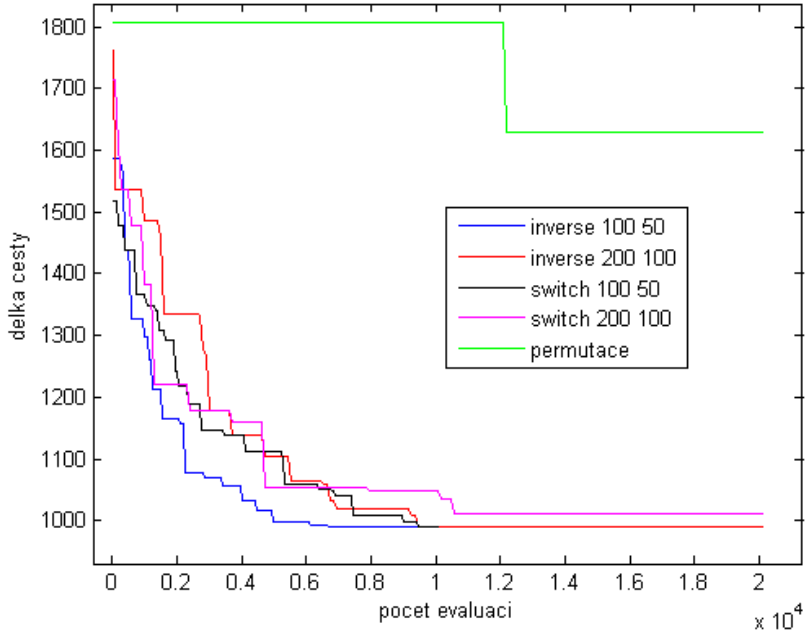
- Náhodně vybraný úsek převrátím
- Stále zachována validita chromozomu

```
function [mutated] = indMutateInverse(individual,indL)
% mutates single individual
% individual - vector of integers, indL long
% takes individual, selects random piece of path and
reverses it

L = indL;
s1 = randi(L);
s2 = randi(L);

a1 = min(s1,s2);  % first cut
a2 = max(s1,s2);  % second cunt


% make sure the cuts are not the same or out of range
if (a1 == a2)
    a2 = a2 + 1;
    if a2 > L
        a2 = L;
        a1 = a1 - 1;
    end
end

mutated = individual;
chunk = individual(a1:a2);
invchunk = flip(chunk);
mutated(a1:a2) = invchunk;
```

Vliv velikosti populace

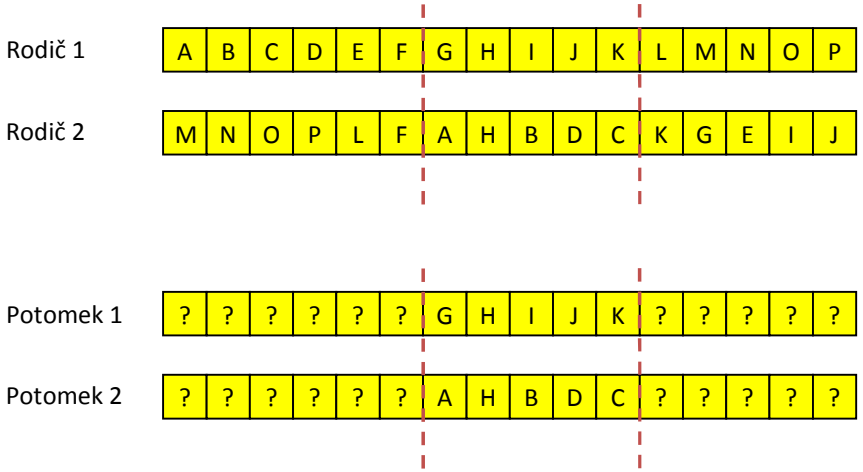
- Co je na x-ové ose?
- Generace – nevypovídá přímo o počtu operací
- Srovnání s náhodným prohledáváním





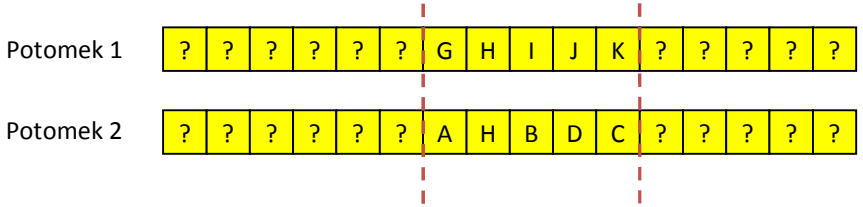
Operátor křížení s částečným zobrazením (PMX - partially mapped crossover)

- Náhodně rozdělím jedince na dvou místech
- Zkopíruji prostřední část




```
function [child1,child2] = indCrossPMX(parent1,parent2)
% performs partially mapped crossover - PMX

L = numel(parent1);  % length of chromosome

s1 = randi(L);
s2 = randi(L);

a1 = min(s1,s2);  % first cut
a2 = max(s1,s2);  % second cunt

% make sure the cuts are not the same or out of range
if (a1 == a2)
    a2 = a2 + 1;
    if a2 > L
        a2 = L;
        a1 = a1 - 1;
    end
end

% cuts meaning - middle section is from a1 to a2
INCLUDING a1 and a2

child1 = zeros(L,1);
child2 = zeros(L,1);

% copy the middle
child1(a1:a2) = parent1(a1:a2);
child2(a1:a2) = parent2(a1:a2);
```

- Střední části řetězce implikují zobrazení

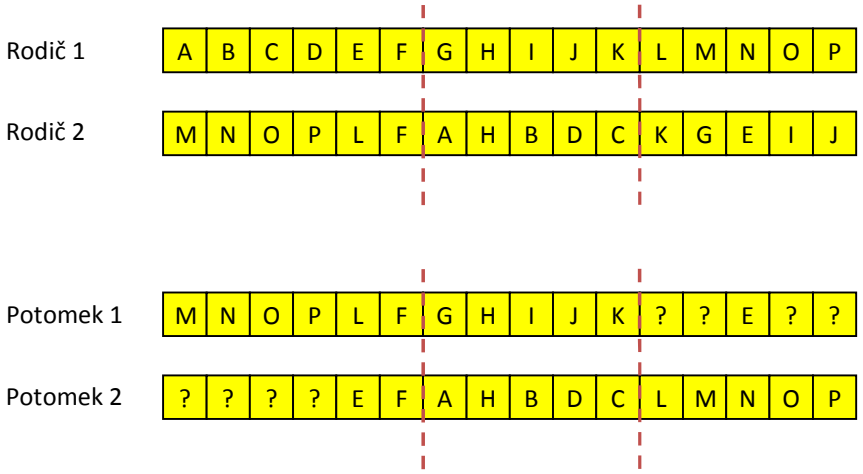


G↔A, H↔H, I↔B, J↔D, K↔C

Tato zobrazení se mi budou hodit v dalších krocích algoritmu.

