VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

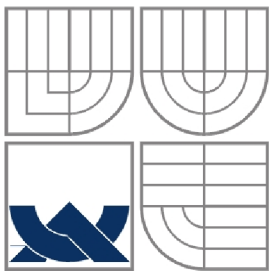
TECHNIKA ALPS V KARTÉZSKÉM GENETICKÉM PROGRAMOVÁNÍ

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

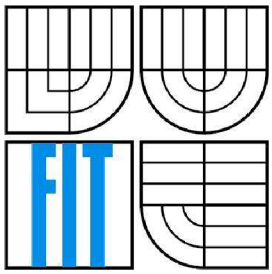
AUTOR PRÁCE
AUTHOR

Bc. Peter Stanovský

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

TECHNIKA ALPS V KARTÉZSKÉM GENETICKÉM PROGRAMOVÁNÍ

ALPS TECHNIQUE IN CARTESIAN GENETIC PROGRAMMING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. Peter Stanovský

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. Ing. Lukáš Sekanina, Ph.D.

BRNO 2009

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2008/2009

Zadání diplomové práce

Řešitel: **Stanovský Peter, Bc.**
Obor: Počítačové systémy a sítě
Téma: **Technika ALPS v kartézském genetickém programování**
Kategorie: Umělá inteligence

Pokyny:

1. Zpracujte studii na téma kartézské genetické programování (CGP) a Age Layered Population Structure (ALPS).
2. Navrhněte způsob začlenění ALPS do CGP.
3. Implementujte CGP využívající ALPS.
4. Ověřte standardní CGP a CGP využívající ALPS na zadaných testovacích úlohách.
5. Zhodnoťte dosažené výsledky.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části diplomového projektu je požadováno:

- Splnění prvních 2 bodů zadání.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Sekanina Lukáš, doc. Ing., Ph.D.**, UPSY FIT VUT
Datum zadání: 22. září 2008
Datum odevzdání: 26. května 2009

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2

doc. Ing. Zdeněk Kotásek, CSc.
vedoucí ústavu

Abstrakt

Úvodem práce přináší stručný přehled problematiky softcomputingu a řešení NP-úplných problémů. Zejména se věnuje evolučním algoritmům a jejich základním typům. V další části je zpracována studie na kartézské genetické programování (CGP), které patří do oblasti evolučních algoritmů. CGP se používá zejména pro návrh číslicových obvodů, symbolickou regresi a jiné. Samostatná kapitola je věnována studii nové techniky Age layered population structure (ALPS), která se věnuje problému předčasné konvergence, přičemž navrhuje způsob rozdělení populace na subpopulace separovaných na základě věkového kriteria. Pomocí zajišťování dostatečné diverzity v populaci dosahuje významně lepších řešení oproti klasickým evolučním algoritmům. V práci jsou navrženy dva způsoby začlenění techniky ALPS do CGP. V další části je popsán postup implementace klasického CGP a jeho dvou variant s využitím techniky ALPS. V rámci práce byly vykonány testy na klasických testovacích úlohách s použitím a bez použití techniky ALPS, přičemž v části „Experimentálne výsledky“ byl diskutován přínos použití techniky ALPS v CGP oproti klasickému CGP.

Abstract

This work introduces a brief summary of softcomputing and the solutions to NP-hard problems. It especially deals with evolution algorithms and their basic types. The next part involves the study of cartesian genetic programming, which belongs to the field of evolution algorithms, used mainly in the evolution of digital circuits, symbolic regression, etc. A special chapter is devoted to the studies of new technique Age layered population structure, which deals with the problems of premature convergence, which suggests the way of how the population could be divided into subpopulations split up according to the age criteria. Thanks to the maintaining of sufficient diversity, it achieves substantially better solutions in comparison to the classical evolution algorithms. This papier includes the suggestion of two ways of incorporation of the ALPS technique into CGP. In the next part of work there were carried out tests on the classic problems, that would be solved with evolution algorithms. These tests were made with and without using ALPS technique. In the part of work „Experimental results“ there was discussed a contribution of using ALPS technique in CGP against the classic CGP.

Klíčová slova

Evoluční algoritmus, kartézské genetické programování, age layered population structure, kombinační obvod, problém předčasné konvergence.

Keywords

Evolution algorithm, cartesian genetic programming, age layered population structure, digital circuit, problem of premature konvergence.

Citace

Stanovský Peter: Technika ALPS v kartézském genetickém programování, diplomová práce, Brno, FIT VUT v Brně, 2009

Technika ALPS v kartézském genetickém programování

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením doc. Ing. Lukáša Sekaniny, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Peter Stanovský
26. 5. 2009

Poděkování

Děkuji vedoucímu diplomové práce Doc. Ing. Lukášovi Sekaninovi, Ph.D. za poskytnutou podporu, cenné informace a trpělivost při mých otázkách.

