Evolučné algoritmy

Úloha č.12 (2000/2001):

Genetický algoritmus pre úlohu obchodného cestujúceho na ortogonálnej štvorcovej mriežke. Použite oba prístupy na kódovanie cesty, tak priamo pomocou permutácie, ako aj pomocou binárneho reťazca (pozri cvičenie 7.1 z kapitoly Kombinatoriálne optimalizačné metódy. Porovnajte efektívnosť týchto dvoch prístupov.

Vypracoval:
Fridrich Matz
FMFI UK, Bratislava, 5.ročník
matz.fridrich@nextra.sk
http://www.mojweb.sk/ea/

Riešenie:

Jeden z prvých veľkých úspechov evolučných algortimov bolo ich použitie na riešenia ťažkých kombinatorických problémov., ktorých čas CPU rastie buď faktoriálovo alebo exponenciálne s rastom dimenzie problému. Takýmto problémom tiež hovoríme NP ťažké problémy. Jedným zo študovaných problémov [1,str.92] je nasledovný:

Funkcia f je definovaná nad symetrickou grupou S_N zloženou zo všetkých permutácii N objektov

$$f: S_N \rightarrow R$$

kde permutácie z S_N sú definované ako N-tica rôznych čísel

$$P = (p_1,p_2, ...,p_N)$$

Optimalizačný problém má tvar

$$P_{opt} = arg\ min\ f(P),\ kde\ P \in S_N$$

Riešenie tohto optimalizačného problému je veľmi ťažké, pretože symetrická grupa S_N obsahuje N! permutácií a je potrebné preskúmať hodnoty pre všetky možné argumenty.

Tento typ kombinatorického problému budeme ilustrovať známou úlohou obchodného cestujúceho, tzv. Traveling Salesman Problem, TSP). Na kódovanie cesty použijeme 2 prístupy (tak ako je v zadaní úlohy)

a) priamo pomocou permutácie
b) pomocou binárneho reťazca

Popíšeme situáciu a) konkrétnejšie. Na začiatku simulácie vytvoríme požadovanú populáciu chromozómov, ktoré reprezentujú jedincov v populácii, podľa funkcií zo zadania. Z [1] vyplýva, že klasická permutácia bude reprezentovať postupnosť, v ktorej mestá navštívime. Graf si zadefinujeme ako ortogonálnu štvorcovú mriežku, kde mestá sú spojené len vodorovne a zvisle, ich hodnota (cena) je jedna. Nebudeme uvažovať „šikmé“ cestičky (cesty, keď musíme prejsť cez iné mestá/mesto, aby sme sa dostali do cieľa). Majme teda populáciu chromozómov, kde každý reprezentuje cestu. Keďže sa jedná o genetický algoritmus, treba ešte dodefinovať operáciu mutácie a operáciu kríženia medzi jednotlivcami.

Operáciu mutácie zadefinujeme klasicky. Pre každé miesto v chromozóme sa hodí kocka (O_{mut}) a na základe výsledku hodu sa, či nastane mutácia. Ak áno, náhodne sa vyberie miesto, s ktorým sa chromozóm vymení. Môže sa stať situácia, že sa vyberie ten istý chromozóm, a teda zmutovaný bude rovnaký ako predtým, toto nebudeme považovať za mutáciu. V nasledujúcom pseudo kóde môžeme vidieť časť procedúry vykonávajúcu mutácie:

```
for i:=0 to MAX_CHROM-1 do
  if Random < pmut then
    begin
      rnd := random(MAX_CHROM);
      temp := chrom[i];
      chrom[i] := chrom[rnd];
      chrom[rnd] := temp;
    end;
```

Operáciu kríženia zadefinujeme opäť klasicky ako v [1]. Najprv zvolíme bod kríženia c, tak že 1 < c < n, kde skrížime chromozómy (permutácie) *P*,*Q*:

$$(P', Q') = O_{cross}(P, Q)$$
$$P = (p_1,...p_c, p_{c+1},...p_n)$$
$$Q = (q_1,...q_c, q_{c+1},...q_n)$$

