

Slovenská technická univerzita v Bratislave  
**FAKULTA INFORMATIKY A INFORMAČNÝCH  
TECHNOLÓGIÍ**

Študijný program: Softvérové inžinierstvo

---

# Evolučné algoritmy

Case Study

**Téma 18. Evolúcia sortovacích sietí**

Vypracoval: Bc. Michal Lokša, 5. ročník, SI  
Letný semester 2005/2006  
E-mail: [lokator@gmail.com](mailto:lokator@gmail.com)

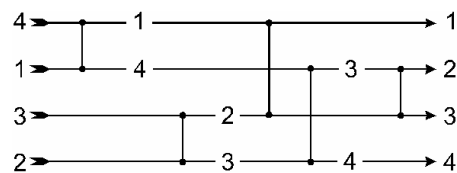
# Obsah

Zadanie .....	1
Použité skratky .....	1
Genetický algoritmus .....	2
Fitness.....	2
Algoritmus.....	2
Programové riešenie .....	2
Výsledky.....	3
Koevolučný genetický algoritmus.....	4
Fitness sortovacích sietí .....	4
Fitness permutácií čísel .....	5
Algoritmus.....	5
Programové riešenie .....	5
Výsledky.....	6
Záver.....	7
Zoznam použitej literatúry .....	7

## Zadanie

Vodorovné línie označujú pozície v sekvencii, vertikálne spoje dvoch horizontálnych línií reprezentujú operáciu výmeny zle usporiadanej dvojice (tzv. komparátor). Komparátor porovnáva dve čísla z pozícií určených horizontálnymi líniami, a keď sú zle usporiadané, prehodí ich vzájomné pozície. Keď chceme vzostupnú sekvenciu a "horné" číslo je väčšie ako "dolné", prehodíme ich navzájom.

- Príklad: pre hodnoty 4,1,3,2 získaj 1,2,3,4 alebo 4,3,2,1 ako výstup: vertikálna čiara označuje výmenu hodnôt medzi horizontálnymi čiarami, keď je prvá hodnota väčšia/menšia ako tá druhá.
- Vyskúšaj: veľkosť populácie=10, kríženie=10%, mutácie=0%
- Vyskúšaj: veľkosť populácie=10, kríženie=10%, mutácie=1% (200 generácií)
- Vyskúšaj: veľkosť populácie=10, kríženie=80%, mutácie=1% (150 generácií)
- Vyskúšaj: veľkosť populácie=50, kríženie=90%, mutácie=1%



Príklad triediacej siete pre testovací príklad - sekvenciu (4,1,3,2), ktorá je korektne usporiadaná na sekvenciu (1,2,3,4). Táto sieť môže byť popísaná ako sekvencia komparátorov [1:2],[3:4],[1:3],[2:4],[2:3], kde čísla v zátvorkách sú indexy spojených línií, na ktorých výmena prebieha.

GA vytvára napr. pre prípad zoradenia piatich čísel 5-vstupové sortovacie siete obsahujúce najviac 9 porovnaní-výmen. Existuje 120 fitness prípadov (všetkých 5! permutácií sekvencie 1,2,3,4,5), a korektná sieť ich všetky zoradí bezchybne. Vyskúšaj koevolúciu pre zložitejšie prípady, kde sekvencie hodnôt sa vyvíjajú nezávisle a sú ohodnotené tým lepšie, čím viac sietí na nich zlyháva.

## Použité skratky

- N počet triedených čísel
- M počet porovnaní
- P<sub>S</sub> veľkosť populácie sortovacích sietí
- P<sub>P</sub> veľkosť populácie permutácií
- T číslo generácie
- T<sub>max</sub> maximálne číslo generácie

## Genetický algoritmus

Genetické algoritmy sa v súčasnosti používajú v najrozličnejších vedecko-technických aplikáciách. Jeho implementácia je odvodená zo všeobecného evolučného algoritmu, kde je operátor reprodukcie špecifikovaný tak, že obsahuje dva časti: časť kríženia a časť mutácie. Výber chromozómov do reprodukčného procesu je realizovaný pomocou rulety. Problém zastavenia genetického algoritmu je riešený tak, že počet generácií je zhora ohraničený číslom  $T_{\max}$ . Druhým prípadom kedy môže byť algoritmus ukončený je vtedy, keď je nájdené jedno zo správnych riešení.