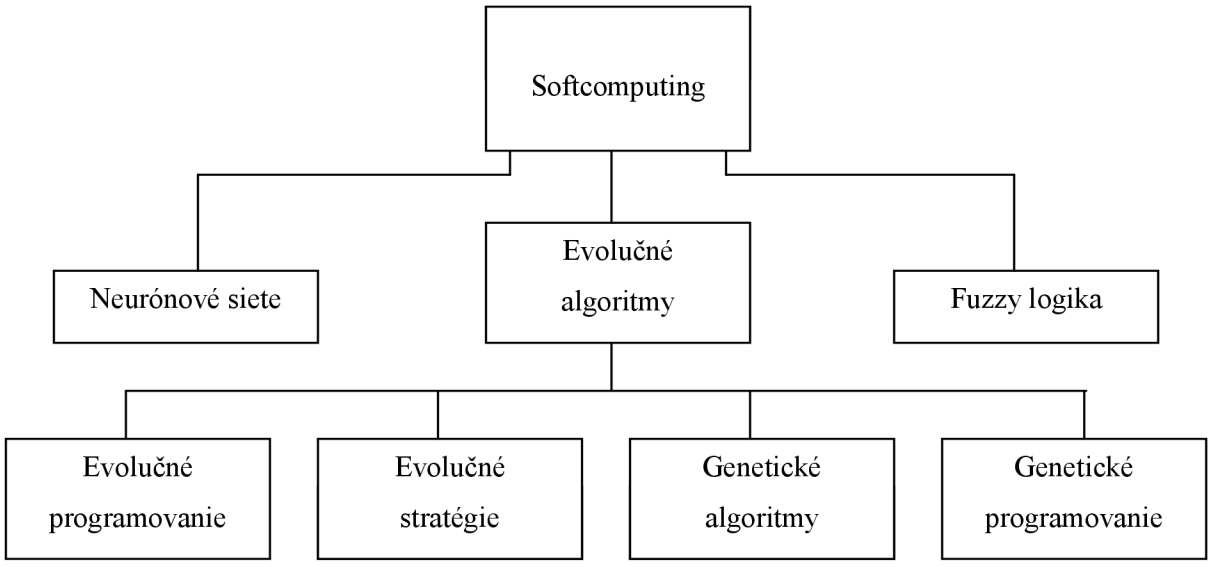
Obsah

1 Úvod

Pri hľadaní odpovede na otázku, ktoré riešenie problému je najlepšie, optimálne, je možné použiť obrovské množstvo postupov, ktoré sú viac či menej úspešné. Avšak pri hľadaní optimálneho riešenia v tzv. NP-úplných problémoch, ako je napríklad riešenie niektorých problémov z teórie grafov, plánovanie úloh na výrobnej linke, hľadania predpisu komplikovaných matematických funkcií či iných, narážame pri klasickom „hard computingu“ na problém s exponenciálne rastúcou pamäťovou či časovou zložitosťou riešenia.

Boli preto postupom času objavené a zavádzané metódy tzv. softcomputingu, ktorý zohľadňuje a akceptuje nepresnosť reálneho sveta. Sú vlastne „inteligentné programy“, teda také, ktoré riešia problémy bez ich exaktného algoritmického popisu.

Do oboru softcomputingu patrí niekoľko oblastí, pričom začlenenie evolučných algoritmov a ich základných techník je vidieť na obrázku č. 1.



Obrázok č. 1: Prehľad evolučných algoritmov [2]

Cieľom tejto práce je vytvoriť štúdiu problematiky kartézskeho genetického programovania (CGP) a novej metódy zabraňujúcej predčasnej konvergencii Age Layered Population Structure (ALPS), na základe ktorej bude vytvorený návrh a implementácia kartézskeho genetického programovania využívajúceho techniku ALPS. Táto bude overená na zvolených testovacích úlohách a porovnaná s výsledkami poskytovanými štandardným CGP.

Druhá kapitola sa zaoberá evolučnými algoritmami a popisuje ich základné rozdelenie, pričom v jednotlivých podkapitolách sú tieto základné typy evolučných algoritmov podrobnejšie vysvetlené.

V tretej kapitole je podrobne popísaná metóda kartézskeho genetického programovania, a jeho využitie pri návrhu kombinačných logických obvodov, symbolickej regresii a programov.

Štvrtá kapitola obsahuje štúdiu techniky Age layered population structure.

V poslednej piatej kapitole je navrhnutý spôsob začlenenia techniky ALPS do kartézskeho genetického programovania, ktorý bude použitý pri implementácii v ďalšej časti diplomového projektu.

2 Evolučné algoritmy

Na základe literatúry (najmä [2] a [3]) je v tejto kapitole popísaná problematika evolučných algoritmov, vysvetlené základné pojmy ako princíp evolučného procesu, selekcia, kríženie, mutácia, fitness jedinca a iné.

Vychádzajúc z Darwinovej evolučnej teórie, jej abstrahovaním a formalizovaním sme dostali univerzálnu optimalizačnú metódu, ktorá slúži ako prostriedok modernej numerickej matematiky. Evolučné algoritmy sú použiteľné pri hľadaní globálnych miním (maxím) funkcií, ktoré sú obklopené veľkým množstvom lokálnych miním (maxím). Sú založené na napodobení prirodzenej biologickej evolúcie v prírode(napr. živočíšnych druhov).

V klasickej Darwinovej teórii sú populácie geneticky variabilné, pričom medzi jedincami prichádza k prirodzenému výberu, kedy prežívajú jedince najviac adaptované na dané prostredie. Týmto spôsobom sa ďalej šíri ich genetický materiál na ich potomkov. V priebehu procesu evolúcie tak dochádza k uprednostňovaní najviac adaptovaných jedincov, až pokým nie je proces evolúcie takmer zastavený (v prírode je tento jav možno vidieť napríklad u niektorých živočíšnych druhov, kedy sa ich genetický materiál za posledných niekoľko miliónov rokov menil iba minimálne).

Biologická evolúcia je teda progresívna zmena genetického obsahu populácie v priebehu mnoho generácií. Obsahuje tieto tri hlavné komponenty [2]:

1. Prirodzený výber, kedy do procesu reprodukcie vstupujú jedince najviac adaptované na prostredie (majú vysokú hodnotu fitness).
2. Náhodný genetický drift - náhodné udalosti počas života jedincov ovplyvňujú populáciu (toto je významné najmä pre malé populácie). Patrí sem napríklad mutácia genetického materiálu alebo smrť jedinca (jedinec s vysokou hodnotou fitness sa tým pádom nemôže zúčastniť procesu reprodukcie a jeho genetický materiál sa tak nedostane do ďalšej generácie).
3. Proces reprodukcie, počas ktorého prichádza k výmene genetického materiálu rodičov (najčastejšie dvoch), a vznikajú potomkovia – najčastejšie tento proces prebieha tak, že z genetickej informácie dvoch jedincov sú vybrané časti, ktoré slúžia k zostaveniu genetickej informácie nového jedinca (tzv. „sexuálna reprodukcia“).