Po výmene dostávame dva nové objekty:

$$P' = (p_1,...p_c, q_{c+},...q_n)$$
$$Q' = (q_1,..q_c, p_{c+},...p_n)$$

Samozrejme, že po skrížení dvoch permutácii nastáva vo väčšine prípadov rozpor z definíciou permutácie a teda *P'* a *Q'* viac nereprezentujú chromozóm. Nasleduje teda opravný proces algoritmu, ktorý hľadá výskyt P_i $i < c$ v Q_j, kde $j >= c$ a naopak. Každú takú dvojicu (P_a,Q_b), čo nájde, vymení. Je zrejmé, že po tejto oprave už ide o permutáciu.

Už sme si povedali ako vytvárame mutáciu a kríženie. Ako ale vytvárame populáciu. Populáciu vytvoríme tak, že vygenerujeme náhodné permutácie danej dĺžky. Tie teda môžeme krížiť (dve permutácie), či ich mutovať, treba ešte zadefinovať silu podľa ich významu, teda fitness. Fitness daného chromozómu, určujúca jeho silu/ význam, definujeme ako súčet vertikálnych a horizontálnych vzdialeností (manhattanovskú vzdialenosť) medzi dvojicami po sebe idúcich miest a tiež vzdialenosť posledného mesta do prvého. Štartovacie mesto tu nie je vhodné použiť, lebo ide vždy o potenciálne iné mesto. Hodnoty fitness pred „prejdením" do ďalšej generácie znormalizujeme:

$$F'=1/(F-minF+1)$$

Potom vytvoríme pole *Fitnesses*, ktoré bude obsahovať jednotlivé fitness chromozómov. Toto pole upravíme tak, že ku každému členu pripočítame všetky predošlé hodnoty fitness, takže bude vytvárať neklesajúcu, dokonca možno povedať rastúcu postupnosť. Keďže žiadna z fitness chromozómov nie je nulová po normalizácii.

V každom kroku pomocou rulety vyberieme dva chromozómy, na ne aplikujeme kríženie a mutáciu. Znovu vyrátame fitness, vysvietime najlepší permutáciu (chromozóm) a taktiež si označíme doteraz najlepšiu permutáciu (chromozóm) zo všetkých generácii.

Teraz si ukážeme, akým spôsobom realizujeme ruletu. Práve na to slúži úprava poľa *fitnesses*. Zvolíme si náhodne nejakú hodnotu z intervalu [0, *max*], kde *max* je posledná hodnota z poľa *fitnesses*. Potom v poli *fitnesses* hľadáme jej najbližšiu vyššiu hodnotu, a práve tento chromozóm je vybratý. Rovnakým spôsobom vyberieme druhý chromozóm a na nich môžeme aplikovať kríženia a mutáciu. Toto vyhľadanie v poli *fitnesses* môžeme zapísať takto:

BinarySearch(fitnesses, Random(fitnesses[pop_size-1]));

