1. No free lunch theorem

- Neexistuje algoritmus, který by dokázal řešit všechny problémy lépe než jiné algoritmy
- Existuje podmnožina problémů, pro které je algoritmus A lepší než algoritmus B a naopak
- Výběr oběda v restauraci:
  - Restaurace (procedura řešící problémy), menu (řešené problémy), cena (výkon, úspěšnost)
- Evoluční algoritmy jsou ekvivalentní, jejich průměrná úspěšnost je stejná, výkon je ale různý
- Na základě teorému nelze očekávat, že můžeme použít jakýkoliv algoritmus na řešení libovolného problému – různé algoritmy se hodí k řešení různých problémů

2. Single objective and multi-objective optimization: differences, approaches

- Jedna funkce X vícero funkcí
- Funkce mohou sdílet určité parametry - když pak zlepším jednu, můžu tím zhoršit druhou
- U multi se snažím najít co nejlepší poměr výsledků funkcí (paretovka)
- U multi dostávám prostor možných řešení jako výsledek
- Krom paretovky můžu dávat funkcím váhy a hodnotit výsledky pomocí nich

3. Types of test functions

- Unimodální (jedno nejlepší řešení resp. jeden globální extrém)
- Multimodální (vícero možných nejlepších řešení - řešení jsou ekvivalentní)

4. Pareto set

- Paretova hranice – množina bodů, které reprezentují takové kombinace f_i .. f_n, že nelze snížit hodnotu žádné účelové funkce f_n, aniž by se nezvýšila hodnota některých jiných funkcí f_j
- Nejlepší možné řešení optimalizovaného problému
- Aby vznikla paretova hranice, musí být hledané funkce v „konfliktu“, jinak se kardinalita paretovy hranice rovná 1 – jedniné optimální řešení

5. Traveling salesman problem: suitable algorithms

- Genetický algoritmus
- Ant Colony Optimalization

6. Local search algorithms: hill climbing, tabu search, simulated annealing

❖ Hill climbing:

- Umožnuje vzít i horší výsledek
- **Parametry**: (M, x_0, N, f, t_{max})
  - M => prostor řešení
  - x_0 => počáteční řešení
  - N => množina sousedů řešení x (počet generovaných bodů v populaci)
  - f => účelová funkce
  - t_{max} => počet iterací
- **Postup**:
  1. Vygenerujeme náhodného jedince
  2. Pro aktuální řešení generujeme okolí (sousedství) a hledá se lepší/horší hodnota účelové funkce (nejlepší řešení se zaznamenává)
  3. Vybraný nejlepší bod je středem v další iteraci

- **Vylepšení**:
  - *Uživatelsky přijatelná odchylka* – např. sleduje se posledních 10 iterací, pokud se hodnota účelové fce neliší o více, než je nastaveno, tak je alg. ukončen, i když ještě neproběhly všechny iterace
  - *Stochaistic hill climbing* – přidá náhodný element do generace sousedů
  - *Tabu search*

```
t :=1;
random selection of initial solutions x0t ;
x* := x0t;
while t ≤ tmax do
      begin generate N(x0t,σ);
      find xloc ∈ N(x0t,σ) so, that f(xloc) ≥ f(x), for each x ∈ N(x0t,
σ);                                   f(x_loc) := max_{x∈N(x_0t,σ)} f(x)

       if f(xloc) > f(x*);
      then x* := xloc
      t := t +1;
      x0t := xloc;
      end;
 { x* is an approximation of the optimum solution}
```

-

❖ Tabu search

- Horolezecký algoritmus + mechanismus pro snížení nebezpečí zacyklení
- Pamatuje si transformace, podle kterých byl spočítán aktuální střed – neuvízne snadno v lokálním extrému
- **Parametry**: TS = (M, x_0, Theta, f, t_{max}, TL, k)
  - Theta => množina přípustných transformací generující okolí
  - TL => seznam tabu informací
  - k => kapacita krátkodobé paměti TL
- **postup**:
  1. vygenerování náhodného počátečního jedince x
  2. vygenerování okolí kolem x podle thety
  3. pro každého vygenerovaného jedince z okolí se zkontroluje, zda je lepší než x
  4. nejlepší takový jedinec se přidá do TL (fronta, při překročení se odebere první prvek) – dále se jedinec porovnává s jedinci v TL, pokud tam již je, přidá se jiný vhodný
  5. opakování do t_{max}
- **vylepšení**:
  - malá hodnota *k* – velká tendence spadnout do lokálního extrému
  - velká hodnota *k* – hrozí přeskočení nadějných lokálních extrémů
  - dlouhodobá paměť – uchovávání četností (frekvence) použitých transformací

```
x *: = x_0t; {x* is an approximation of the optimum solution}
f_max: = f (x_0t);
TL: = ();
        while t ≤ t_max do
                begin x_loc: = x_0t; f_loc: = f (x_0t);
                        for ∀ Θ ∈ θ do
                                begin y: = θ (x_0t);
                                        if (f (y)> f_loc) and ((θ ∉ T_L) or (f (y)> f_max))
                                        then begin x_loc: = y; f_loc: = f (y); θ_loc: = θ
                                                end
                                end;
                        if f_loc> f_max
                                then begin f_max: = f_loc; x*: = xloc
                                end;
                        if | T_L| < k
                                then T_L: = T_L+ (θ_loc^-1)
                                else T_L:= T_L - (θ_1) + (θ_loc^-1);
                        t := t +1;
                        x_0t: = x_loc
end;
```