Schopnosť jedinca prežiť v danom prostredí je charakterizovaná jeho hodnotou fitness – relatívne definuje schopnosť daného jedinca prežiť v prostredí a reprodukovať sa konkrétnej populácii.

Evolučné algoritmy teda pracujú na princípe evolúcie v reálnom svete. Riešenie úlohy je prevedené na proces evolúcie náhodne (alebo za pomoci nejakej heuristiky) generovaných počiatočných riešení. Jednotlivé riešenia sú zakódované do reťazca symbolov (binárny reťazec,

prípadne strom, reťazec reálnych čísiel...). Nasleduje ich ohodnotenie pomocou funkcie fitness – čím vyššia hodnota fitness, tým väčšmi je jedinec adaptovaný na prostredie a je takisto vyššia jeho šanca vstúpiť do procesu reprodukcie. Populácia riešení je nazývaná aj populácia indivíduí alebo chromozómov, pričom chromozóm sa skladá z jednotlivých génov.

Reprodukciu posúvajú vpred najmä tieto dve hlavné sily:

1. Selekcia, ktorá umožňuje presádzať sa najmä jedincom s vysokou hodnotou fitness (teda najviac adaptovaných na dané prostredie), čím je umožnený prenos genetického materiálu.
2. Operátory kríženia a mutácie, ktoré zabezpečujú dostatočnú rozmanitosť populácie.

Kombináciou týchto dvoch vyššie uvedených metód je možné získať jedince v nasledujúcej generácii s vyššou hodnotou fitness. Ako bolo spomenuté vyššie, pri sexuálnej reprodukcii použitím operátoru kríženia prichádza ku kríženiu (najčastejšie dvoch) jedincov, pričom vznikajú (takisto najčastejšie dvaja) potomkovia. Metóda selekcie uprednostňuje jedince s vyššou hodnotou fitness, avšak aj jedince s nižšou hodnotou by mali dostať príležitosť k reprodukcii – prispieva to k vyššej diverzite populácie a zmenšuje riziko predčasnej konvergencie populácie – teda že dostaneme iba suboptimálne riešenie niekde v lokálnom minime, v dôsledku malej diverzity sa teda nedostávame ku globálnemu maximu. Operátor mutácie simuluje proces mutácie v prírode, kedy časť genetickej informácie jedinca je zmenená čisto náhodne.

Nižšie je uvedený všeobecný evolučný algoritmus [2]:

```
begin
  t := 0;                          //nastavenie počiatočného času
  initpopulation P (t);            //inicializácia populácie – náhodné generovanie jedincov
  evaluate P  (t);                 //ohodnotenie počiatočnej populácie
while not finished do              //pokým neplatí ukončovacia podmienka
  t := t + 1;                      //zvýš číslo populácie
  P' := selectpar P  (t);          //selekcia rodičov
  recombine P'  (t);               //rekombinácia rodičov - kríženie
  mutate P'  (t);                  //mutácia náhodných jedincov
  evaluate P'  (t);                //ohodnotenie nových potomkov
  P := survive P,P'  (t);          //obnova populácie pre ďalšiu generáciu
do
end
```

2.1 Základné pojmy z EA

Gén je základná stavebná jednotka chromozómov, ktorá môže nadobúdať iba určitý počet hodnôt nad definovanou abecedou (napr. ‘0‘ alebo ‚1‘ v prípade kódovania do binárneho reťazca, ‘+’ napríklad u genetického programovania v prípade stromovej reprezentácie).

Chromozóm je jedinec zložený z istého počtu génov, reprezentuje jedno konkrétne riešenie v prehľadávanom stavovom priestore. Je to vektor obvykle pevnej dĺžky (toto neplatí u genetického programovania).

Populácia je tvorená konečným počtom jedincov (chromozómov). Počiatočná populácia je generovaná náhodne.

Genotyp predstavuje kódovaný tvar riešenia, napr. binárny reťazec.

Fenotyp predstavuje dekódovaný tvar riešenia, teda konkrétna hodnota nejakého jedinca, napríklad konkrétny obvod, cesta obchodného cestujúceho.

Fitness funkcia relatívne ohodnocuje adaptovanosť konkrétneho riešenia v danom priestore. Hľadáme pomocou nej najlepší chromozóm – jedinca, ktorý najviac vyhovuje našim požiadavkám.

2.2 Komponenty EA

Evolučné algoritmy pozostávajú z nasledujúcich komponent:

1. Reprezentácia riešenia – spôsob zakódovania.
2. Funkcia fitness – spôsob ohodnotenia jedincov (ich adaptovanosti na prostredie).
3. Populácia.
4. Technika výberu rodičovských jedincov.
5. Operátory kríženia a mutácie.
6. Obnova populácie.

Všetky body budú podrobnejšie rozobrané v jednotlivých podkapitolách, pretože sú odlišné u konkrétnych typov evolučných algoritmov.

2.3 Aplikácia evolučných algoritmov, ich nasadenie a nevýhody

Evolučné algoritmy nachádzajú uplatnenie všade tam, kde zlyhávajú tradičné metódy optimalizácie, a to najmä pri riešení zložitých NP – úplných úloh. Patrí sem napríklad [2]:

- Numerická, kombinatorická optimalizácia,
- plánovanie a riadenie,
- inžiniersky návrh,
- dolovanie dát,
- strojové učenie, umelá inteligencia.