### ***Fitness***

Každý jedinec (sieť) v populácii zoraďuje každú permutáciu  $N$  čísel, t.j. počet ohodnotení je  $N!$ . Jeho fitness sa rovná počtu bezchybne zoradených permutácií. Maximálna fitness je  $N!$ .

### ***Algoritmus***

1. Vygenerovanie náhodnej populácie sietí
2. Ak  $T=T_{\max}$ , tak vyhlás neúspech a skonči algoritmus
3. Triedenie všetkých  $N!$  permutácií každým jedincom z populácie sietí
4. Vytvorenie novej populácie sietí ruletovým výberom (kríženie+mutovanie)
5. Ak nebolo nájdené riešenie tak skoč na 2

### ***Programové riešenie***

Programové riešenie je založené na objektoch z ktorých sa skladajú zložitejšie objekty.

#### **TCompare**

Objekt, ktorý tvorí jedno porovnanie v sortovacej sieti. Obsahuje dva indexy, ktoré udávajú, ktoré čísla sa majú porovnávať. Objekt obsahuje metódy na mutáciu a samotné porovnanie príslušných číslíc v permutácii. Mutovanie tohto objektu neumožňuje vygenerovanie takých dvojíc indexov, ktoré by sa rovnali. Keďže porovnanie čísla so samým sebou nemá význam a pretože by to zmenšovalo šance na nájdenie najlepšieho riešenia, tak boli takéto indexy zakázané.

#### **TSort**

Objekt, ktorý sa skladá z  $M$  objektov TCompare. Objekt je schopný sám sa mutovať, krížiť, triediť permutácie a vypočítavať fitness.

#### **TPermut**

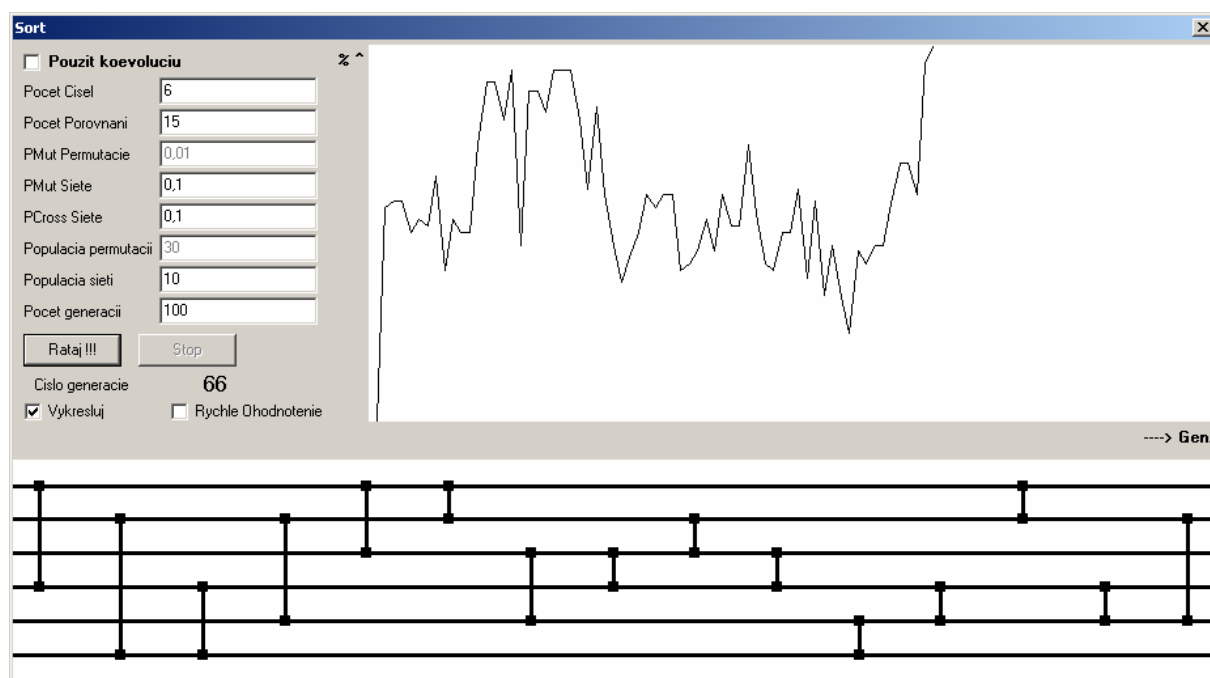
Objekt, ktorý uchováva v sebe permutáciu čísel. Je schopný postupne generovať všetky permutácie  $N$  číslíc.