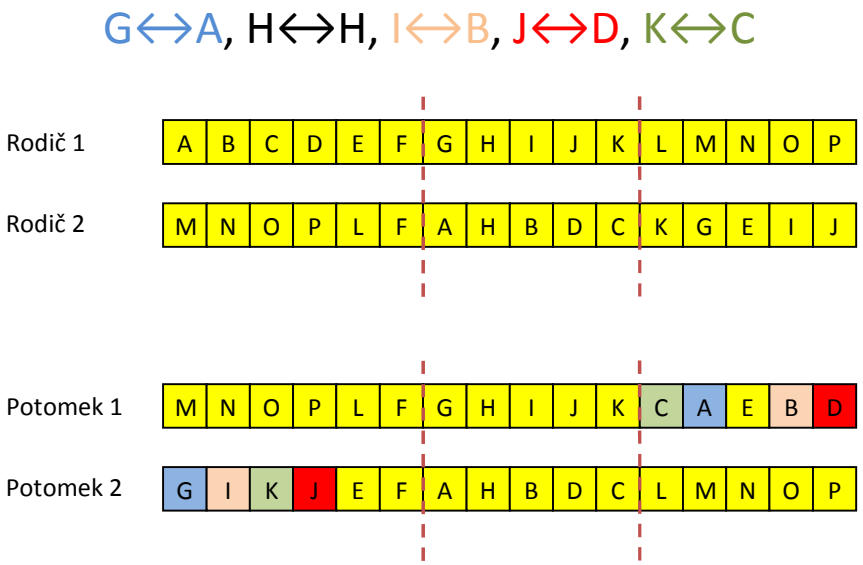
```
% create implications from the middle section between
corresponding cities
for i=1:(a2-a1)+1
    implications(i,:) = [child1(a1-1+i),child2(a1-
1+i)];
end
```

- Doplním nekonfliktní města z druhého rodiče




```
% now lets add front and end elements from the other
parent, that do not
% violate duplicity condition

% front first
for i=1:a1-1
    tmp = parent1(i);
    if isempty(find(child2==tmp))  % city not there yet
        child2(i) = tmp;
    end
    tmp = parent2(i);
    if isempty(find(child1==tmp))  % city not there yet
        child1(i) = tmp;
    end
end

% now the end
for i=a2+1:L
    tmp = parent1(i);
    if isempty(find(child2==tmp))  % city not there yet
        child2(i) = tmp;
    end
    tmp = parent2(i);
    if isempty(find(child1==tmp))  % city not there yet
        child1(i) = tmp;
    end
end % for
```

- Využiji zobrazení a doplním co chybí (pokud to jde)

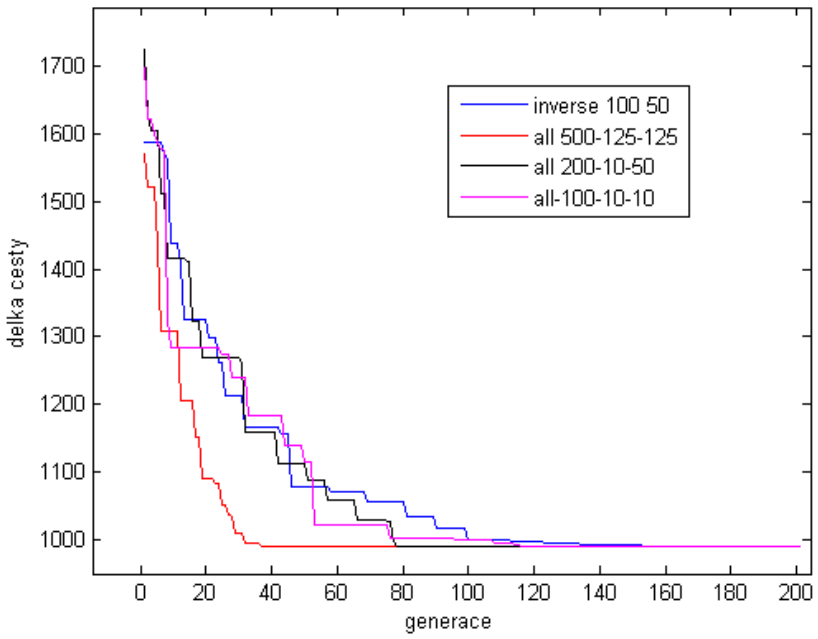


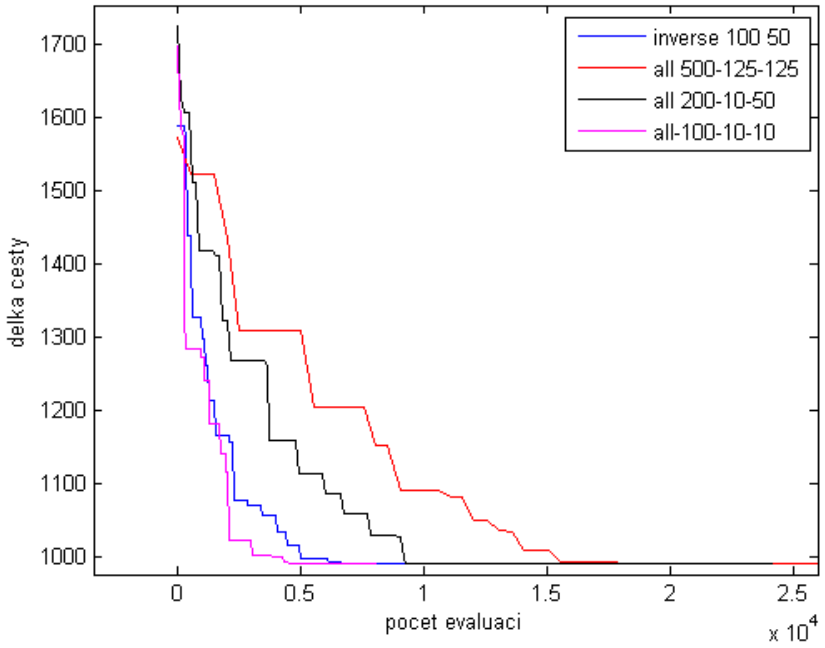

```
% now lets use implications
for i=1:L
    if child1(i) == 0  % missing city
        parentcity = parent1(i);
        implicationidx =
find(implications(:,1)==parentcity);
        if ~isempty(implicationidx)
            candidate = implication(implicationidx,2);
            if isempty(find(child1==candidate))
                child1(i) = candidate;
            end
        end
    end %if
    % same for second child
    if child2(i) == 0  % missing city
        parentcity = parent2(i);
        implicationidx =
find(implications(:,2)==parentcity);
        if ~isempty(implicationidx)
            candidate = implication(implicationidx,1);
            if isempty(find(child2==candidate))
                child2(i) = candidate;
            end
        end
    end %if
end %for
```

- Pokud mi něco chybí, doplním nekonfliktně

```
% finally add missing cities

missingidx = find(child1==0);  % find all missing positions
% find missing cities (this should be done in more elegant way)
missing = [];
for i=1:L
    if isempty(find(child1==i))
        missing = [missing, i];
    end
end
for i = 1: length(missingidx)
    % put random valid city there
    child1(missingidx(i)) = missing(i);
end

% same for child2
clear missingidx;
missingidx = find(child2==0);  % find all missing positions
% find missing cities (this should be done in more elegant way)
missing = [];
for i=1:L
    if isempty(find(child2==i))
        missing = [missing, i];
    end
end
for i = 1: length(missingidx)
    % put random valid city there
    child2(missingidx(i)) = missing(i);
end
```

Výsledky kombinace operátorů





Závislost na velikosti populace – nenechte se příliš ovlivnit grafy výše. Každý běh algoritmu je trochu jiný. A hodně záleží na mohutnosti problému, pro násobně větší množství měst to bude vypadat jinak…