-

❖ Simulované žíhání

- Připouští zhoršení hodnoty účelové funkce (překonat lok. extrém) – ze začátku, poté se chová jako hill climb
- **Parametry**: SA = (M, x_0, N, f, T_0, T_f, α, n_T)
  - T_0 => počáteční teplota
  - T_f => konečná teplota
  - n_T => počet opakování metropolisova algoritmu pro danou teplotu T
  - α => funkce redukce teploty, často α = *decr* * T, decr se volí mezi 0,8 a 0,99
- **postup**:
  1. náhodně vybrané počáteční řešení x_0
  2. náhodně vyber x z množiny sousedů N(x_0)
  3. výpočet Δf = f(x) – f(x_0)
  4. pokud je Δf < 0, je akceptováno lepší řešení tzn. x_0 = x
  5. pokud je Δf > 0, zvolíme rand(0,1) < metropolisovo kritérium ($e^{-\frac{\Delta f}{T}}$) – pokud je tato podmínka splněna, přijme se horší řešení, tzn. x_0 = x
  6. ochladíme teplotu alfou – opakování do finální teploty
- **vylepšení**:
  - *paralelizace* – začne se z několika bodů, za výslednou hodnotu se považuje nejvyšší (hledáme max)
  - *elitismus* – uchovává se nejlepší dosažené řešení v průběhu algoritmu
  - generování vice sousedů pro jednoho jedince

```
repeat
        for i: = 1 to n_T do
                begin randomly select x from the set of neighbors N(x_0) solution x_0;
                        Δf: = f (x) - f (x_0);
                        if Δf < 0
                        then x_0: = x {move to a better solution is always accepted}
                                else begin randomly choose r from uniform
                                distribution from (0,1);
                                        if r < e^-Δf/T
                                        then x_0: = x {move to a worse solution or
                                        current solution will remain unchanged}
                                end
                end
        T: = α(T);
until T < T_f; {crystallization annealing}
```

-

7. Evolution strategy: principle, variants

- ES používají reprezentaci jedinců v oboru reálných čísel, GA v binární
- ES nepoužívaly operátory křížení, používaly operátory selekce a mutace
- Nastavuje se parametr FV – vhodnost, při jejímž dosažení se ES zastaví
- Značení ES (+) a (,)
  - \+ do nové populace se vybírají nejlepší z rodičů i potomků
  - , do nové populace se vybírají nejlepší pouze z potomků

❖ (1+1) – ES (dvoučlenné ES)

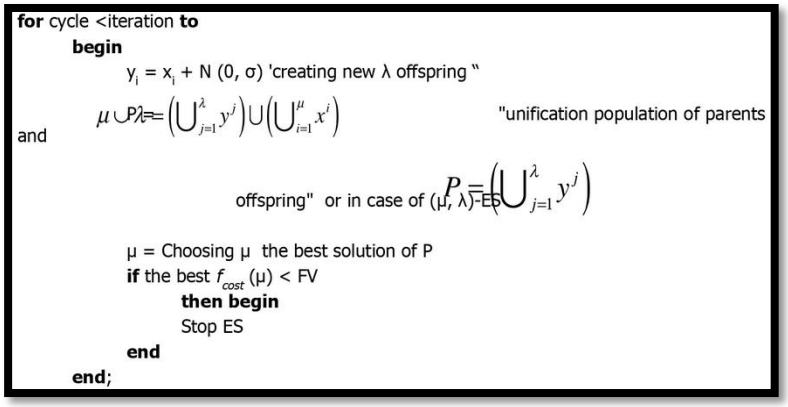
- Nejjednodušší verze, pracuje se pouze s jedním jedincem – rodičem, pomocí Gaussova mutačního operátoru se z něj vytváří nový potomek
  - K rodiči se přičte náhodné číslo, vygenerované **normálním rozdělením N(0, sigma)**
  - **Sigma** – směrodatná odchylka, pro její určení lze aplikovat různá pravidla (např. jestli se jedná o lineární funkce apod.)
- **Pravidlo jedné pětiny**
  - Poměr úspěšných mutací ke všem mutacím by měl být **1/5**
  - **Úspěšná mutace** – **potomek** má **lepší fitness** než **rodič**

```
for loop < iteration do
begin
    y = x + N(0, σ)  „new offspring"
    if f_cost (y) < f_cost (x)
        then begin
        x = y;
        end
    if f_cost (x) < FV
        then begin
        Stop ES
        end
        end;
```