#### **TGA**

Objekt zapuzdrujúci genetický algoritmus. Obsahuje populáciu sietí a množinu všetkých  $N!$  permutácií, ktoré sa triedia sortovacími sieťami. Objekt má vstupné parametre: počet číslíc  $N$ , počet porovnaní  $M$ , pravdepodobnosť kríženia a mutácie populácie a samotnú veľkosť populácie  $P_s$ .

V dialógovom okne aplikácie (Obr. 1) sa cez príjemné užívateľské prostredie nastavujú príslušné parametre. Po stlačení tlačidla „Rátať“ sa spustí genetický algoritmus, ktorý na

pravej strane obrazovky kreslí graf, kde sa nachádza na vodorovnej osi číslo generácie algoritmu a na zvislej osi percento zatriedených permutácií najlepším jedincom z populácie. V spodnej časti okna je graficky nakreslený najlepší jedinec z populácie.



Obr. 1 Aplikácia

## Výsledky

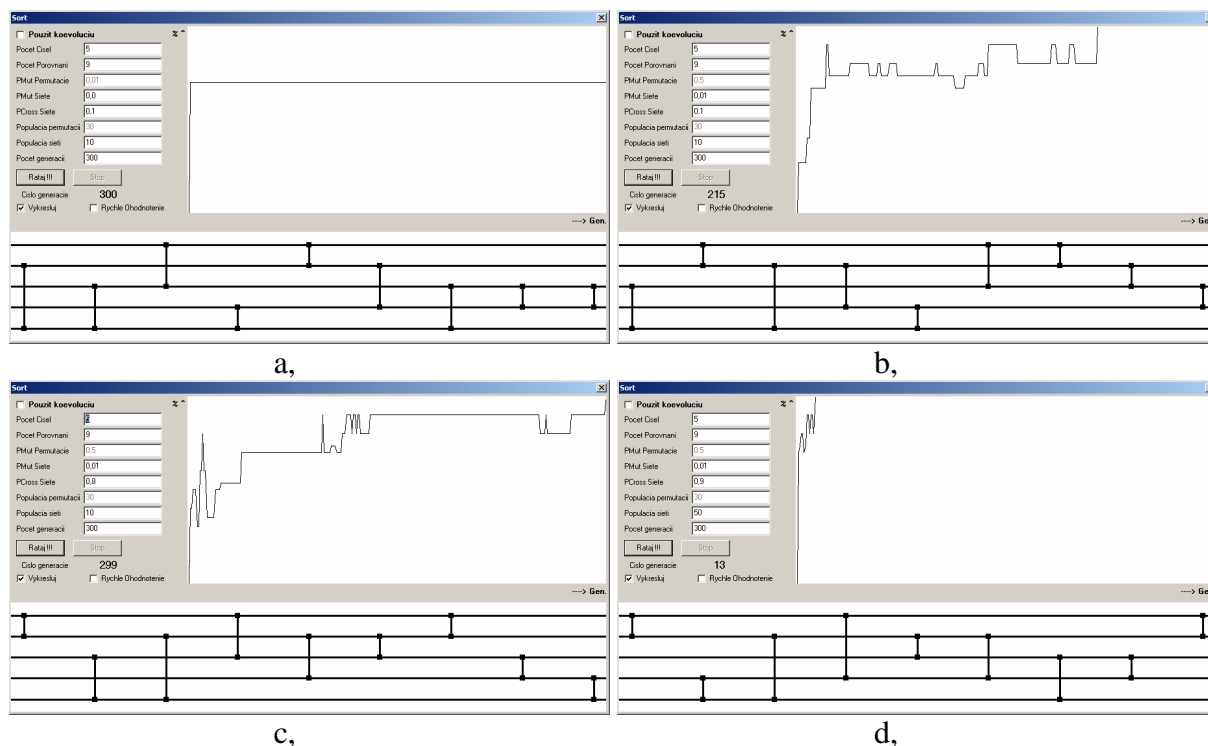
N=5

M=9

Počet spustení programu: 10

Populácia sietí	Kríženie siete	Mutácia siete	Ø Počet generácií	Obr.
10	0,1	0,0	$\infty$	Obr. 2,a
10	0,1	0,1	210	Obr. 2,b
10	0,8	0,1	280	Obr. 2,c
50	0,9	0,1	14	Obr. 2,d

Tab. 1 Namerané výsledky



Obr. 2 Obrazovky aplikácie

## Koevolučný genetický algoritmus

Evolúcia je adaptácia zameraná na naplnenie vlastných potrieb. Koevolúcia je vo väčšej miere adaptácia na naplnenie potrieb navzájom medzi druhmi. V prípade sortovacích sietí tu máme dva druhy: sortovacie siete a permutácie čísel. Na jednej strane máme populáciu sortovacích sietí, v ktorých hľadáme tú, ktorá správne zoradí všetkých  $N!