Je teda zrejmé, že nachádzajú uplatnenie v tých prípadoch, kedy:

- Prehľadávaný priestor je príliš rozsiahly a chýba znalosť, ktorá by umožňovala nájsť vhodné riešenia,
- nie je možná matematická analýza problému (zložité matematické funkcie, nemožnosť vyjadrenia problému pomocou analytickej rovnice) a
- úloha má príliš veľa obmedzujúcich podmienok, kritérií.

Avšak aj nasadenie evolučných algoritmov nemusí vždy zaručovať prijateľný výsledok, medzi ich najväčšie problémy patrí najmä:

- Schopnosť ohodnotiť riešenie iba relatívne – nemáme záruku že nájdené optimum je skutočne globálne a neuviazli sme iba v lokálnom extréme,
- veľká časová náročnosť,
- EA končí na základe časového obmedzenie alebo stagnácie konvergencie funkcie

V nasledujúcich kapitolách sa oboznámime so základnými typmi evolučných algoritmov, pričom z dvoch z nich (genetické programovanie a evolučné stratégie), vychádza aj kartézske genetické programovanie, ktoré bude hlavným predmetom štúdie práce.

2.4 Evolučné stratégie

Evolučné stratégie (ES) boli navrhnuté už v 60–tych rokoch 20. storočia dvomi nemeckými študentami. Patrí tak k najstarším úspešným stochastickým metódam. Evolučné stratégie majú veľa spoločného s klasickým genetickým algoritmom (dedenie genetického materiálu, mutácia, postupné evolvovanie...), no sú tu podstatné rozdiely napríklad v reprezentácii chromozómov, spôsobe použitia genetických operátorov či obnove populácie. ES sú vhodné pri optimalizácii zložitých funkcií. Na rozdiel od GA evolučné stratégie používajú na reprezentáciu chromozómov reálne čísla. Hybnou silou tu [4] najmä mutácia využívajúca Gaussovu distribučnú funkciu, kríženie sa začalo využívať až neskôr.

2.4.1 Návrh evolučnej stratégie a jej priebeh[4]:

- Voľba zloženia chromozómov.
- Voľba typu stratégie ($(\mu+\lambda)$, (μ,λ)).
- Voľba ukončujúcej podmienky.
- Pokým nie je splnená ukončujúca podmienka:
  - Kríženie rodičovskej populácie (iba u niektorých typov ES)
  - Mutácia

- Selekcia (závisí od typu stratégie)

2.4.2 Reprezentácia riešenia

Na rozdiel od binárnej reprezentácie chromozómu u GA, u ES sa používa na reprezentáciu chromozómu reálne čísla. Príklad možného jedinca je:

0.3970	-0.7710	0.7262	-0.3400	0.0028	0.7853	0.5197	-0.7962	-0.5586	0.5439

Obrázok č. 2: Reprezentácia chromozómu v ES

2.4.3 Výpočet fitness

Jej úlohou je rovnako ako u GA ohodnotiť kvalitu jedinca v populácii, používa sa rovnako ako u genetického algoritmu (s prihliadnutím samozrejme na zmenu reprezentácie).

2.4.4 Obnova populácie

Obnova populácie môže v evolučných stratégiách prebiehať dvomi spôsobmi:

1. Tzv. plus stratégia (μ+λ), kedy z rodičovskej populácie μ vygenerujeme λ potomkov pomocou operátorov kríženia alebo mutácie, zoradíme ich podľa hodnoty fitness a do ďalšej populácie sú vybrané najúspešnejšie jedince – uplatňuje sa tu elitizmus. U základnej stratégie 1+1 (jeden rodič jeden potomok), je z jedného rodiča generovaný istý počet jedincov, a ak je niektorý lepší ako rodič, tento je ním nahradený. Pripomína to gradientné metódy alebo horolezecký algoritmus [1].
2. Tzv. čiarková stratégia (μ,λ). Z rodičovskej populácie μ generujeme λ potomkov, týchto zoradíme podľa hodnoty fitness a vyberieme najúspešnejších. Rodičovská populácia je kompletne nahradená potomkami, pričom tieto môžu mať aj horšiu priemernú hodnotu fitness – to umožňuje tejto metóde opustiť lokálne minimá a úspešne optimalizovať aj veľmi zložité funkcie.

2.4.5 Technika výberu rodičovských jedincov

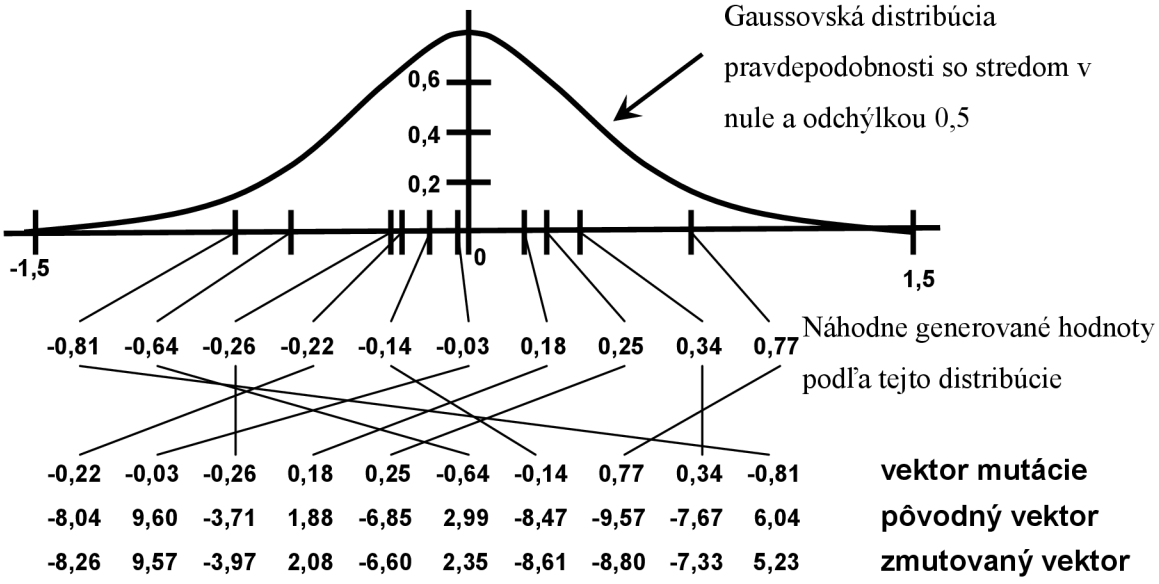
Výber rodičov prebieha náhodne, s uniformným rozložením (každý jedinec má rovnakú šancu vstupu do reprodukčného procesu).