Turnajová selekce

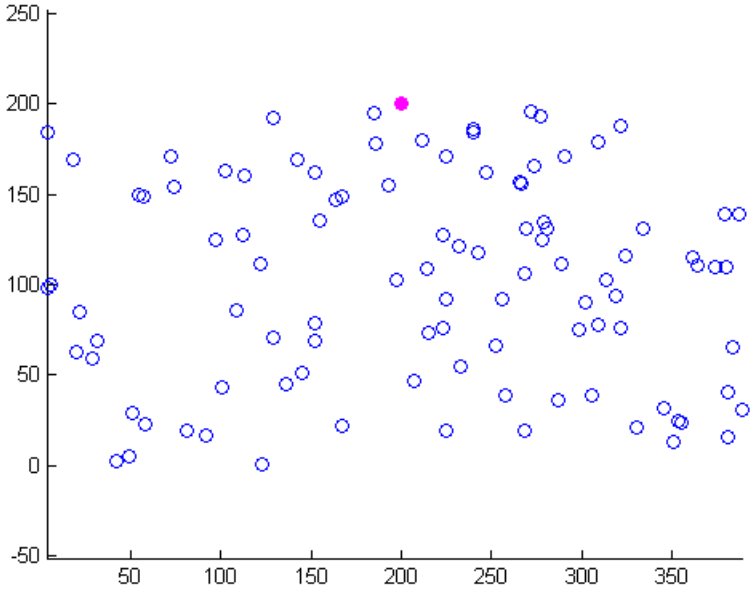
- uchováme nejlepší řešení
- nebo množinu nejlepších – elita
- další jedince vybíráme tak, že vybereme náhodně několik jedinců do turnaje a vybereme pouze vítěze
- velikost turnaje může být i malá (2)

```
function [newPop] =
popSelectTournament(pop,FinalPopSize,eliteCount,tournamentSize)
% selects individuals for further breeding
% pop - original population - sorted
% FinalPopSize - newPop size
% eliteCount - how many individuals to store automatically
% tournamentSize - number of individuals competing in each
group

% keep the elite
for i=1:eliteCount
    newPop{i} = pop{i};
end

% remove elite from population
pop2 = pop;
pop2(1:eliteCount) = [];

% now repeat tournaments until the population is fulfilled
for i=eliteCount+1:FinalPopSize
    % arrange tournament
    candidates = randi(numel(pop2),tournamentSize,1);  % this
returns 3 random individual indexes
    % select the winner
    winner = min(candidates);  % find the best candidate
    % add to population
    newPop{i} = pop2{winner};
    % remove winner from population
    pop2(winner) = [];
end
```

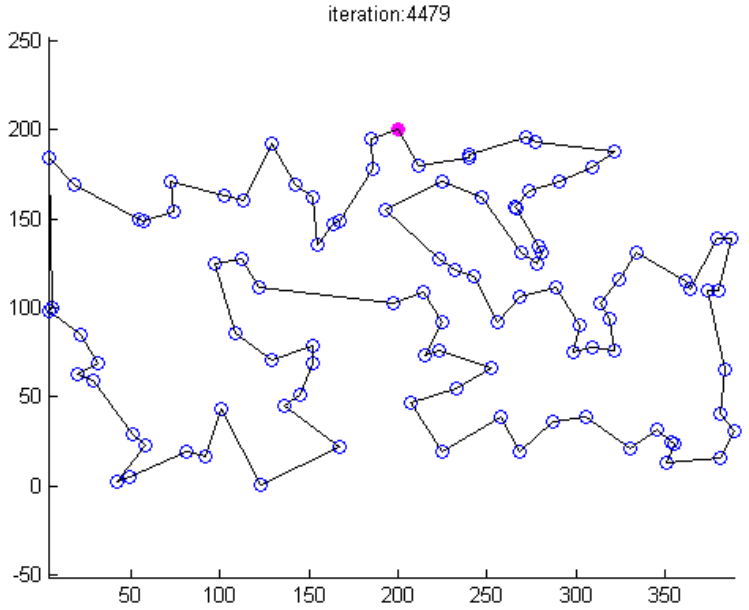
Příklad s více městy

- 100 náhodně generovaných měst
- Počet permutací: 10^{157}
- Použití křížení i obou typů mutací
- Roste počet parametrů algoritmu

```
for i= 1:iters
    %pop2 = popSelect(pop,popSize);
    pop2 = popSelectTournament(pop,popSize,eliteCount,tournamentSize);
    %pop3 = CrossOver(pop2);
    pop3 = popCrossover(pop2,cityCount,popSizeCross);
    pop3 = popSort(map,pop3,startCity);
    pop3 = popMutate2(pop3,cityCount,popSizeMutate,mutateSwitch);
    pop3 = popMutate2(pop3,cityCount,popSizeMutate,mutateInverse);
    pop = popSort(map,pop3,startCity);
    val2 = evaluateIndividual(map,startCity,pop{1});
    if val2 < val
        bestChain = pop{1};
        val = val2;
        mapDraw(map,startCity,bestChain,i);
        fprintf('improvement, iteration:%d, best so far: %f \n',i,val)
    end
    bestCourse(i+1) = val;
end
```



Náhodně rozložených 100 měst



Řešení s délkou cesty 2217km

Poznámky k úlohám s omezením

- Úloha s omezením může vést po rekombinaci na invalidního jedince
- Co se s tím dá dělat?
  - Speciální rekombinační operátory (to jsme si ukázali)
  - Opravy – korekce jedince na validního
  - Velmi nízké ohodnocení invalidního jedince – vypadne v selekci
  - Reprezentace, která neumožňuje vznik invalidního jedince (genotyp – fenotyp)

Příklad: jiná reprezentace pro úlohu obchodního cestujícího

- Ordinální reprezentace pro M měst
- Seznam o délce M
- i-tý prvek je číslo v rozsahu <1, 1 - (M – i + 1)>
- Co to znamená? Prvek uvádí, kolikátý prvek referenčního seznamu mám vzít
- Příklad dekódování:

Genotyp (zakódováno)	Cesta (fenotyp)	Reference
2 4 2 3 2 1	[]	[a, b, c, d, e, f]
2 4 2 3 2 1	[b]	[a, c, d, e, f]
2 4 2 3 2 1	[b, e]	[a, c, d, f]
2 4 2 3 2 1	[b, e, c]	[a, d, f]
2 4 2 3 2 1	[b, e, c, f]	[a, d]
2 4 2 3 2 1	[b, e, c, f, d]	[a]
	[b, e, c, f, d, a]	[]

- Předpis omezující rozsah i-tého prvku zabezpečí bezproblémové jednobodové křížení
- Nevýhoda: na úlohu obchodního cestujícího to moc dobře ,nefunguje, ale na některé úlohy ano.

Závěr genetických algoritmů

- Reálně to funguje
- Dobře se to paralelizuje
- Nevím, zda jsem našel globální optimum
- Špatně se škáluje při rostoucím počtu parametrů k optimalizaci
- Vlastní GA má hromadu parametrů
  - Velikost populace
  - Způsob selekce, další lokální parametry (velikost elity, velikost turnaje, …)
  - Typ mutace, její rozsah
  - Kdy mám skončit?