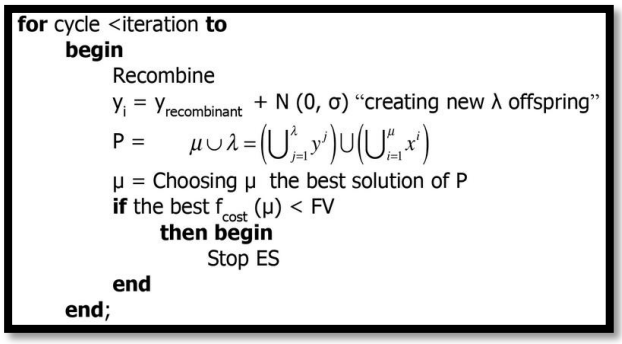
-

❖ (μ+ λ)-ES a (μ, λ)-ES (vícečlenné ES)

- **Mý (μ)** – populace rodičů
- **Lambda (λ)** – populace potomků
- Práce s populacemi
- **ALG**
  - Z každého rodiče se vytvoří nový potomek (viz 1+1) a sjednotí se lamda a mý (sjednocení populace rodičů a potomků)
  - Dojde k výběru nejlepších řešení ze sjednocení, pokud obsahuje lepší, než stanovenou fitness value, tak konec
  - **ES s elitismem** – do nové rodičovské populace jsou vybíráni jak rodiče, tak potomci na základě dosažené fitness
  - (mý + lambda)-ES někdy stagnovala -> tak (my, lambda) bez elitism



- ❖ Rekombinační ES
  - Před mutací (vytvoření potomka pomocí N(0, sigma) je vytvořen recombinant (předpotomek) z více rodičů (dochází ke křížení)
  - **Rho** udává počet rodičů



- ❖ Adaptivní ES
  - Adaptace směrodatné odchylky mezi iteracemi
  - Oblast optimálního řešení nemusí být předem známa, algoritmus sám upravuje své řídící parametry

8. The normal distribution in evolutionary algorithms

- sdf

9. Evolutionary algorithms - typical phases, examples of algorithms

- Vymezení parametrů, stanovení účelové funkce
- Generace prvotní populace jedinců
- Ohodnocení jedinců
- Výběr rodičů
- Křížení rodičů = tvorba potomků
- Mutace potomků
- Opět ohodnocení
- Výběr jedinců z rodičů a potomků
- Přepsání staré generace na novou
- Opakujeme od výběru rodičů dále
- Diferenčka, ACO, SOMA??? Geneťák tam nepatří

10. Swarm intelligence - typical phases, examples of algorithms

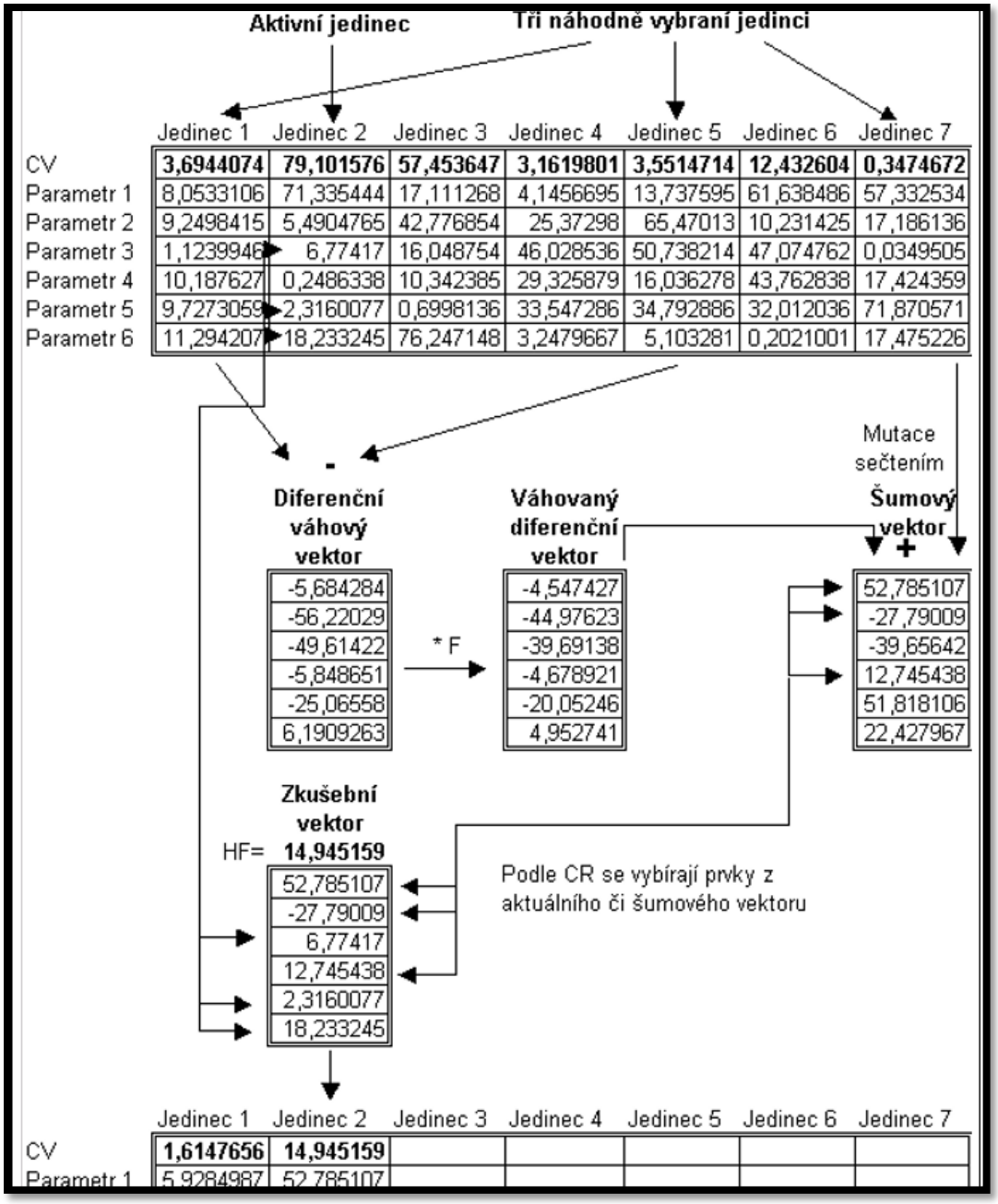
- Nemá evoluční operátory
- Řídí se pohybem dalších jedinců
- SOMA, PSO, Firefly
- Náhodně se vygeneruje populace
- Ohodnocení jedinců
- Výběr směru a rychlosti pro každého jedince
- Ohodnocení a update nejlepšího řešení (leader, pbest, gbest..)