2.4.6 Kríženie a mutácia

2.4.6.1 Mutácia

U evolučných stratégií je uvedená ako prvá mutácia, pretože je hnacím motorom evolúcie, a spočiatku bola používaná iba ona (podobne ako v kartézskom genetickom programovaní), kríženie v ES sa začalo používať až neskôr.

Na rozdiel napríklad od GA, kde nastáva zmena väčšinou iba na jednom bite chromozómu, tu je mutácia väčšinou uplatňovaná na každý prvok vektoru (ale nemusí). Využíva sa Gaussovej distribúcie pravdepodobnosti zmeny s nulovou strednou hodnotou a rozptylom σ. Často sa používa normované rozdelenie formalizované zápisom N (0,1).



Obrázok č. 3: Generovanie mutácie u ES [1]

Ako je vidieť, na základe distribučnej funkcie sú s istými pravdepodobnosťami vygenerované vektor náhodných čísiel, a tento je po patričnom premiešaní pripočítaný k pôvodnému vektoru. Prečo je gaussovská distribúcia lepšia ako zmena bitov v binárnom vektore a viac o tzv. Hammingovej bariére je možné nájsť v príslušnej literatúre [1].

Štandardná odchýlka σ sa v priebehu evolúcie mení podľa pravidla 1:5. To znamená, že ak v priebehu evolúcie dostávame väčší počet úspešne zmutovaných potomkov, potom by sme mali postupovať väčšími krokmi (zväčšiť štand. odchýlku), a naopak, pri zlých výsledkoch by sme mali postupovať pomalšie. Nech $\varphi(k)$ je koeficient úspešnosti definovaný ako pomer úspešných mutácií v priebehu posledných k iterácií k počtu k iterácií, z ktorých bola úspešnosť meraná, potom [1]:

$$\sigma' = \begin{cases} c_d.\sigma, & ak \quad \varphi(k) < 1/5, \\ c_i.\sigma, & ak \quad \varphi(k) > 1/5, \\ \sigma, & ak\varphi \quad (k) = 1/5, \end{cases}$$

kde $c_i>1$ a $c_d<1$ riadia zväčšovanie a zmenšovanie štandardnej odchýlky. V literatúre bývajú tieto koeficienty často špecifikované ako $c_d=0.82$ a $c_i=1/c_d=1.22$.

2.4.6.2 Kríženie

U evolučných stratégií sa využívajú tri typy kríženia: priemerom, diskrétne a lineárna kombinácia.

2.4.6.2.1. Kríženie priemerom

Využíva sa tu aritmetický priemer, z dvoch rodičov vzniká jeden potomok.

0.549	-0.157	0.321	-0.597	0.463	0.554	0.199	-0.759	-0.530	Rodič 1
-0.943	0.407	0.251	-0.023	0.883	0.074	-0.469	-0.240	-0.337	Rodič 2
-0.197	0.125	0.286	-0.310	0.673	0.314	-0.135	-0.499	-0.433	Potomok

Obrázok č. 4: Kríženie priemerom v ES

2.4.6.2.2. Diskrétne kríženie

Hodnoty génov potomka sú náhodne preberané od jedného alebo druhého rodiča.

0.549	-0.157	0.321	-0.597	0.463	0.554	0.199	-0.759	-0.530	Rodič 1
-0.943	0.407	0.251	-0.023	0.883	0.074	-0.469	-0.240	-0.337	Rodič 2
0.549	-0.157	0.251	-0.597	0.463	0.074	0.199	-0.240	-0.337	Potomok

Obrázok č. 5: Diskrétne kríženie v ES

2.4.6.2.3. Lineárna kombinácia

Tu vznikajú dvaja potomkovia, pričom tieto vektory sú lineárnou kombináciou rodičov. x_1 a x_2 sú rodičovské chromozómy, y_1 a y_2 ich potomci. Bod kríženia *a* je z intervalu <0,1>.

$y_1 = a.x_1 + (1-a).x_2$

$y_2 = (1-a).x_1 + a.x_2$

2.4.7 Autoevolúcia riadiacich parametrov

Okrem základného pravidla 1:5 (viď kapitola 2.5.6.1) je možné použiť aj autoevolúciu riadiacich parametrov, kedy je chromozóm tvorený dvomi sekciami – jedna je sekcia cieľových parametrov a jedna sekcia riadiacich parametrov. Detailnejší popis je možné nájsť napr v [2]. Takto evolvované parametre dosahujú lepšie výsledky ako v prípade samotného použitia pravidla 1:5.

Evolučné stratégie bývajú používané pre optimalizáciu funkcií o niekoľkých premenných alebo pri úlohách so zmiešanými spojito-diskrétne-celočíselnými parametrami (napríklad pri návrhu korekčných cievok pre nedeštruktívnu magnetickú rezonanciu).