Genetický algoritmus v Matlabu

- Potřebujeme Optimization toolbox + Global optimization toolbox
- Genetický algoritmus pokrývá funkce **ga()**

- Vstupní argumenty
  - ga(*funkce, pocet_vstupu*) – *funkce* je adresa funkce, kterou chceme optimalizovat
  - ga(*funkce, pocet_vstupu, …*) – omezení
  - ga(*problem*) - *problem* je struktura, do které můžu postupně nacpat všechny parametry

- Výstupní parametry
  - x = ga(…) – nalezené řešení
  - [x, fit] = ga(…) – fit je hodnota fitness pro dané řešení
  - [x, fit, exitflag, output, population, scores] = ga(…)
    - exitflag – důvod proč byl GA ukončen
    - output – struktura s dalšími informacemi o průběhu výpočtu
    - population – kompletní populace poslední generace
    - scores – hodnoty fitness pro poslední populaci

Příklad 1 – hledání extrému funkce dvou proměnných

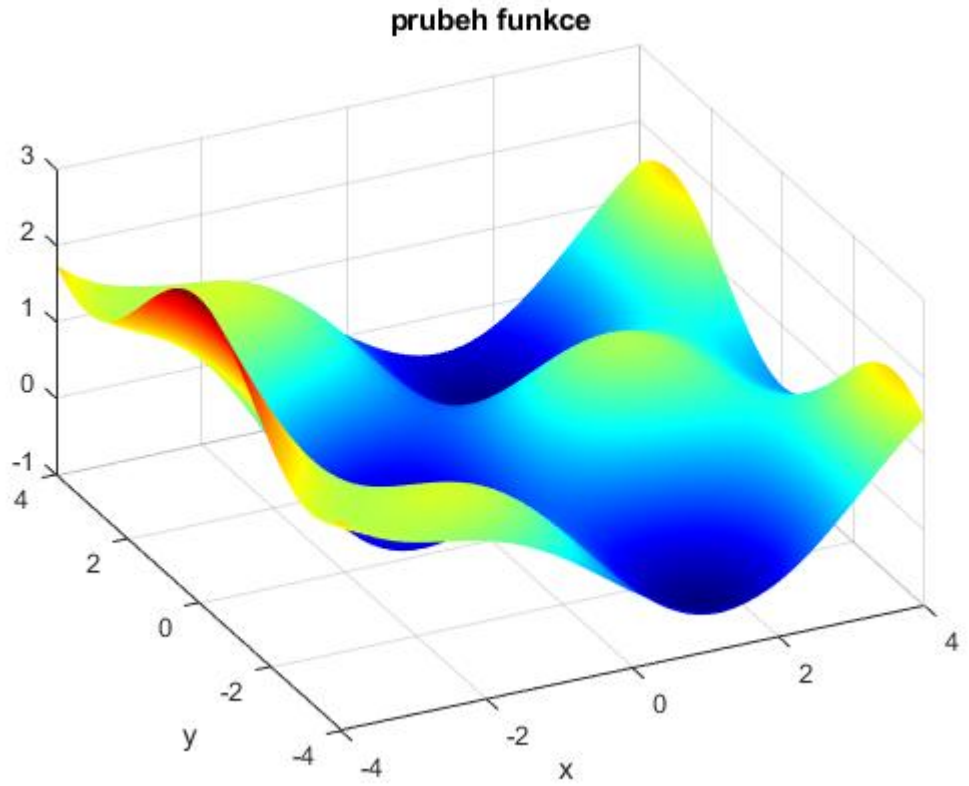
Uměle vytvořená funkce

$$f(x,y) = \sin(x)\cos(y) + 0.1x^2 - 0.01x^3$$

Funkce musí mít jediný vstupní parametr, kde na jednotlivé proměnné se dostaneme podle indexu.

```
function [vystup] = mojeFunkce(vstup)
% jednoducha funkce dvou promennych
% vstup je dvourozmerny vektor
x = vstup(:,1);
y = vstup(:,2);
vystup = sin(x) .* cos(y) + 0.1.*x.^2 - 0.01.*x.^3;

end
```

Zobrazíme si ji na intervalu <-4,4>




```
xi = linspace(-4,4,300);
yi = linspace(-4,4,300);
[X,Y] = meshgrid(xi,yi);
Z = mojeFunkce([X(:),Y(:)]);
Z = reshape(Z,size(X));
surf(X,Y,Z,'MeshStyle','none')
colormap 'jet'
view(-26,43)
xlabel('x')
ylabel('y')
title('prubeh funkce')
```

A můžeme zkusit spustit GA v Matlabu v té nejjednodušší podobě

```
% vlastni optimalizace
rng default % nastaveni random generatoru, aby se to dalo pocitat furt dokola se
stejnym vysledkem
x = ga(@mojeFunkce,2)
```

```
>> ga_test01
Optimization terminated: maximum number of generations exceeded.

x =

   1.0e+03 *

    1.2260    0.1346
```

Vidíme, že jsme sice dostali jakýsi výsledek, ale je v řádu tisíců. Je to proto, že se hledá globální extrém funkce. Nás ale zajímá jen interval <-4, 4> (pro obě proměnné).

Jak to zadat v Matlabu?

Pracujeme s takzvanými Linear inequality constraints (lineární „nerovnicová“ omezení)

Zadávají se v maticové podobě

$$\boldsymbol{A}x <= \boldsymbol{b}$$

Kde

- počet řádků v matici A je počet omezení
- počet sloupců v matici A je počet proměnných naší hledané funkce

Matice ***A*** a ***b*** jsou parametry zadané do funkce *ga()*

Všimněte si ovšem, že je to nerovnost **menší nebo rovno** než. Ale my máme interval uzavřený z obou stran.

Co s tím?

V naší funkci máme dvě proměnné, x a y

Konkrétní nerovnice tedy budou

$x < 4$ $y < 4$ $-x < 4$ $-y < 4$

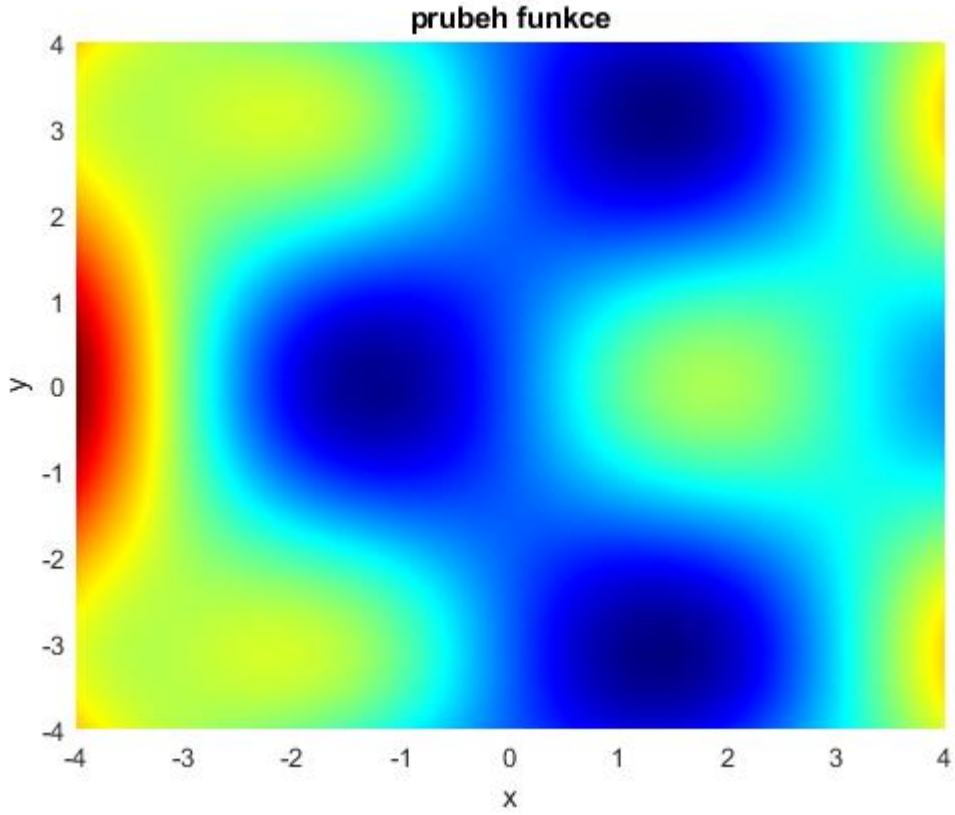
Maticově

$A = [1\ 0;0\ 1;-1\ 0;0\ -1]$; $b = [4\ 4\ 4\ 4]$

```
% vlastni optimalizace
rng default % nastaveni random generatoru, aby se to dalo pocitat furt dokola se
stejnym vysledkem
A = [1 0;0 1;-1 0;0 -1];  % linear constraint
b = [4 4 4 4];

x = ga(@mojeFunkce,2,A,b)
```

```
>> ga_test01

Optimization terminated: average change in the fitness value less than
options.FunctionTolerance.

x =

    1.3534    3.1416
```

Je to opravdu globální minimum?