Ako je to v prípade b)? Permutácia *N* objektov sa môže zadať ako binárny vektor dĺžky *kN*, kde *k* je také celé číslo, ktoré zabezpečuje, aby binárny vektor dĺžky *k* bol schopný reprezentovať číslo *N*. Takýto vektor sa ľahko pretransformuje postupom *TRANSFORM*, ktorý bližšie popíšeme neskôr, na vektor *N* celých čísel z intervalu $[0, 2^k-1]$. Potom je permutácia *P* definovaná tak, že celočíselné zložky sú usporiadané do nerastúcej alebo neklesajúcej postupnosti. Napríklad, nech $N = 5$, $k = 3$, binárny vektor (101|011|001|111|101), potom jeho celočíslená verzia urobená *TRANSFORM*-om má tvar (5,4,1,7,5). Teraz musíme ešte túto *N*-ticu presusporiadať do postupnosti. Nech je permutácia *P* určená tak, že preusporiada túto *N*-ticu do rastúcej postupnosti, potom *P* = (3,2,1,4,5). Tu je veľmi dôležité ako funguje procedúra (algoritmus), ktorá vykonáva toto preusporiadanie do klesajúcej alebo rastúcej postupnosti. Hodnota Fitness je počítaná tak, že najprv dekódujeme binárny vektor dĺžky *kN* a až potom počítame jej hodnotu, spôsobom ako v prípade a). Výber chromozómu je aplikovaný pomocou rulety, kríženie a mutácie sú robené iným spôsobom ako prípad a). Kríženie sa robí tak, že pomocou rulety sa vyberú dva chromozómy (vektory), určí sa náhodný bod kríženia *c*, z intervalu [0, *kN*]. Samostatné kríženie je už potom robené veľmi jednoduchou zámenou. Nové vektory sú tvorené takto:
Prvý - vezme sa jeho časť pred bodom kríženia a zreťazí sa s časťou druhého za bodom kríženia
Druhý - vezme sa jeho časť pred bodom kríženia a zreťazí sa s časťou druhého za bodom kríženia.
Mutácia je robená výmenou bitu v danom vybranom vektore s pravdepodobnosťou P_{mut}. Ešte si ukážeme akým spôsobom pracuje *TRANSFORM*, ktorý prekódováva binárne číslo do celého čísla. Táto transformácia je robená nasledovným spôsobom:

Nech sa binárny vektor α dĺžky *k*, kde

$$\alpha = (\alpha_1 \alpha_2 ... \alpha_k) \in \{0,1\}^k$$

interpretuje ako celé číslo

$$\text{int}(\alpha) = \sum_{i=1}^{k} \alpha_i 2^{k-i} .$$

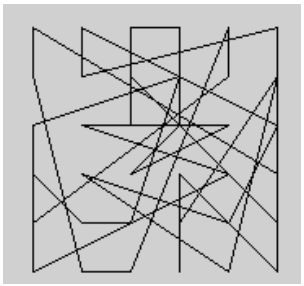
K tomuto celému číslu jednoduchým spôsobom priradíme reálne číslo, ktoré sa môže chápať ako aproximácia reálneho čísla x ∈ [a,b]:

$$x \approx \text{real}(\alpha) = a + \frac{b-a}{2^k - 1}\text{int}(\alpha)$$

Týmto spôsobom je robená transformácia binárnej reprezentácie na reálnu reprezentáciu.

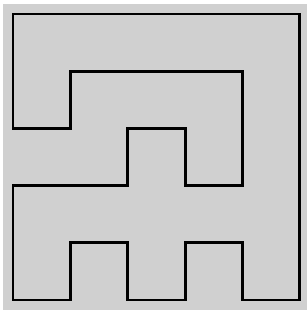
Nevýhodou tohto prístupu ku kódovaniu permutácií je, že rovnaká permutácia je určená rôznymi binárnymi reťazcami, čo môže spôsobovať pomerne malú efektívnosť evolučných algoritmov aplikovaných na riešenie kombinatorických algoritmov, čo uvidíme v nasledujúcej časti práce.

Dostávame sa k porovnaniu obidvoch postupov na kódovanie cesty. Ukážeme si ako môže vyzerať mriežka aj v prípade a) aj b) začiatku, kde veľkosť mriežky je 6x6.



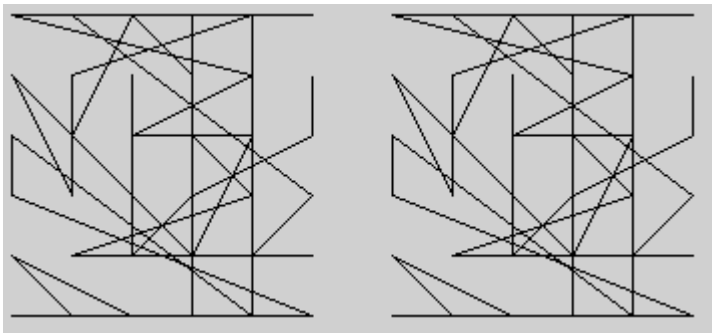
Takto môže vyzerať na začiatku mriežka [6x6]