11. Genetic algorithm - types of crossover

- Rozdělení v určitém bodě a prohození částí
- Rozseknu ve 2 bodech - prohodí se střed
- Rozseknu v X bodech - náhodně vyberu části, které se prohodí

12. Differential evolution - mutation vector, trial vector, target vector, mutation strategies, crossover, natural selection, control parameters

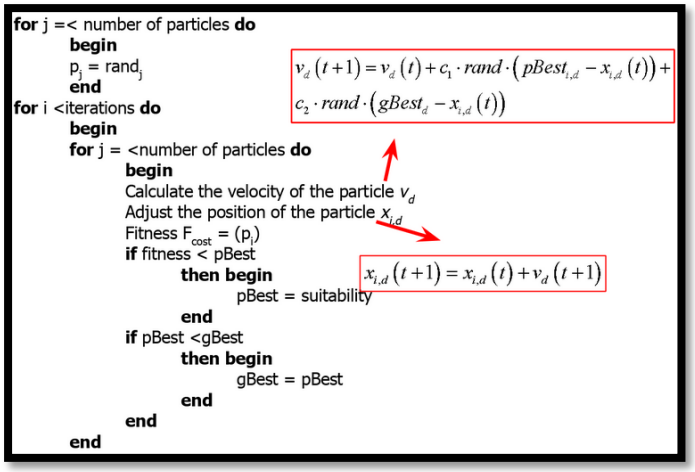
- Nejdřív mutace, pak křížení
- CR - práh křížení, čím větší tím spíš se provede
- Mutace probíhá spojením 3 náhodně vybraných rodičů Xnew = X3 + F* (X2 – X1) - Xnew jak pak tzv. Šumový vektor
- F – mutační konstanta (0,2)
- Ze šumového vektoru a vybraného jedince (target vector) se pak vytvoří zkušební vektor



13. Particle swarm optimization - pBest, gBest, velocity, inertia weight, equation of movement, control parameters

- První populace je náhodná
- Každý jedince má pak svůj vektor rychlosti (určuje směr a rychlost = velikost kroku)
- GBest - nejlepší hodnota celkově, globální proměnná, kterou zná každý jedinec
- PBest - nejlepší hodnota konkrétního jedince
- Inertia weight (setrvačnost) - násobí se tím v následujícím vzorci současná rychlost, malá inertia (pod 1) – bude to směřovat k lokálním extrémům, velká inertia – bude to směřovat ke globálním
- Výpočet rychlosti: vNow + c1 * rand *(pBest – pNow) + c2 * rand * (gBest – pNow)
- Výpočet nové pozice: současná pozice + nová rychlost
  - c1 a c2 - učící faktory (mezi 0 a 4, obvykle 2)
  - PNow - současná pozice