$  permutácií. Na druhej strane je vyvíjaná nezávislá populácia permutácií v ktorých hľadáme tie permutácie, ktoré sú najhoršie zotriedené populáciou sortovacích sietí. Teda sú ohodnotené tým lepšie, čím viac sortovacích sietí na nich zlyháva.

Koevolučný algoritmus je veľmi výhodný a to hlavne z časového hľadiska. Pri veľkých hodnotách  $N$  by bolo testovanie sortovacích sietí veľmi zdĺhavé pretože by sa museli testovať pre každú sieť všetky permutácie  $N$  čísel, čo je  $N!$ . Už pri čísle  $N=10$  je to 3628800 permutácií. Z tohto faktu je jasné, že treba testovať siete len na malej populácii permutácií. Reprodukovať len siete, ktoré lepšie zatriedili populáciu permutácií. Reprodukovať permutácie, ktoré boli horšie zatriedené populáciou sietí. Tým budú siete vyvíjané stále na permutácie, ktoré sú horšie zatriedené a vyhneme sa tomu, že by sa siete „naučili“ triediť len konkrétne permutácie.

### ***Fitness sortovacích sietí***

Pri koevolúcii sa sortovacie siete testujú len na malý počet permutácií čísel. Ak by sme vypočítavali fitness podľa prechádzajúcej metodiky, tak sa môže stať, že fitness jedincov bude ohodnotená nulou. Pretože sortovacie siete nedokázali úplne zatriediť dané permutácie. Preto je vhodné hodnotiť zotriedenie permutácií podľa kvality zotriedenia. Kvalita je založená na rozdieloch pozícií čísel. Z týchto rozdielov sa sumáciou počíta výsledná hodnota. V Tab. 2 je ukázaný najhorší a najlepší prípad zotriedenia permutácie pre  $N=6$ .