Optimálne riešenie pre veľkosť mriežky 6x6 je:

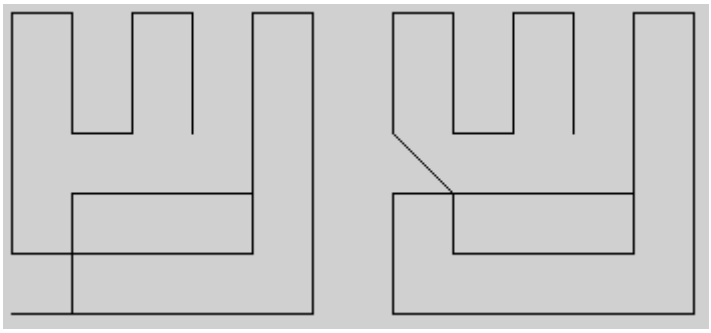


Optimálne riešenie pre 6x6

Ukazuje sa určitá závislosť veľkosti chromozómu a koeficientu P_{mut}. Čím je väčšia veľkosť chromozómu, ktorá závisí priamo od veľkosti šírky a výšky mriežky [môžeme ju počas behu programu meniť podľa potrieb], tím nižší koeficient pre P_{mut} je vhodnejší. Ale je dosť problematické vybrať vhodný, nakoľko v neskorších generáciách to bude hlavne tento parameter, čo zabráni opusteniu doteraz najlepšieho riešenia, či naopak, populácia sa odoberie iným smerom. Ukazuje sa, že Murphyho zákon funguje a vždy sa väčší koeficient správa tak, ako by sme si to najmenej želali. Keď ho zvýšime, aby nezastalo riešenie príliš skoro: napr. pri mriežke 8x8 je koeficient 0,07 priveľký, použijeme teda 0,06. Teraz zase výpočet nedosahuje sub-optimálne riešenie v skorých generáciach, navyše ale v neskorších zase začne oscilovať. Vráťme sa späť k porovnávaniu. Použijeme na to mriežku 6x6 a 10x10. Najprv si však ukážeme nejaké náhodne vybraté správanie algoritmu a) na mriežke 6x6 samostatne, kedy ešte nebudeme porovnávať.



Začiatok je „chaotický“, po malom počte generácií sa to už zmení k lepšiemu



Vidíme ti najlepšie možnosti (až do generácie 4458). Vľavo je ako to vyzerá v tejto generácii, vpravo ako to bolo doteraz. Oba obrázky majú rovnakú fitness, avšak pravý sa objavil skôr ako ľavý. Vždy sa zobrazuje len jeden z najlepších.

Na ukážku ako je efektívna táto metóda si zobrazíme tabuľku, v ktorej budeme počítať priemernú chybu od optimálneho riešenia. Z tejto tabuľky výsledkov (skôr by sa dalo povedať, z tohoto prehľadu), zo získaných údajov sa nedá urobiť záver o tejto metóde, možno vyrátať priemernú hodnotu, ktorú vie táto metóda získať z mriežky 6x6:

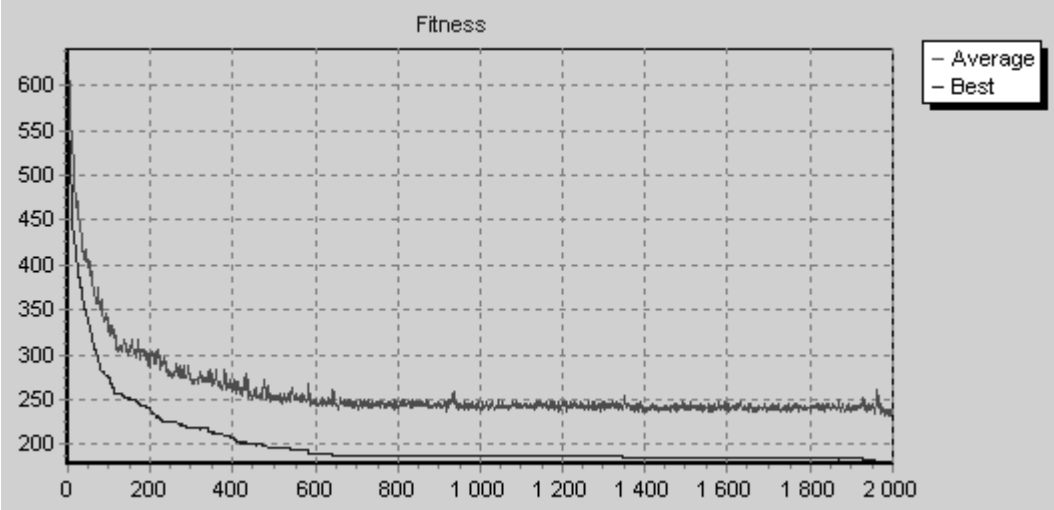
Clasik	Populácia	P_{cross}	P_{mut}	Generácia	Vzdialenosť	Počet iterácii	Chyba
1	100	0.7	0.05	2632	38	10000	2.63%
2	100	0.7	0.05	1600	44	10000	15.91%
3	100	0.7	0.05	3834	40	10000	7.50%
4	100	0.7	0.05	9467	42	10000	11.90%
5	1000	0.7	0.05	1917	40	2000	7.50%
6	1000	0.7	0.05	830	40	2000	7.50%
7	1000	0.7	0.05	713	42	2000	11.90%
8	500	0.5	0.07	1788	38	2000	2.63%
9	500	0.5	0.09	831	40	2000	7.50%
10	500	0.5	0.075	1615	42	2000	11.90%
11	500	0.5	0.075	1079	40	2000	7.50%
priemer							**8.58%**

Priemerná hodnota je 8.58%, čo je dosť vysoké percento, avšak iterácii nebolo veľa. Je však vidno, že dokážeme dostať aj sub-optimálne riešenie a k 9. riadku treba už len dodať, že si riešenie nenašlo stále miesto v populácii. Z týchto (a ďalších) pokusov by sa dalo usúdiť, že v prípade, že nájdeme vhodne parameter P_{cross} a P_{mut} pre menšie mriežky, je skôr vhodne zväčšit o niečo populáciu ako výrazne zmenšiť P_{mut}. Takisto priveľká populácia neprinesie výrazné zlepšenia. Kým prejdeme k výsledkom zo zadania b), aby sme mohli urobiť záver, treba ešte povedať, že táto metóda dáva riešenia, ako prejsť za čo najkratší čas ***všetky*** mestá.

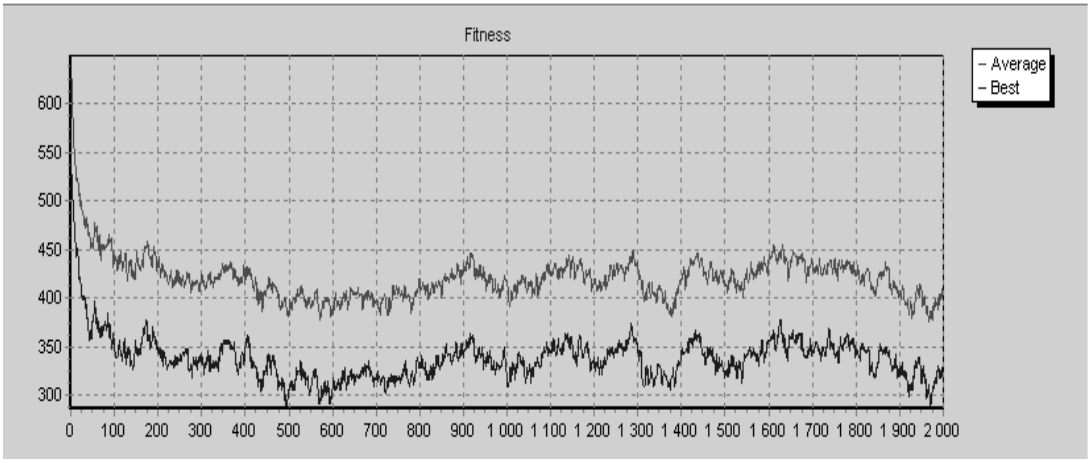
Binary	Populácia	P_{cross}	P_{mut}	Generácia	Vzdialenosť	Počet iterácii	Chyba
1	100	0.7	0.04	619	64	10000	42.19%
2	100	0.7	0.03	5418	56	10000	33.93%
3	100	0.5	0.03	5327	62	10000	40.32%
4	100	0.5	0.01	7669	46	10000	19.57%
5	100	0.3	0.01	5070	42	10000	11.90%
6	500	0.3	0.01	5687	42	10000	11.90%
7	500	0.3	0.005	2116	40	5000	7.50%
8	500	0.1	0.001	2353	40	5000	7.50%
9	500	0.7	0.01	2103	38	5000	2.63%
10	500	0.7	0.005	468	48	5000	22.92%
11	500	0.5	0.01	574	42	5000	11.90%
priemer							**19.30%**