  - VNow - současná rychlost
  - c1 – upřednostňuje návrat na svou nejlepší pozici
  - c2 – upřednostňuje posun k gBest
- Částice se mohou ubírat třemi směry:
  - Individuální – pokračují svou vlastní cestou
  - Konzervativní – vracejí se na svou dosud nejlepší pozici
  - Přizpůsobivý – následují nejlepšího jedince
- Vylepšení:
  - Sousedství – omezení rizika předčasné konvergence částic do lokálních extrémů, jedinci jsou rozděleni do skupin – sousedství, nejčastější topologií je kruh, rozdělení:
    - Geografické – částice ve stejné oblasti
    - Sociální – jedinci jsou sousedi nehledě na to, kde se nacházejí



14. Self-organizing migrating algorithm - leader, strategies, equation of movement, control parameters

- Leader – jedince s nejlepším fitness v dané generaci
- AllToOne - všichni jedinci směřují k leaderovi (nejlepší hodnotě)
- AllToAll - všichni jedinci si vyzkouší směr ke každému z dalších jedinců, zapamatují si nejlepší bod, který našli, na konci migrace se pak přesunou do nových bodů
- AllToAll Adaptive - na rozdíl od normálního ATA se body přesouvají ihned (změna se tedy projeví hned v současné migraci místo až v té následující)
- AllToOne Random - náhodně vybraný leader
- Ve větším počtu jedinců je možnost rozdělit je do subpopulací (budu mít víc leaderů)
- Path length – o kolik překročí jedinec leadera při cestě za ním (míň jak 0 – nedojde k němu, 1 dojde přesně na jeho pozici, 2 půjde za něj o stejnou vzdálenost jako byl od něj apod.)
- Step – velikost kroku (doporučuje se 0.11, max path length)
- Perturbační vektor (mutace) – [0, 1] určí jestli se bude jedince pohybovat přímo k leaderovi nebo ne (může se pohybovat jen po ose X např.)
- D - počet dimenzí
- Migrace – počet migrací/přesunutí populace, ukončovací parametr
- PopSize – velikost poulace, u multimodální funkce lze nastavit popsize = D
- MinDiv – definuje, jaký maximální rozdíl mezi nejhorším a nejlepším jedincem v aktuální populaci je povolen, jestliže je rozdíl menší než mindiv, je alg. ukončen
- Je třeba řešit i krok mimo vymezené hranice - např. vygenerováním nového random jedince
- Postup:
  1. Je vytvořena náhodná populace a zvolen leader

2. Jedinci se pohybují směrem k leaderovi skoky (param. Step), po každém skoku se přepočítá hodnota účel. fce, je-li lepší než předchozí pozice, zapamatuje si ji, skoky jedince pokračují, dokud nedosáhne pozice dané param. pathLength
   - Skok se počítá: jmp = x_{start} + (x_{leader} – x_{start}) * step * PRTvector, x.. pozice
3. po ukončení běhu se jedinec vrátí na pozici, kde byla nalezena nejlepší hodnota účel. fce během jeho cesty => všichni jedinci, mimo leadera, jsou přemístěni
4. PRTvector – je generováno náh. Číslo (pro každou d), pokud je číslo > než prt, pak je vektor nastaven na 0, v opačném případě na 1 – má velký vliv na výsledný pohyb jedince

```
vstup:
    x: prvopočáteční náhodně vygenerovaná populace rodičů;
    Řídicí a ukončovací parametry algoritmu - viz Tab. 8.3
    f_cost: účelová funkce vracející vhodnost aktuálního řešení
    Specimen: vzorový jedinec
    for i < Migrace do
                begin
                selekce nejlepšího jedince – Leadera
                        for j ≤ PopSize do
                        vyber j-tého jedince
                        vypočítej f_cost nové pozice z (8.23)
                        zapiš nejlepšího nalezeného řešení do nové populace
                        end
                if MinDiv < |nejlepší_jedinec - nejhorší_jedinec|
                        then begin
                        Stop SOMA
                        end
                end
```

15. Ant colony optimization - probabilities of movement, recalculation of pheromones

- Vybírá cestu na základě pravděpodobnostní matice
- Pravděpodobnost určuje vzdálenost bodu a množství feromonu, obě tyto hodnoty jsou regulovány parametry alfa (feromon) a beta (vzdálenost)
- Feromon se pak zvětšuje o 1/vzdálenost mezi body
- Po každé migraci se feromony zmenší (vypaří) o násobek koeficientu
- Může se korigovat způsob přidávání feromonu (např. Pro nejlepší výsledek za migraci)
- Parametry:
  - α – stupeň důležitosti feromonu
  - β – stupeň důležitosti vzdálenosti
- vzorec: ($\tau(r, s)^{\alpha}$ * $\eta(r, s)^{\beta}$) / celková suma feromonu a vzdálenosti (r – aktuální uzel, s – uzel kam mířím)

- 5 cities
- Distance matrix for these cities is:

0	10	12	11	14
10	0	13	15	8
12	13	0	9	14
11	15	9	0	16
14	8	14	16	0

- Matrix of inverse distances (visibility matrix):

0	0.1000	0.0833	0.0909	0.0174
0.1000	0	0.0769	0.0667	0.1250
0.0833	0.0769	0	0.1111	0.0714
0.0909	0.0667	0.1111	0	0.0625
0.0714	0.125	0.0714	0.0625	0