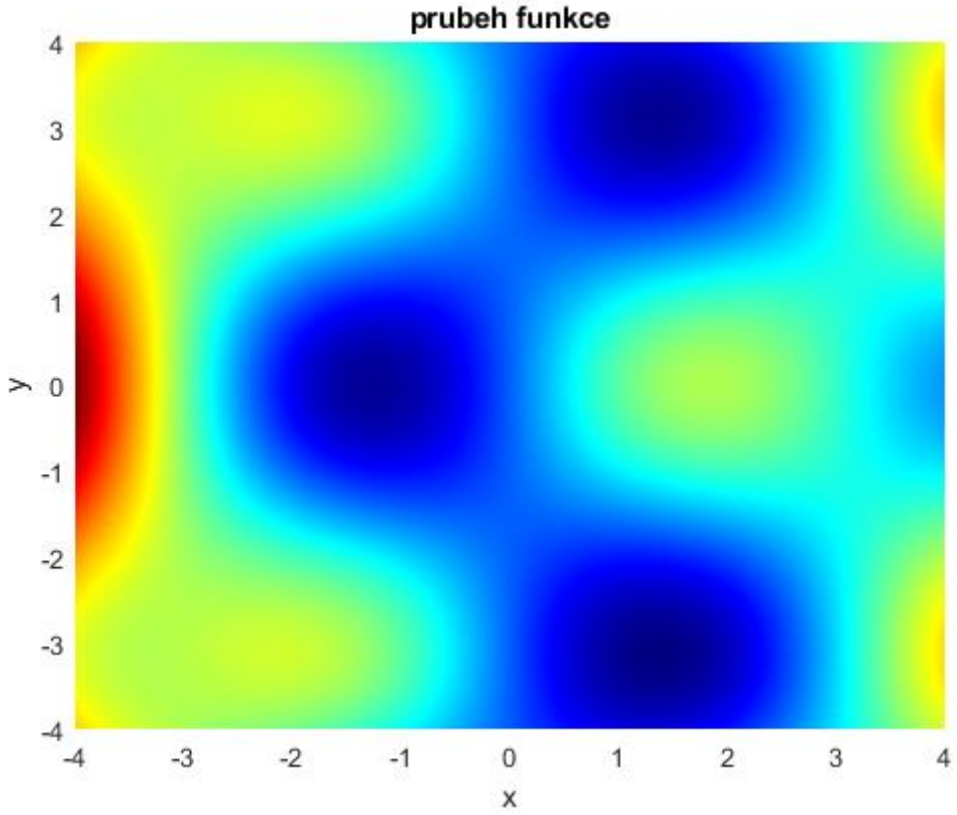


Naše funkce je kulišácká, ma dva stejné extrémy. Který dostaneme?

Když budemem optimalizaci pouštět opakovaně, tak někdy jeden a někdy druhý

Upravíme funkci tak, aby měla jen jedno minimum, ale bude se lišit jen o maličko

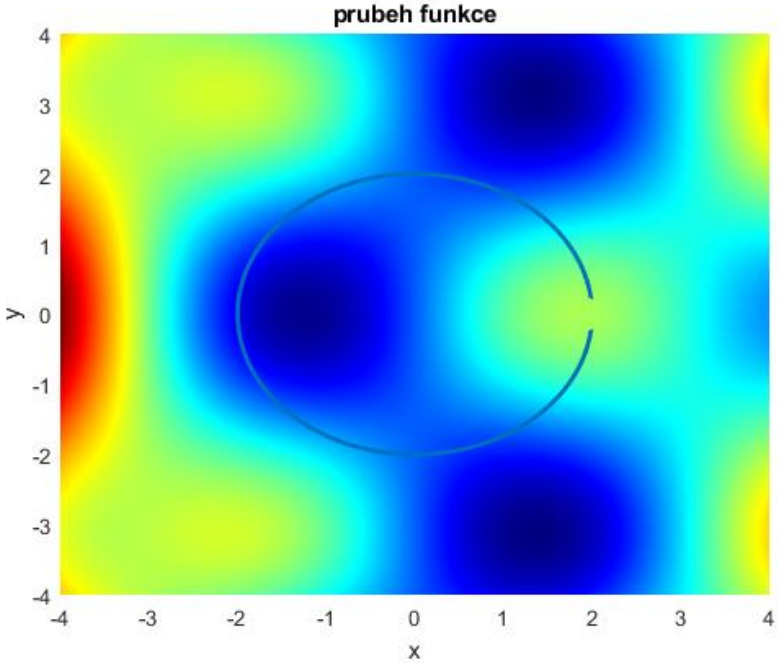
$$f(x,y) = \sin(x)\cos(y) + 0.1x^2 - 0.01x^3 + 0.01y$$



A to už stačí, aby se našlo vždy to správné minimum

Existují i lineární omezení na rovnost, omezení na interval (což jsme v předchozím kroku ošálili použitím lineární omezení na nerovnost), viz help k funkci *ga()*.

Dá se využít i nelineárních omezení, například zkusme hledat minimum naší funkce na kružnici o poloměru 2. K tomu musíme implementovat Matlabovskou funkci, která takovou podmínku definuje.




```
function [vystup, vystup2] =
omezeniNaKruznici(vstup)
% funkce definujici omezeni definicniho
oboru na kruznici
x = vstup(:,1);
y = vstup(:,2);
vystup = x.^2 + y.^2 - 2;  % nerovnost
vystup2 = []; % rovnost - nepouzivame
end

% vlastni optimalizace
x = ga(@mojeFunkce, 2, [], [], [], [],
[], [], @omezeniNaKruznici)
```

```
>> ga_test03

x =

   -1.1973    0.0297
```

Bohatší výstup

Do výstupu můžeme dát více parametrů a získat tak například celou populaci a hodnoty jejího fitness

```
[x, fit, exitflag, output, population, scores] =
ga(@mojeFunkce,2,[],[],[],[],[],[],@omezeniNaKruznici)
```

Nastavení dalších parametrů

Můžeme nastavit další parametry běhu algoritmu pomocí funkce ***optimoptions()***