Táto tabuľka je pre prípad b). Počítali sme opäť priemerná chybu od optimálneho riešenia. Vidíme, že je ešte väčšia ako v prípade a). Ani „pohraním sa“ s parametrami, či opakovaným spustením sa nám nepodarí dostať pre mriežku 6x6 sub-optimálne riešenie pre metódu b), ktorá dáva 2-2.5 - krát horšie výsledky ako pre prípad a). Samozrejme nie je to jediné hľadisko, z ktorého sa môžeme na algoritmus dívať. Ďalšou zvláštnosťou algoritmu a) je jeho vytrvalosť, po veľa iteráciách dostaneme naozaj nízke hodnoty, avšak stále treba manipulovať s parametrami, čo len zvyšuje výpočtovú náročnosť a na druhej strane, v algoritme b) aj tak v priemere nedosiahneme nízke hodnoty.

Ešte si ukážeme aj priebehy jednotlivých algoritmov pre lepšie pochopenie fungovania algoritmov. Tentoraz pre mriežku veľkosti 10x10, zvyšné parametre popíšeme priamo pod každým obrázkom samostatne.

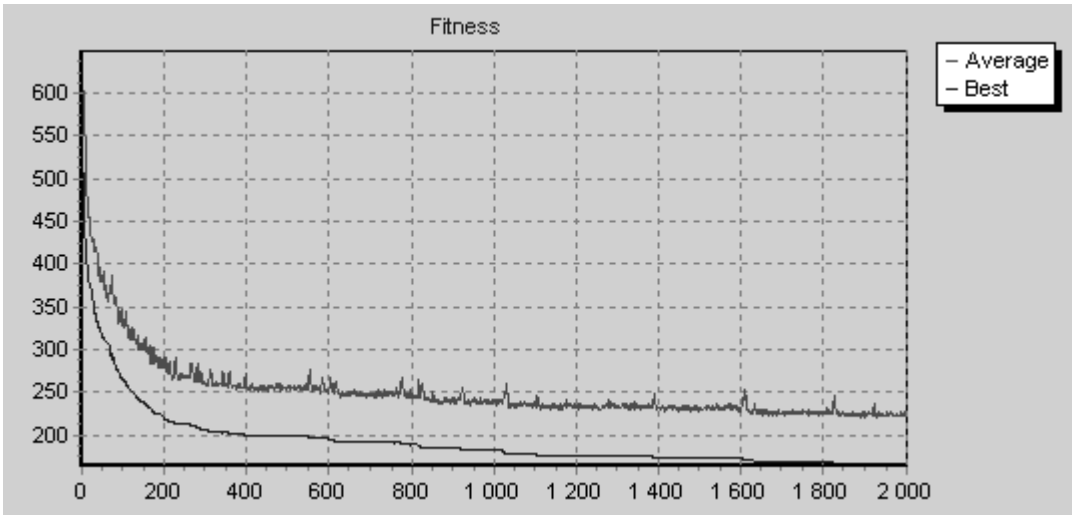


Metóda a) je pomalá, ale vytrvalá [pop = 500; P_{cross} = 0,65; P_{mut} = 0,025]