- 5 cities, city 1 is a departure city. Since city 1 is chosen as the beginning , it cannot be chosen again → visibility of the city is **0**

- Initial pheromone matrix

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

- Visibility matrix

0	0.1000	0.0833	0.0909	0.0174
0	0	0.0769	0.0667	0.1250
0	0.0769	0	0.1111	0.0714
0	0.0667	0.1111	0	0.0625
0	0.125	0.0714	0.0625	0

- We calculate the posibility to visit other city using the formula

$$p_k(r,s) = \begin{cases} \frac{\tau(r,s)^{\alpha}\eta(r,s)^{\beta}}{\sum_{u\in M_k}\tau(r,u)^{\alpha}\eta(r,u)^{\beta}}, for\ s\epsilon M_k \\ 0, otherwise \end{cases}$$

$\tau(1,s)^1\eta(1,s)^2$:

- $1 * 0.1000^2 = 0.0100$
- $1 * 0.0833^2 = 0.0069$
- $1 * 0.0909^2 = 0.0083$
- $1 * 0.07142^2 = 0.0051$

$$\sum_{s\in M_k} \tau(1,s)^1\eta(1,s)^2 = 0.0303$$

- Visibility matrix

0	0.1000	0.0833	0.0909	0.0174
0	0	0.0769	0.0667	0.1250
0	0.0769	0	0.1111	0.0714
0	0.0667	0.1111	0	0.0625
0	0.125	0.0714	0.0625	0

[1]

$\tau(1,s)^1\eta(1,s)^2$:

- $1 * 0.1000^2 = 0.0100$
- $1 * 0.0833^2 = 0.0069$
- $1 * 0.0909^2 = 0.0083$
- $1 * 0.07142^2 = 0.0051$

$$\sum_{s\in M_k} \tau(1,s)^1\eta(1,s)^2 = 0.0303$$

- Visibility matrix

0	0.1000	0.0833	0.0909	0.0174
0	0	0.0769	0.0667	0.1250
0	0.0769	0	0.1111	0.0714
0	0.0667	0.1111	0	0.0625
0	0.125	0.0714	0.0625	0

The probabilities:

City 1 to 2: $\frac{0.01}{0.0303} = 0.3299$

City 1 to 3: $\frac{0.0069}{0.0303} = 0.2291$

City 1 to 4: $\frac{0.0083}{0.0303} = 0.2727$

City 1 to 5: $\frac{0.0051}{0.0303} = 0.1683$

[1]

- The probabilities:

City 1 to 2: $\frac{0.01}{0.0303} = 0.3299$ City 1 to 4: $\frac{0.0083}{0.0303} = 0.2727$

City 1 to 3: $\frac{0.0069}{0.0303} = 0.2291$ City 1 to 5: $\frac{0.0051}{0.0303} = 0.1683$

- The cumulative numbers of these probabilities:
  - City 2: 0.3299
  - City 3: 0.5590
  - City 4: 0.8317
  - City 5: 1

- The cumulative numbers of these probabilities:
  - City 2: 0.3299
  - City 3: 0.5590
  - City 4: 0.8317
  - City 5: 1
- Generate random number $r \in [0,1]$: suppose we have generated number $r = 0.6841$
- Compare r with cumulative numbers:
  - $\mathbf{0.5590 < r < 0.8317}$ ⇒ an ant will visit the **city 4**

- The city 4 was visited ⇒ the visibility matrix must be adjusted:

0	0.1000	0.0833	**0**	0.0174
0	0	0.0769	**0**	0.1250
0	0.0769	0	**0**	0.0714
0	0.0667	0.1111	**0**	0.0625
0	0.125	0.0714	**0**	0

- Now, the proces of the calculation of the probability of visiting the neighbor city will be repeated

$\tau(4,s)^1\eta(4,s)^2$:

- $1 * 0.0667^2 = 0.0044$
- $1 * 0.1111^2 = 0.0123$
- $1 * 0.07142^2 = 0.0039$

$$\sum_{s \in M_k} \tau(1,s)^1\eta(1,s)^2 = 0.0207$$

- Visibility matrix

0	0.1000	0.0833	0	0.0174
0	0	0.0769	0	0.1250
0	0.0769	0	0	0.0714
0	0.0667	0.1111	0	0.0625
0	0.125	0.0714	0	0

The probabilities:

City 4 to 2: $\frac{0.0044}{0.0207} = 0.2147$ City 4 to 5: $\frac{0.0039}{0.0207} = 0.1887$

City 4 to 3: $\frac{0.0123}{0.0207} = 0.2291$

- The cumulative numbers of these probabilities:
  - City 2: 0.2147
  - City 3: 0.8113
  - City 5: 1
- Generate random number $r \in [0,1]$: suppose we have generated number $r = 0.4024$. Compare r with cumulative numbers:
  - $\mathbf{0.2147 < r < 0.8113}$ ⇒ an ant will visit the **city 3**
- For now, we have path $1 \rightarrow 4 \rightarrow 3$