Defaultní hodnoty

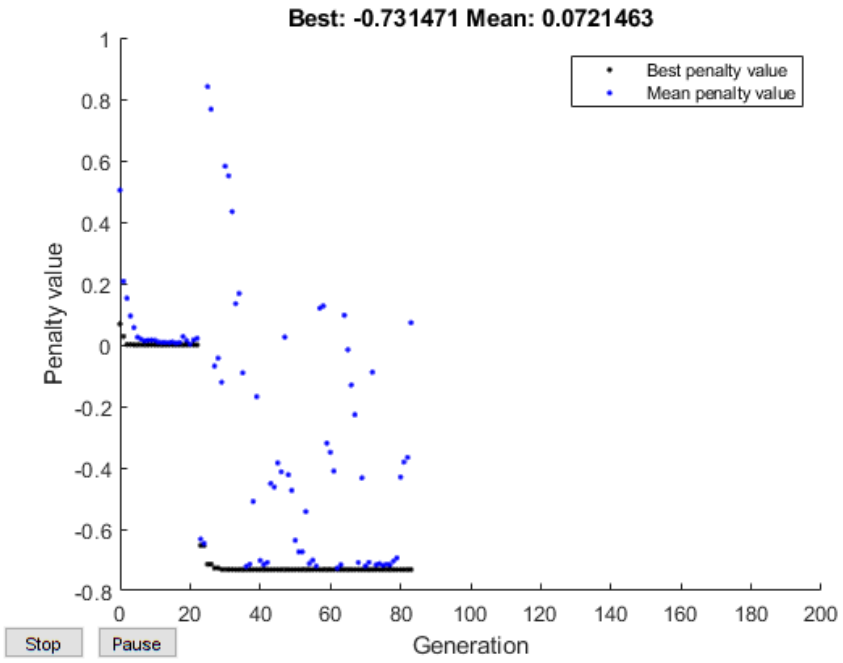
```
options = optimoptions('ga')

options =
   Default properties:
              ConstraintTolerance: 1.0000e-03
                      CreationFcn: @gacreationuniform
                     CrossoverFcn: @crossoverscattered
                CrossoverFraction: 0.8000
                          Display: 'final'
                      EliteCount: '0.05*PopulationSize'
                     FitnessLimit: -Inf
                FitnessScalingFcn: @fitscalingrank
                FunctionTolerance: 1.0000e-06
                         HybridFcn: []
          InitialPopulationMatrix: []
           InitialPopulationRange: []
              InitialScoresMatrix: []
                   MaxGenerations: '100*numberOfVariables'
              MaxStallGenerations: 50
                     MaxStallTime: Inf
                          MaxTime: Inf
                      MutationFcn: {@mutationgaussian  [1]  [1]}
    NonlinearConstraintAlgorithm: 'auglag'
                        OutputFcn: []
                          PlotFcn: []
                  PopulationSize: '50 when numberOfVariables <= 5, else 200'
                  PopulationType: 'doubleVector'
                    SelectionFcn: @selectionstochunif
                     UseParallel: 0
                   UseVectorized: 0
```

- Tolerance, velikost populace, …

  Nastavení změníme pomocí tečkové notace a příslušného parametru

```
options = optimoptions('ga')
options.PopulationSize = 500;
```

- Sledování průběhu optimalizace




```
options = optimoptions('ga','PlotFcn',
@gaplotbestf);

[x, fit, exitflag, output, population,
scores] = ga(@mojeFunkce,2,[],[],[],[],
[],[],@omezeniNaKruznici,1,options)
```

Optimalizace mravenčí kolonií (ACO - Ant Colony Optimization)

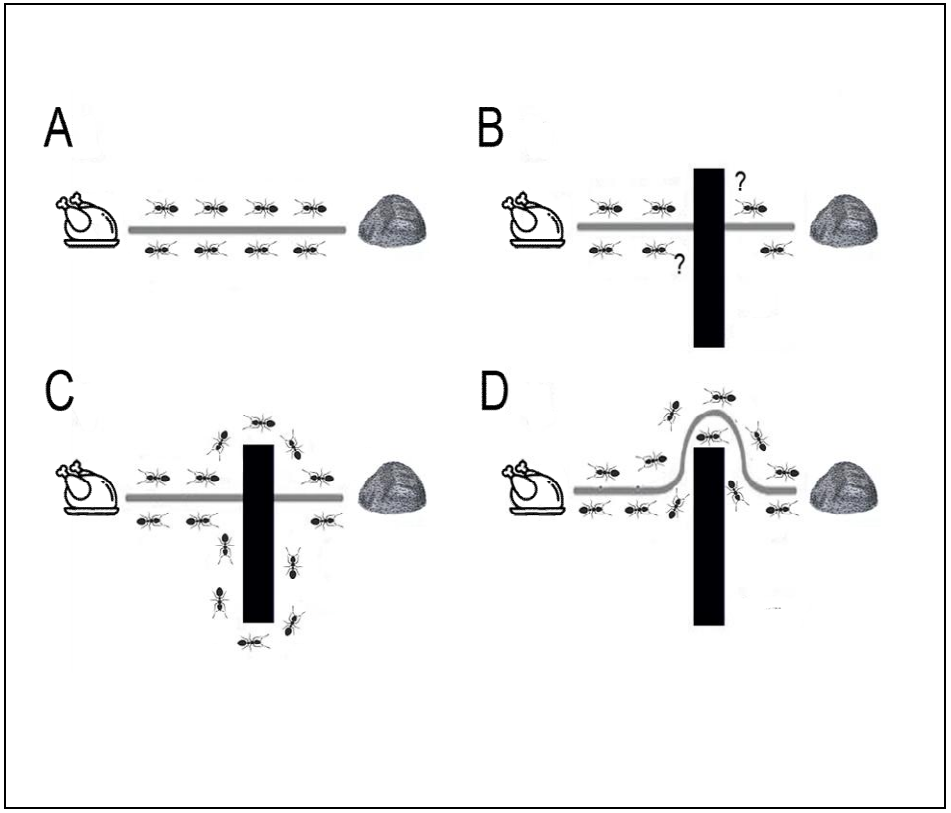
Inspirováno reálným chováním mravenců

A. mravenci chodí po feromonových cestičkách od mraveniště ke zdroji potravy

B. překážka na cestě

C. nalezení cest kolem překážky, pokládání feromonové stopy (dočasné – vypařuje se)

D. nejkratší (nejlepší) cesta má nejsilnější stopu, protože se nestihne tolik vypařit, láká ostatní mravence



Jak to reálně funguje?

- Úloha musí být formulována jako hledání nejkratší cesty v orientovaném grafu
- Jeden „mravenec“ = jedno řešení (jedna cesta)
- Pracujeme s kolonií mravenců (obdoba populace v GA)
- Když mravenec leze z uzlu do uzlu, rozhoduje se podle dvou kritérií
  - Dohlednost (například převrácená hodnota vzdálenosti)
  - Množství feromonu na dané hraně
- Kritéria pouze stanovují pravděpodobnost, kam mravenec poleze
- Když všichni mravenci z kolonie prolezou své cestičky, udělají se dvě věci
  - Položí na své cestičky feromon
  - Feromony ze všech cestiček se trochu vypaří

Jak to reálně funguje – vzorečky

Mravenec ***k*** je v uzlu ***r***. Pravděpodobnost volby uzlu ***s*** je následující:

$$p_k(r,s) = \begin{cases} \dfrac{\tau(r,s)^{\alpha}\,\eta(r,s)^{\beta}}{\sum_{u \in M_k} \tau(r,u)^{\alpha}\,\eta(r,u)^{\beta}} & pro \;\; s \in M_k \\ 0 & jinak \end{cases}$$

Kde

τ je množství feromonu na hraně z uzlu do uzlu (index v závorkách)

η je viditelnost z uzlu do uzlu (index v závorkách, je to například převrácená hodnota vzdálenosti)

α, β mocniny jsou váhové parametry (většinou jedničky)

M_k je množina všech uzlů, dosažitelných z uzlu r (někdy se označuje jako okolí)

Slovně: vezmu všechny možné cesty z daného uzlu, vynásobím viditelnost feromonem a nanormuji (proto je ve jmenovateli ta suma). Pro „nemožné“ cesty z uzlu je pst = 0.