Ideálne zotriedenie	Najhoršie zotriedenie	Ohodnotenie zotriedenia číslíc	Najlepšie zotriedenie	Ohodnotenie zotriedenia číslíc
1	6	5	1	0
2	5	3	2	0
3	4	1	3	0
4	3	1	4	0
5	2	3	5	0
6	1	5	6	0

**Tab. 2 Príklad ohodnotenia zotriedenia permutácií**

Ak označíme permutáciu číslíc „p“ a hodnotu číslice na i-tom mieste v permutácii ako „p(i)“, tak výsledný vzorec pre ohodnotenie zotriedenia permutácie je definovaný takto

$$\sum_{i=1}^N |i - p(i)| \quad (1)$$

Najlepšie ohodnotenie zotriedenia je 0. Aby bolo fitness lepších sortovacích sietí ohodnotené lepšie, tak ho musíme vypočítať takto

maximálne ohodnotenie – ohodnotenie zotriedenia permutácie

t.j.

$$\sum_{i=1}^N |i - (N - i + 1)| - \sum_{i=1}^N |i - p(i)| \quad (2)$$

### ***Fitness permutácií čísel***

Výber permutácií v reprodukcii závisí od ich kvality. Ich kvalita je tým väčšia, čím viac sortovacích sietí na nich zlyháva. Preto sa bude počítať fitness permutácií podľa vzťahu (1)

### ***Algoritmus***

1. Vygenerovanie náhodnej populácie sietí a populácie permutácií
2. Ak  $T = T_{\max}$ , tak vyhlás neúspech a skonči algoritmus
3. Triedenie každej permutácie s každou sieťou
4. Pre najlepšiu sieť sa zistí zatriedenie všetkých permutácií (kvôli grafu a ukončovaciemu kritériu)
5. Vytvorenie novej populácie sietí ruletovým výberom (kríženie + mutovanie)
6. Vytvorenie novej populácie permutácií ruletovým výberom (iba mutovanie)
7. Ak nebolo nájdené riešenie tak skoč na 2

### ***Programové riešenie***

Riešenie je založené na predchádzajúcej kapitole.

Nachádza sa tu nový objekt **TGACoevolution**, ktorý zapuzdruje koevolučný algoritmus. Obsahuje populáciu sietí a populáciu testovaných permutácií, ktoré sa triedia sortovacími sieťami. Objekt má vstupné parametre: počet číslíc N, počet porovnaní M, pravdepodobnosť kríženia a mutácie populácie, veľkosť populácie sietí  $P_s$ , pravdepodobnosť mutácie permutácie a veľkosť populácie permutácií  $P_p$ .

Na viac je tu možnosť „rýchleho ohodnocovania“. To znamená, že krok 4 v algoritme bude triediť permutácie dovtedy, pokiaľ ich najlepšia sortovacia sieť z populácie triedi správne. Ak sa nájde permutácia, ktorá nie je správne zotriedená, tak sa skončí krok 4. Tým urýchlíme výpočet, zatiaľ čo ukončovacie kritérium ostane nezmenené. Na to aby bolo splnené, musia byť všetky permutácie bezchybne zatriedené. Zapnutím tejto možnosti sa však dopúšťame veľkej chyby pri kreslení grafu. Táto možnosť je veľmi výhodná pri použití väčších N.