- This processs is repeated until all ants have their own paths
- Remember: Each ant starts from another city
- Suppose that we have the paths as follows:
  - Ant 1: $1 \rightarrow 4 \rightarrow 3 \rightarrow 5 \rightarrow 2 \rightarrow 1$ Total distance: 52
  - Ant 2: $1 \rightarrow 4 \rightarrow 2 \rightarrow 5 \rightarrow 3 \rightarrow 1$ Total distance: 60
  - Ant 3: $1 \rightarrow 4 \rightarrow 5 \rightarrow 2 \rightarrow 3 \rightarrow 1$ Total distance: 60
- Use total distances (objective function evaluation) to recalculate the pheromones on the edges:

$$\tau_{r,s} \leftarrow (1-\rho)\tau_{r,s} + \sum_{k=1}^{N} \Delta\tau_{r,s}^{k}$$

[1]

- Use total distances (objective function evaluation) to recalculate the pheromones on the edges:

$$\tau_{r,s} \leftarrow (1-\rho)\tau_{r,s} + \sum_{k=1}^{N} \Delta\tau_{r,s}^{k}$$

ρ... evaporation coefficient equals to 0.5

- Pheromone matrix recalculated based on the Ant 1:

0.5000	0.5000	0.5000	0.5192	0.5000
0.5192	0.5000	0.5000	0.5000	0.5000
0.5000	0.5000	0.5000	0.5000	0.5192
0.5000	0.5000	0.5192	0.5000	0.5000
0.5000	0.5192	0.5000	0.5000	0.5000

$$\Delta\tau_{r,s}^{k} = \frac{Q}{f(s)} = \frac{1}{52}$$

[1]

- Use total distances (objective function evaluation) to recalculate the pheromones on the edges:

$$\tau_{r,s} \leftarrow (1-\rho)\tau_{r,s} + \sum_{k=1}^{N} \Delta\tau_{r,s}^{k}$$

ρ... evaporation coefficient equals to 0.5

- Pheromone matrix recalculated based on the Ant 2:

0.5000	0.5000	0.5000	0.5359	0.5000
0.5192	0.5000	0.5000	0.5000	0.5167
0.5167	0.5000	0.5000	0.5000	0.5192
0.5000	0.5167	0.5192	0.5000	0.5000
0.5000	0.5192	0.5167	0.5000	0.5000

$$\Delta\tau_{r,s}^{k} = \frac{Q}{f(s)} = \frac{1}{60}$$

- Use total distances (objective function evaluation) to recalculate the pheromones on the edges:

$$\tau_{r,s} \leftarrow (1-\rho)\tau_{r,s} + \sum_{k=1}^{N} \Delta\tau_{r,s}^{k}$$

ρ... evaporation coefficient equals to 0.5

- Pheromone matrix recalculated based on the Ant 3:

0.5000	0.5000	0.5000	0.5526	0.5000
0.5192	0.5000	0.5167	0.5000	0.5167
0.5334	0.5000	0.5000	0.5000	0.5192
0.5000	0.5167	0.5192	0.5000	0.5167
0.5000	0.5359	0.5167	0.5000	0.5000

$$\Delta\tau_{r,s}^{k} = \frac{Q}{f(s)} = \frac{1}{60}$$

16. Firefly algorithm - the movement of fireflies, equation of movement, control parameters

- Porovnám vzdálenost mezi každými dvěma jedinci - horší z nich se pak pohybuje směrem k tomu lepšímu podle vzorce
- Xi + (beta * (1/(omega + r)) * (Xj – Xi)) + alfa*(random – 0.5)
- Omega = nastavuje se na malou hodnotu jen aby nedošlo k dělení nulou
- Světlušky jsou po každé migraci seřazeny od nejlepší po nejhorší, tím pádem se mi spíš budou pohybovat horší k lepším a lepší se budou pohybovat více náhodně (tj, pokud není nalezena žádná atraktivnější světluška, je pohyb náhodný)
- https://homel.vsb.cz/~ska206/bia.html

Algorithm 1. Pseudo code for the standard FA.

```
begin
  generate initial population X_i, i = 1...popSize;
  evaluate fitness values f(X_i), i = 1...popSize;
  while evalMax not reached do
    for (i = 1; i ≤ popSize; i++) do
      for (j = 1; j ≤ popSize; j++) do
        if f(X_j) > f(X_i) then
          move X_i towards X_j in all directions;
        end
        attractiveness varies with dist. r via e^{-γr^ε}
        evaluate new solutions and update light intensity
      end
    end
    rank fireflies and find the current best
end
results and visualization
end
```