2.5 Genetické algoritmy

Genetický algoritmus patrí medzi základné stochastické optimalizačné techniky s výraznými evolučnými črtami. V súčasnosti patrí medzi najpoužívanejšie evolučné algoritmy. Vychádza z Darwinovskej teórie [5], je robustný, aplikovateľný na širokú paletu problémov. Je časovo náročný, avšak vždy vedie aspoň k suboptimálnemu riešeniu, pričom nepožaduje znalosti o prehľadávanom priestore, zato je potrebné mať skúsenosti pri nastavovaní parametrov. Najčastejšie používané je binárne alebo permutačné zakódovanie chromozómu.

Štandardný GA algoritmus je možné popísať nasledovne [2]:

1. Nastav čas t=0, náhodne vygeneruj počiatočnú populáciu D (0) s mohutnosťou N.
2. Pomocou fitness funkcie F (X) ohodnoť všetky jedince populácie D (t).
3. Pomocou operátorov kríženia a mutácie generuj populáciu potomkov O (t) s mohutnosťou M<=N.
4. Vytvor novú populáciu D (t+1) nahradením časti populácie D (t) jedincami z populácie O (t).
5. Nastav čas t na čas t+1.
6. Pokým nie je splnená podmienka ukončenia algoritmu, pokračuj bodom 2.

V nasledujúcich podkapitolách budú popísané jednotlivé komponenty genetického algoritmu.

2.5.1 Spôsob zakódovania

2.5.1.1 Binárne zakódovanie

Najčastejším spôsobom zakódovania chromozómov v genetickom algoritme je binárne zakódovanie. Je to preferovaný spôsob nevyžadujúci žiadne špeciálne genetické operácie, má ale problémy s reprezentáciou reálnych premenných. Chromozóm budeme chápať ako binárny vektor pevnej dĺžky k:

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

Populácia P obsahuje jedincov, ktoré sú zakódované v tvare binárnych vektorov α

$P = \{\alpha_1, \alpha_2, ..., \alpha_p\}$,

kde p udáva kardinalitu populácie P (počet jedincov v populácii).

Nech f je účelová funkcia definovaná nad množinou binárnych vektorov dĺžky k

$f : \{0,1\}^k \rightarrow R$,

ktorá ohodnotí každý binárny vektor $\alpha \in \{0,1\}^k$ reálnym číslom. Našou úlohou je hľadať globálne minimum tejto funkcie nad množinou $\{0,1\}^k$

$$\alpha_{opt} = \arg\min_{\alpha \in \{0,1\}^k} f(\alpha)$$

Funkcia *f* reprezentuje prostredie, v ktorom sa dané jedince populácie nachádzajú. Mierou úspešnosti daného jedinca je jeho funkčná hodnota ohodnotená fitness funkciou.

2.5.1.2 Permutačné zakódovanie

Je možné ho previesť na binárne zakódovanie, vyžaduje špeciálne genetické operátory. Je použiteľný napríklad aj v typickom príklad aplikácie evolučných algoritmov akým je problém obchodného cestujúceho.

2.5.1.3 Stromová štruktúra

Stromová štruktúra je výhodná najmä pri VLSI návrhu a podobne. Takisto ako permutačného zakódovanie vyžaduje špeciálne genetické operátory.

2.5.2 Funkcia fitness

Slúži na ohodnotenie chromozómov v populácii. Býva to časovo najnáročnejšia etapa evolučného algoritmu, býva vykonávaná nad všetkými jedincami.

2.5.3 Populácia

Počiatočná populácia býva generovaná náhodne, prípadne pomocou nejakej heuristiky. Jej typická veľkosť u genetického algoritmu býva 30-100 jedincov.

2.5.4 Technika výberu rodičovských jedincov

Slúži na výber vhodných jedincov pre kríženie a mutáciu. Výber môže prebiehať medzi dvomi jedincami alebo skupinami jedincov, pričom býva uprednostňovaný jedinec s vyššou hodnotou fitness. V nasledujúcich podkapitolách sú vysvetlené tri techniky výberu rodičov.

2.5.4.1 Ruletový výber

Táto metóda výberu rodičovských jedincov je založená na pravdepodobnosti výberu určitého chromozómu do procesu reprodukcie – čím vyššia hodnota fitness (väčšia adaptovanosť na prostredie), tým väčšia pravdepodobnosť. Nech populácia P má mohutnosť p=5.

$P = \{a_1, a_2, a_3, a_4, a_5\}$, nech ohodnotenie fitness funkciou F (X) jednotlivých chromozómov je:

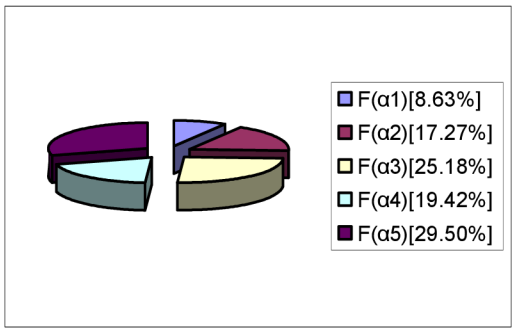
$$F(\alpha_1)=12$$

F (α_2)=24

F (α_3)=35

F (α_4)=27

F (α_5)=41



Obrázok č. 6: Pravdepodobnosť vstupu chromozómov do procesu reprodukcie

Potom pravdepodobnosť vstupu jednotlivých jedincov do procesu reprodukcie je priamo úmerná hodnote ich fitness, čo je vidieť na obrázku č.3.

2.5.4.2 Turnajový výber

Tento spôsob výberu rodičovských jedincov je dosť odlišný od predchádzajúcej metódy – najprv sú úplne náhodne vybrané dva jedince z rodičovskej populácie, ktoré medzi sebou bojujú – sú porovnané na základe ich hodnoty fitness, víťazný chromozóm je ten s vyššou hodnotou.

Výhodou tohoto výberu je ten fakt, že aj jedince s nižšou hodnotou fitness dostávajú príležitosť zúčastniť sa procesu reprodukcie, čo zabraňuje predčasnej konvergencii a uviaznutiu v nejakom lokálnom minime (je tu dostatočná diverzita).