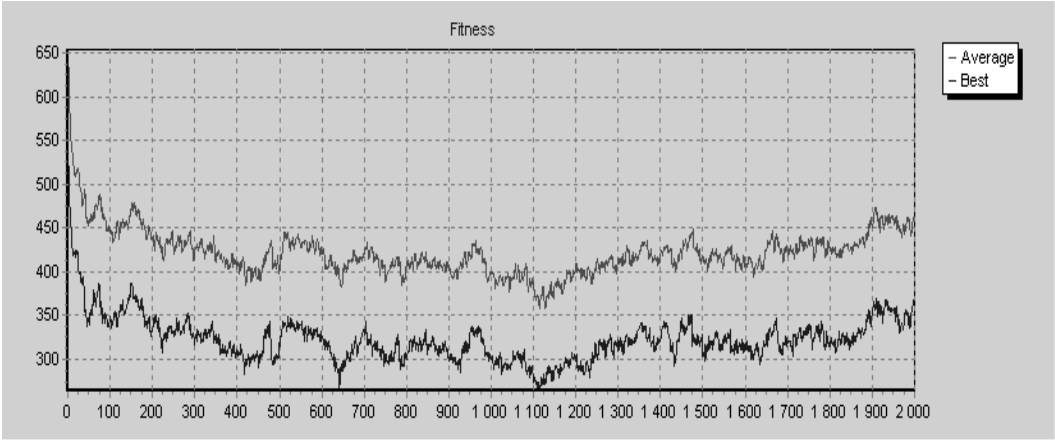


Metóda b) taktiež pomalá a neefektívna [pop = 500; P_{Cross} = 0,6; P_{mut} = 0,01]

Skúsme teda ešte pár testov pre populáciu veľkosti 1000 a pre 2000 generácii.



Metóda a) pre [P_{cross} = 0,65; P_{mut} = 0,025] ; bestFiness = 164 v 1918. generácii



Metóda b) pre [P_{cross} = 0,7; P_{mut} = 0,01] ; bestFitness = 264 v 1116. generácii

Aj z týchto priebehov, kde sme opäť menili veľkosť populácie, pravdepodobnosť mutácie a pravdepodobnosť kríženia môžeme sledovať, že metóda b), kódovanie cesty pomocou binárneho vektora je veľmi pomalá a menej efektívna ako metóda a), teda kódovanie cesty priamo pomocou permutácie.

Vidíme, že pri malom čase CPU vieme povedať s 10% chybou riešenie v prípade a) alebo 20% v prípade b). Ak však máme k dispozícii viac času CPU, vieme takto ohraničiť aj zhora metódou a). Napriek neustálym meneniam ovplyvňujúcich parametrov sa nám nepodarilo ani mriežku 6x6, ani pre mriežku 10x10 nájsť optimálne riešenie v oboch prípadoch. Na zlepšenie výsledkov by sme mohli tieto prístupy zmodifikovať. Jedna možnosť je si vytipovať isté chromozómy ako jedinečné. Alebo iná možnosť. Používame napríklad len jednu ohodnocovaciu funkciu, avšak človeka (živého tvora) často krát inšpiruje viac faktorov. Niektoré síce len s malou pravdepodobnosťou. Nakoniec, aj "čarodejníctvo" kedysi so "slabou fitness" je dnes vhodná alternatíva k medicíne, ktorá má vysokú fitness vo všeobecnosti. Pridaním niekoľko (napr. 3% z populácie) náhodne vybratých chromozómov spomedzi najmenej ohodnotených by sme mohli mať alternatívnu zásobu chromozómových reťazcov. Tieto chromozómy by boli akoby pod ochranou spoločnosti.

Literatúra:

[1] V. Kvasnička, J.Pospíchal, P.Tiňo: ***Evolučné algoritmy,*** **Vydavateľstvo STU, Bratislava 2000**