**evalMax:* maximum number of function evaluations
**popSize:* population size

-

17. Teaching-learning based algorithm - teaching phase, learning phase, teacher, mean, learner, control parameters

- 2 fáze teachnig a learning
- Teaching fáze - vypočte se střední hodnota všech jedinců, pozice každého jedince se pak upraví pomocí této střední hodnoty
- Xnew = Xold + rand * (Xteacher – Tf * Xstř), Tf = round (+rand(0,1)) - teacher factor – je buď je 1 nebo 2
- Xnew se přijme pouze pokud je lepší jak Xold
- Learner fáze - jedince náhodně vybere dalšího jedince a upraví se pomocí vzorce
- Xnew = Xold + rand * (Xoldlepší - Xoldhorší)
- Opět se přijme pouze pokud je lepší

18. Diversity of population - positive and negative influences. How to preserve the diversity of the population. Premature convergence

- Diverzita = různorodost populace
- Malá diverzita může vést k elitizmu
- Čím větší diverzita, tím větší prostor prohledávám - může se zpomalit algoritmus, ale díky tomu mám menší šanci že se seknu v lokálním extrému
- Diverzita se uchovává většinou pomocí řídících parametrů, případně můžu za určité podmínky (např. když se mi jedinci sbíhají v jednom místě) pro ty jedince vygenerovat nové náhodné pozice
- Premature convergence – jedinci se začnou sbíhat v lokálním extrému, díky nedostatečně prohledanému prostoru (můžu mít např. funkci, která má 2 velké extrémy daleko od sebe na jinak relativně rovné ploše)

19. Elitism and its influence on the algorithm convergence

- Vybírám pouze nejlepší (nebo nejhorší) jedince, mám pak malou diverzitu a můžu se tak velice rychle dostat k lokálním extrémům
- Jedinci se pak budou rychleji sbíhat k sobě

20. Complexity of algorithms

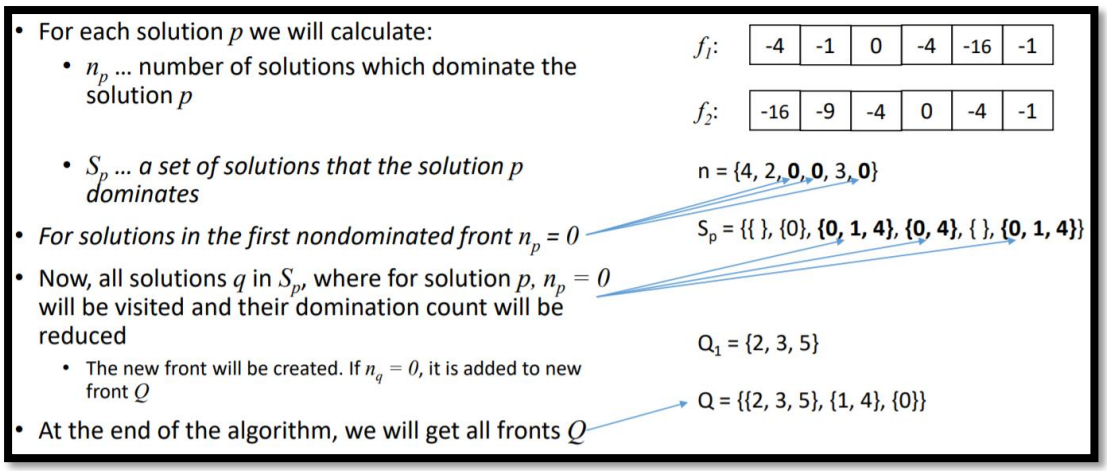
-

21. Number of evaluations of the objective function

- Bere se jako ohodnocení kvality algoritmu pro funkci. Čím méně ohodnocení je potřeba pro dosažení výsledku tím lépe.
- Můžu mít algoritmus, který najde výsledek např. v 5 generacích, ale během jednoho cyklu provede velké množství ohodnocení funkce což se bere jako nejvíc výpočetně náročná operace

22. NSGA II: Principle, fast non-dominated sorting

- Pro každé řešení si uložím počet řešení které mu dominuje n = {4,2,0,0,3,0}
- Pro každé řešení si uložím kterým řešení dominuje Sp{{},{0},{0,1,4},{0,4},{},{0,1,4}}
- Pokud jsou v n nějaké 0, odpovídající řešení vytvoří první rank, ty řešení pak dále ignoruju - Q1 = {2,3,5}
- Navštívím každé q v Sp kterým v n odpovídá 0 – v tomto příkladu to bude {0,1,4}, {0,4} a {0,1,4}
- V n pak ty navštívené snížím o 1 – pro tenhle příklad 4 snižím o 3, 2 snížím o 2 a 3 snížím o 3 – to kde jsou v n 0
- Získám v n další 0 a odpovídající řešení mi vytvoří další rank Q2 = {1,4}
- Nakonec budu mít všechny ranky Q = {{2,3,5},{1,4},{0}}
- Fitness Crowdin – na základě vzdálenosti se pak setřizují řešení, jako nejlepší beru ty, které jsou od sebe nejdál vzdálené (dám jim tak vyšší ohodnocení)



-