2.5.4.3 Pravdepodobnostný výber

Kombinuje predchádzajúce dva spôsoby výberu. Na populácii sú vykonané vždy najprv dva výbery ruletovým spôsobom (čiže proporcionálne k hodnote fitness), a následne tieto dva jedince bojujú turnajovým spôsobom. Víťazný z nich postupuje do množiny jedincov určených ku kríženiu a mutácii.

2.5.5 Kríženie a mutácia

2.5.5.1 Kríženie

Je hnacím motorom genetického algoritmu, hoci býva často veľmi diskutovaný a časť vedcov tvrdí, že nemá patričný biologický podklad. Tvrdia, že sú vďaka nemu rozbíjané stavebné bloky (teória stavebných blokov je nad rámec tejto publikácie, viac je možné dočítať sa napr. v [2]).

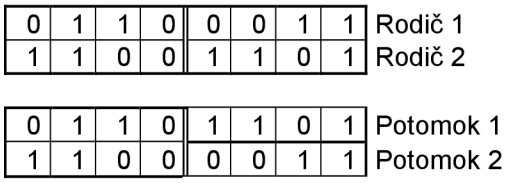
Majme dva chromozómy $\alpha, \beta \in \{0,1\}^k$, potom operáciu kríženia je definovaná ako operácia, ktorá dvom rodičovským chromozómom priradí dva nové jedince s rovnakou dĺžkou ako mali pôvodné [1]:

$(\alpha', \beta') = O_{cross}(\alpha, \beta)$.

Existuje niekoľko realizácií stochastického operátora kríženia, rozdielne napríklad pre binárne alebo permutačné zakódovanie chromozómu. Uvedieme si iba základné jednobodové kríženie pri binárnom zakódovaní, ostatné sú podobné; kríženie u permutačného zakódovania je pre nás nezaujímavé.

2.5.5.1.1. Jednobodové kríženie

Majme dva chromozómy, vyberme náhodne bod kríženia. Potom dostávame dvoch potomkov, kde prvý potomok má prvú časť z prvého rodiča a druhú za bodom kríženia z druhého. Druhý potomok naopak – viď obrázok č. 7.



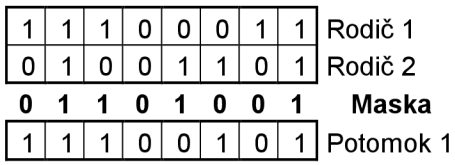
Obrázok č. 7: Jednobodové kríženie v GA

2.5.5.1.2. Dvojbodové kríženie

Prebieha podobne, len sú vybraté 2 body kríženia a do potomka sa vyberá striedavo časť z jedného a druhého rodiča.

2.5.5.1.3. Uniformné kríženie

Využíva náhodne vygenerovanú binárnu masku, na základe ktorej sa vyberá časť genetického materiálu raz z jedného raz z druhého rodiča. Avšak na rozdiel od predchádzajúcich spôsobov kríženia vzniká iba jeden potomok.



Obrázok č. 8: Uniformné kríženie pomocou masky

2.5.5.2 Mutácia

Operácia mutácia stochasticky transformuje binárny vektor α na binárny vektor α'. Hromadná mutácia sa využíva iba v ojedinelých prípadoch (v kartézskom genetickom programovaní je avšak jediným využívaným operátorom). Jej pravdepodobnosť býva u genetického algoritmu väčšinou v rozsahu 0,1 – 10%, pričom máva buď konštantný alebo exponenciálny priebeh. U binárneho zakódovania chromozómov je pri mutácii náhodne vybraný jeden bit, a jeho hodnota je invertovaná. U permutačného zakódovania sú náhodne vybrané dve pozície (čísla) v chromozóme, a tieto sú vymenené.

2.5.6 Obnova populácie

Najpoužívanejšie varianty obnovy populácie sú:

- Generatívna výmena, kedy je celá rodičovská populácia nahradená potomkami a
- čiastočná obnova, kedy iba jeden jedinec s najnižšou hodnotou fitness z rodičovskej populácie je nahradený potomkom.

Často sú používané varianty kombinujúce tieto dve varianty. Pri čiastočnej obnove sú používané rôzne techniky náhrady pôvodnej populácie, a to:

- Podľa kvality (veľkosť populácie zostáva zachovaná),
- turnaj (bojujú spolu jedince rodičovskej populácie a potomkov),
- elitizmus (časť najúspešnejších jedincov rodičovskej populácie je skopírovaná do nasledujúcej bez kríženia či mutácie),
- faktor premnoženia, kedy náhodne vybranú skupinu rodičov nahradzujú genotypicky podobné jedince z potomkov.

2.6 Genetické programovanie

Genetické programovanie ako súčasť evolučných algoritmov vzniklo koncom 80–tych rokov minulého storočia. Je aplikovateľné na rôzne problémy, medzi najčastejšie patrí napríklad symbolická regresia a vývoj programov, no dosiahlo úspechy aj pri návrhu kombinačných a kvantových obvodov, mechanických systémov, či antén. Je odlišné od klasického genetického algoritmu (viz. kapitola 2.6), a to v nasledujúcich troch krokoch [2]

- Reprezentácia. Genetické programovanie (GP) pracuje s tzv. spustiteľnými štruktúrami (programy, elektronický obvod...). Najzásadnejším rozdielom oproti klasickému genetickému algoritmu je ten, že chromozóm v GP má premenlivú dĺžku, s čím súvisia isté výhody aj problémy (viď nižšie).
- Genetické operátory pracujú nad spustiteľnými štruktúrami – existuje tu okrem operátorov kríženia a mutácie množstvo operátorov, ktoré umožňujú pracovať s modulmi, podprogramami a podobne.
- Pri výpočte hodnoty fitness daného chromozómu je vykonaný kód programu (chromozómu) pre danú množinu vstupov a sú vyhodnotené získané výsledky.