Kolik feromonu ***k***-tý mravenec nechá mezi uzly ***i*** a ***j*** ?

$$\Delta\tau_{i,j}^{k} = \begin{cases} \dfrac{1}{L_k} & pro\ mravence\ k \\ 0 & jinak \end{cases}$$

Kde L_k je délka cesty pro mravence *k*

Feromon pro hranu mezi uzly ***i*** a ***j*** se upraví pro výše uvedené $\Delta\tau_{i,j}^{k}$

$\tau_{i,j}^{k} \leftarrow \tau_{i,j}^{k} + \Delta\tau_{i,j}^{k}$ poznámka: je tam přiřazení

A ještě potřebujeme vztah pro odpařování feromonu

$\tau_{i,j} \leftarrow (1-\rho)\tau_{i,j}$, kde ρ je konstanta z intervalu $\rho \in \langle 0,1 \rangle$ která udává “jak moc” se feromon odpařuje

Příklad 1: problém obchodního cestujícího

Jednoduché zadání: 5 měst

```
x=[82 91 12 92 63];
y=[66 3 85 94 68];
```

Dohlednost (viditelnost): převrácená hodnota vzdálenosti

```
       Inf    0.0157    0.0138    0.0336    0.0523
    0.0157       Inf    0.0088    0.0110    0.0141
    0.0138    0.0088       Inf    0.0124    0.0186
    0.0336    0.0110    0.0124       Inf    0.0257
    0.0523    0.0141    0.0186    0.0257       Inf
```

Počáteční hodnota feromonů – stejná pro všechny hrany, může být jednička, častěji se používá 1/(počet_proměnných . průměrná_vzdálenost)

```
tau =
    0.0394    0.0394    0.0394    0.0394    0.0394
    0.0394    0.0394    0.0394    0.0394    0.0394
    0.0394    0.0394    0.0394    0.0394    0.0394
    0.0394    0.0394    0.0394    0.0394    0.0394
    0.0394    0.0394    0.0394    0.0394    0.0394
```

1. Vnější cyklus přes počet iterací
2. Vnější cyklus přes počet mravenců
3. Vyšleme prvního (i-tého) mravence
4. Náhodně zvolíme počáteční město, např. 5 (odpovídá poslednímu řádku v předchozích maticích). Cesta mravence je tedy [5].
5. Vypočítáme pravděpodobnosti volby dalších uzlů, alfa a beta jsou jedničky
   a. $$\frac{\tau(r,s)^{\alpha}\,\eta(r,s)^{\beta}}{\sum_{u\in M_k}\tau(r,u)^{\alpha}\,\eta(r,u)^{\beta}}$$
6. Napřed čitatele (0.0523 x 0.0394, ….)

```
P =

    0.0021    0.0006    0.0007    0.0010       Inf
```

7. Nemůžu si vybrat město, ve kterém už jsem byl, tedy všechny tyto pravděpodobnosti budou 0

```
P =

    0.0021    0.0006    0.0007    0.0010         0
```

8. Jmenovatel je suma, tedy 0.0021 + 0.0006 + … = 0.0044

9. Po dělení dostanu výsledné pravděpodobnosti volby dalších měst

```
P =

    0.4726    0.1276    0.1680    0.2318         0
```

10. Nyní aplikujeme metodu ruletového kola (další město vybereme náhodně, ale pravděpodobnost výběru je dána vektorem P). Kumulativní pravděpodobnosti jsou

```
C =

    0.4726    0.6002    0.7682    1.0000    1.0000
```

Náhodně vybrané číslo z intervalu <0,1) je 0.9058, takže jako další uzel vybereme město číslo 4. Cesta mravence je teď [5, 4]

11. Nyní pokračujeme z uzlu 4 krokem 5, tedy

```
P =

    0.0013    0.0004    0.0005       Inf    0.0010
```

Vynulujeme všechny uzly, které už cesta obsahuje

```
P =

    0.0013    0.0004    0.0005         0         0
```

Pravděpodobnosti výběru dalšího uzlu jsou nyní

```
P =

    0.5896    0.1926    0.2178         0         0
```

Kumulativní pravděpodobnosti jsou

```
C =

    0.5896    0.7822    1.0000    1.0000    1.0000
```

Náhodné číslo je například 0.127, takže další výběr bude uzel číslo 1, cesta je potom [5, 4, 1]. Dále pokračujeme až je celá cesta hotova.

12. Cestu ohodnotíme (spočítáme délku cesty)

13. Pokračujeme přes všechny mravence v populaci. Po prvním kroku jsme pro 4 mravence dostali následující cesty (všimněte si, že dvě z cest jsou úplně stejné)

5 4 1 3 2, ohodnocení 325.8

1 4 5 3 2, ohodnocení 299.9

1 5 4 3 2, ohodnocení 316.1

1 4 5 3 2, ohodnocení 299.9

14. Nyní upravíme feromon, nejprve přidáme podle aktuálních cest, a potom necháme odpařit. Přidání provádíme v cyklu přes všechny mravence

Připomeňme, že aktuální matice feromonů je

```
tau =
    0.0394    0.0394    0.0394    0.0394    0.0394
    0.0394    0.0394    0.0394    0.0394    0.0394
    0.0394    0.0394    0.0394    0.0394    0.0394
    0.0394    0.0394    0.0394    0.0394    0.0394
    0.0394    0.0394    0.0394    0.0394    0.0394
```

První mravenec má první hranu 5 – 4, ohodnocení cesty prvního mravence je 325.8, takže změna příslušné hrany bude 1/325.8 = 0.003 a nové tau tak bude

```
tau =
    0.0394    0.0394    0.0394    0.0394    0.0394
    0.0394    0.0394    0.0394    0.0394    0.0394
    0.0394    0.0394    0.0394    0.0394    0.0394
    0.0394    0.0394    0.0394    0.0394    0.0394
    0.0394    0.0394    0.0394    0.0425    0.0394
```

Po úpravě tau přes celou cestu prvního mravence pak

```
tau =

    0.0394    0.0394    0.0425    0.0394    0.0394
    0.0394    0.0394    0.0394    0.0394    0.0425
    0.0394    0.0425    0.0394    0.0394    0.0394
    0.0425    0.0394    0.0394    0.0394    0.0394
    0.0394    0.0394    0.0394    0.0425    0.0394
```

A přes všechny mravence

```
tau =

    0.0394    0.0394    0.0425    0.0461    0.0426
    0.0493    0.0394    0.0394    0.0394    0.0425
    0.0394    0.0523    0.0394    0.0394    0.0394
    0.0425    0.0394    0.0426    0.0394    0.0461
    0.0394    0.0394    0.0461    0.0457    0.0394
```

Posledním krokem je odpaření feromonu (které bude stejné všude), například pro ρ = 0,05 to bude

```
tau =

    0.0375    0.0375    0.0404    0.0438    0.0405
    0.0468    0.0375    0.0375    0.0375    0.0404
    0.0375    0.0497    0.0375    0.0375    0.0375
    0.0404    0.0375    0.0405    0.0375    0.0438
    0.0375    0.0375    0.0438    0.0434    0.0375
```

Tím máme hotovou jednu iteraci algoritmu a můžeme pokračovat, dokud nás to nepřestane bavit.