## Výsledky

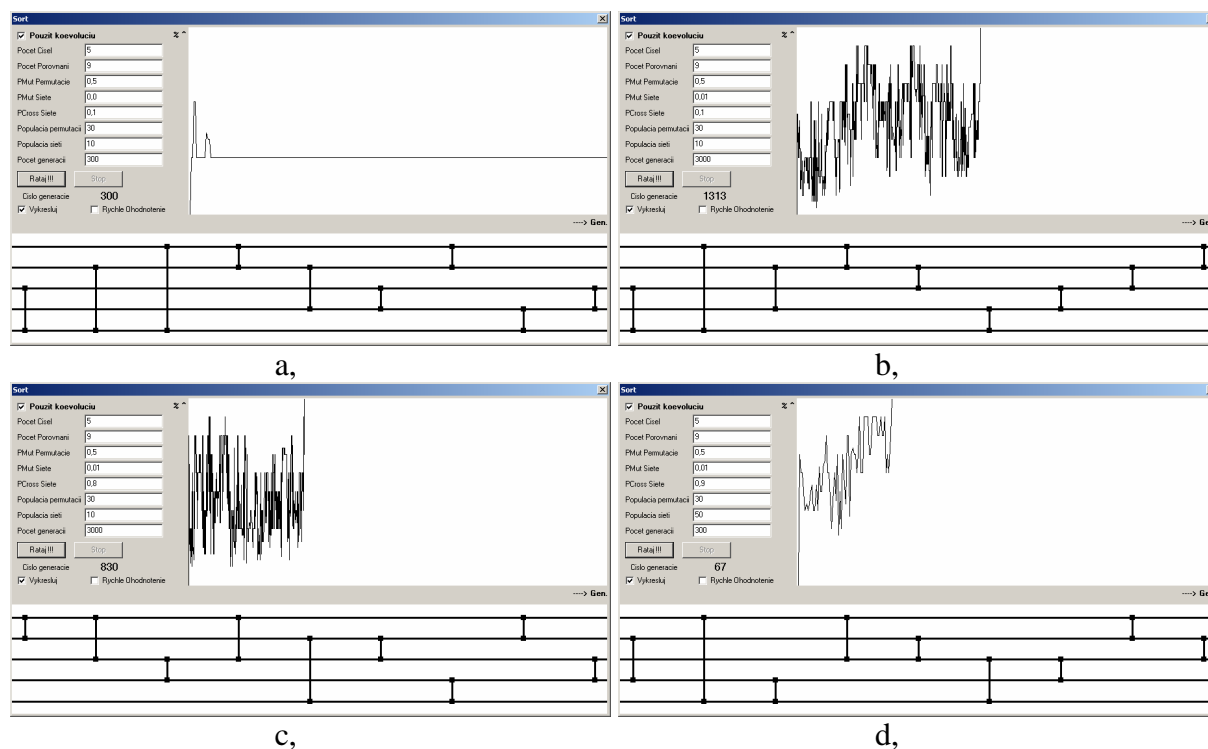
N=5

M=9

Počet spustení programu: 10

Populácia sietí	Populácia permutácií	Kríženie siete	Mutácia siete	Mutácia permutácie	Ø Počet generácií	Obr.
10	30	0,1	0,0	0,5	$\infty$	Obr. 3,a
10	30	0,1	0,1	0,5	1160	Obr. 3,b
10	30	0,8	0,1	0,5	800	Obr. 3,c
50	30	0,9	0,1	0,5	58	Obr. 3,d

Tab. 3 Namerané výsledky



Obr. 3 Obrazovky aplikácie pri použití koevolučného algoritmu



## **Záver**

Z uskutočnených pokusov, som zistil, že na evolúciou sortovacích sietí kladne vplýva veľkosť populácie jedincov. Na to aby sme získali vôbec nejaký rozumný výsledok, tak musí byť možné mutovať jedince. V opačnom prípade sa do populácie prestane pridávať „nová krv“ a najlepšie riešenie sa usídli v lokálnom minime.

Vyskúšal som aj použitie koevolučného algoritmu na rovnakú skupinu úloh. Pre nízke hodnoty  $N$  tomuto algoritmu trvalo dlhšie kým sa dopracoval k výsledku. Avšak už pri počte čísel  $N=7$ , počtu porovnaní  $M=20$  (pop. sietí a perm. = 50) dával koevolučný algoritmus badateľne lepšie výsledky. Klasickou evolúciou dosiahol algoritmus výsledok za priemerne 34 s. Pri použití koevolúcie to bolo už za 18s.

## **Zoznam použitej literatúry**

[1] – Prof. Ing. Vladimír Kvasnička, DrSc., Doc. RNDr. Jiří Pospíchal, CSc.: Evolučné algoritmy, 2000, 223 s., ISBN 80-227-1377-5