Hlavní smyčka ACO v Matlabu

nVar je počet měst

ant().Tour je cesta - vektor indexů měst

ant().Cost je délka cesty

tau je matice feromonů

eta je matice viditelnosti

```
%% ACO Main Loop
for it=1:MaxIt
    % Move Ants
    for k=1:nAnt
        ant(k).Tour=randi([1 nVar]);
        for l=2:nVar
            i=ant(k).Tour(end);
            P=tau(i,:).^alpha.*eta(i,:).^beta;
            P(ant(k).Tour)=0;
            P=P/sum(P);
            j=RouletteWheelSelection(P);
            ant(k).Tour=[ant(k).Tour j];
        end
        ant(k).Cost=CostFunction(ant(k).Tour);
    end
    % Update Pheromones
    for k=1:nAnt
        tour=ant(k).Tour;
        tour=[tour tour(1)]; %#ok
        for l=1:nVar
            i=tour(l);
            j=tour(l+1);
            tau(i,j)=tau(i,j)+Q/ant(k).Cost;
        end
    end
    % Evaporation
    tau=(1-rho)*tau;
end
```

Další úlohy k implementaci

Kvadratický přiřazovací problém (quadratic assignment problem)

Máme množinu M měst a množinu N továren. Mezi každou dvojicí měst známe jejich vzdálenost. Mezi každou dvojicí továren známe jejich vzdálenost. Mezi každou dvojicí továren známe jejich tok zboží (váha). Cílem je rozmístit továrny tak, aby se minimalizovala suma vzdáleností násobených váhou (proto se úloha označuje jako kvadratická – cost function má v sobě násobení)

Příklady: rozmístění součástek na desce, rozmístění budov v nemocnici, …

Problém batohu (knapsack problém)

Máme batoh s omezenou nosností. Máme řadu předmětů různé váhy a hodnoty. Úlohou je maximalizovat celkovou hodnotu při dodržení omezení hmotnosti na nosnost batohu.

Další metaheuristiky

Existuje jich celá řada, některé rozumné, některé už jsou spíše zábavné a nepřináší ve skutečnosti nic nového.

Používané metaheuristiky

Simulované žíhání (simulated annealing)

Inspirováno procesem žíhání v metalurgii – pomalé chlazení odpovídá pomalému snižování pravděpodobnosti toho, že během prohledávání prostoru možných řešení akceptujeme horší řešení (nejdříve hodně prohledáváme, časem už jen ladíme nejlepší řešení)

Včelí roj (artificial bee colony optimization)

Inspirováno chováním včel ve včelím roji. Včely jsou rozděleny na tři skupiny: dělnice (employed), dohlížitelky (onlookers) a průzkumnice (scouts). Dělnice létají ke zdrojům jídla a přinášejí nektar a zatančí informaci o zdroji. Jakmile se zdroj jídla vyčerpá, stanou se z nich průzkumnice. Dohlížitelky sledují tance dělnic a vybírají zdroje jídla.

Poloha zdroje představuje řešení úlohy, množství nektaru představuje ohodnocení zdroje. Množství dělnic odpovídá množství zdrojů (řešení úlohy).

Hejno částic (partical swarm optimization) je obsaženo v Global Optimization Toolboxu Matlabu

Každé řešení odpovídá jedné částici, a pohyb částic se řídí jednoduchými předpisy pro polohu a rychlost. Pohyb je ovlivňován jednak lokálním okolím částice, a také tím, co zatím našly ostatní částice (s lepším ohodnocením)

V Matlabu implementováno pomocí funkce ***particleswarm()***, použití je hodně podobné funkci ***ga()***, včetně omezení:

```
% vlastni optimalizace
rng default % nastaveni random generatoru
A = [1 0;0 1;-1 0;0 -1];  % linear constraint
b = [4 4 4 4];

x = particleswarm(@mojeFunkce,2,A,b)
```

```
>> ps_test01

Optimization ended: relative change in the objective value over the last
OPTIONS.MaxStallIterations iterations is less than OPTIONS.FunctionTolerance.


x =

    1.3534    3.1416
```

Zábavné metaheuristiky

Harmonické prohledávání - Harmony search (Geem, Kim & Loganathan 2001)

Inspirováno improvizací jazzových hudebníků. Náhodně se vytvoří množina řešení. Nová řešení se tvoří ze všech stávajících (nikoliv jen ze dvou jako u GA), a pokud je nové řešení lepší než dosud nejhorší řešení, je toho nahrazeno.

Hejno světlušek - Glowworm swarm optimization (Krishnanand & Ghose 2005)

Světlušky jsou schopné měnit intenzitu světla, které vyzařují. Intenzita světla odpovídá ohodnocení optimalizované funkce a přitahuje ostatní světlušky, které září méně. Algoritmus zahrnuje také dynamické vyhodnocení okolí – pokud je světluška obklopena dostatečným množstvím jiných světlušek, tak už nevidí světlušky, které jsou vzdálenější. Díky tomu se hejno rozdělí na několik podhejn, kde každé konverguje k lokálnímu extrému optimalizované funkce.

Hejno koček - Cat Swarm Optimization (Chu, Tsai, and Pan 2006)

Inspirováno chováním koček, obdoba mravenčí kolonie. Používá dva módy: prohledávací a sledovací. Prohledávací představuje odpočívající kočku, která se rozhoduje kam se vydat, vybírá se z několika možností náhodně, s větší pravděpodobností u míst, která mají lepší ohodnocení. Ve sledovacím módu se kočka snaží dostat k místu s lepším ohodnocením. Módy se střídají dokud není ohodnocení dostatečně vysoké.

Imperialistický soutěživý algoritmus - Imperialist competitive algorithm (Atashpaz-Gargari & Lucas 2007)

Zatímco GA jsou simulací biologické evoluce, tento algoritmus je simulací lidské sociální evoluce. Náhodně se vygenerují řešení úlohy, zvané Státy. Moc státu odpovídá ohodnocení úlohy. Nejmocnější státy se stanou Imperialisty, začnou přebírat kontrolu nad ostatními státy (Kolonie) a vytvoří počáteční Říše. Operátory algoritmu jsou asimilace (assimilation), revoluce (revolution) a střet (imperialistic competition). V asimilaci se kolonie v Říších snaží stát Říšemi. Revoluce představují výrazné náhle změny, při střetu se Říše snaží získat kolonie nejslabší Říše.

A další hromada metaheuristik

- Shuffled frog leaping algorithm (Eusuff, Lansey & Pasha 2006)
- River formation dynamics (Rabanal, Rodríguez & Rubio 2007)
- Intelligent water drops algorithm (Shah-Hosseini 2007)
- Gravitational search algorithm (Rashedi, Nezamabadi-pour & Saryazdi 2009)
- Cuckoo search (Yang & Deb 2009)
- Bat algorithm (Yang 2010)
- Spiral optimization (SPO) algorithm (Tamura & Yasuda 2011,2016-2017)
- Flower pollination algorithm (Yang 2012)
- Cuttlefish optimization algorithm (Eesa, Mohsin, Brifcani & Orman 2013)
- Colliding bodies optimization (Kaveh and Mahdavi 2014)
- Duelist Algorithm (Biyanto 2016)
- Harris hawks optimization (Heidari et al. 2019)
- Killer Whale Algorithm (Biyanto 2016)
- Rain Water Algorithm (Biyanto 2017)
- Mass and Energy Balances Algorithm (Biyanto 2018)
- Hydrological Cycle Algorithm (Wedyan et al. 2017)
- Emperor Penguins Colony (Harifi et al. 2019)
- Shuffled Shepherd Optimization Algorithm (SSOA) (Kaveh and Zaerreza 2020)
- A mayfly optimization algorithm (MA) (Zervoudakis & Tsafarakis 2020)
- Political Optimizer (PO) (Qamar Askari, Irfan Younas & Mehreen Saeed 2020)
- Forensic-based investigation algorithm (FBI) (JS Chou and NM Nguyen, 2020)
- Jellyfish Search (JS) (JS Chou and DN Truong, 2021)