Pred začiatkom riešenia úlohy genetickým programovaním, je nutné definovať 5 prípravných krokov:

- Definícia množiny terminálov (nachádzajú sa v listoch stromu).
- Definícia množiny funkcií.
- Definícia spôsobu výpočtu fitness.
- Definícia parametrov GP (počet generácií, veľkosť populácie).

- Definícia spôsobu určenia výsledku a spôsob ukončenia evolúcie.

2.6.1 Množina terminálov

Reprezentuje vstupy, teda premenné, konštanty a funkcie bez argumentov, napríklad pri symbolickej regresii môžu patriť do tejto množiny prirodzené alebo reálne čísla, konštanta neznámej 'x', iné premenné a podobne.

2.6.2 Množina funkcií

Veľkosť tejto množiny je prakticky neobmedzená, je závislá na type riešenej úlohy. Pri symbolickej regresii sem môžu patriť rôzne aritmetické alebo goniometrické funkcie, pri evolvovaní programov sem patria napríklad konštrukcie daného programovacieho jazyka (cykly, podmienky, skoky…).

Voľba množiny funkcií je síce neobmedzená, ale musí byť volená opatrne. Pri používaní jednoduchých funkcií nie je možné riešiť náročnejšie úlohy, avšak pri nadmernom počte použitých funkcií príliš narastá prehľadávaný priestor a nemusíme sa dostať ku konečnému riešeniu.

2.6.3 Výpočet fitness

Je takisto závislý na type konkrétnej úlohy. Napríklad pre výpočet fitness pri hľadaní programov je možné fitness počítať ako pomer správnych výstupov na dané vstupy k celkovému počtu (najlepší jedinec teda bude mať hodnotu fitness 1), zatiaľ čo u problému symbolickej regresie je možné fitness rátať napr. ako sumu kvadratických odchýlok výstupných bodov získaných vyhodnotením daného jedinca k požadovanej funkcii (tu má najlepší jedinec hodnotu fitness 0).

V literatúre sú udávané 4 termíny pre fitness hodnoty: hrubá, štandardizovaná, prispôsobená a normalizovaná fitness [2].

Definícia parametrov GP a spôsobu ukončenia evolúcie je plne v rukách vývojára, pričom je možné použiť rozličné prístupy s rôznymi výsledkami v závislosti od typu úlohy (veľké populácie a malý počet krokov evolúcie alebo naopak).

Algoritmus GP je podobný tomu u genetickému algoritmu:

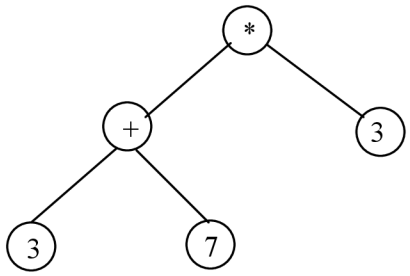
- Náhodná inicializácia populácie.
- Vyhodnotenie fitness funkciou každého chromozómu v populácii.
- Pokým nie je nová generácia plne obsadená, opakujeme nasledovné kroky:
    - Zvoleným selekčným algoritmom vyberieme jedincov.
    - Nad vybranými jedincami (jedincom) uskutočníme genetické operácie.
    - Výsledok týchto operácií vložíme do novej generácie.

- Pokým nie je splnená podmienka na ukončenie algoritmu pokračujeme bodom 2.
- Výstupom algoritmu je jedinec s najvyššou hodnotou fitness.

V nasledujúcich podkapitolách budú uvedené rozdiely v komponentách oproti klasickému genetickému algoritmu.

2.6.4 Reprezentácia riešenia

Ako bolo spomenuté, v genetickom programovaní sú chromozómy najčastejšie reprezentované stromovými štruktúrami (autor metódy J. Koza pracoval pôvodne v LISP-e). Uvažujme problémy symbolickej regresie, kedy máme zadané vstupné a výstupné hodnoty funkcie a hľadáme jej predpis, potom príklad takéhoto chromozómu môže byť na obr. 9:



Obrázok č. 9: Príklad chromozómu v GP

2.6.5 Spôsob výpočtu fitness

Spôsob výpočtu fitness v GP bol popísaný v úvode kapitoly 2.7.

2.6.6 Populácia

Generovanie počiatočnej populácie je prvým krokom evolučného algoritmu. Rovnako ako pri genetickom algoritme aj pri GP je počiatočná populácia generovaná náhodne, avšak riadi sa istými pravidlami pri zostavovaní stromu. Majme definované množiny funkcií a terminálov, a takisto je nutné definovať maximálnu hĺbku stromu. Rozlišujeme tri základné metódy generovania počiatočnej populácie:

Grow – uzly stromu sú náhodne vyberané z množiny terminálov alebo funkcií. Akonáhle je vybraný terminál, je ukončené generovanie vetvy stromu, hoci nemusela dosiahnuť maximálnej hĺbky.

Full – pri tejto metóde sa náhodne vyberajú iba prvky z množiny funkcií, až pokým nie je dosiahnutá maximálna hĺbka. Strom sú pravidelné, na rozdiel od prechádzajúcej metódy kde dostaneme väčšinou nepravidelné stromy.

Ramped Half-and-Half – pomocou tejto metódy dostávame populáciu s dostatočnou diverzitou. Je takisto definovaná maximálna hĺbka stromu *h*, pričom sú postupne generované stromy s maximálnou hĺbkou *2, 3,...,h*, kde na polovicu stromov každej dĺžky je použitá metóda grow a na druhú polovicu metóda full.

2.6.7 Technika výberu rodičovských jedincov

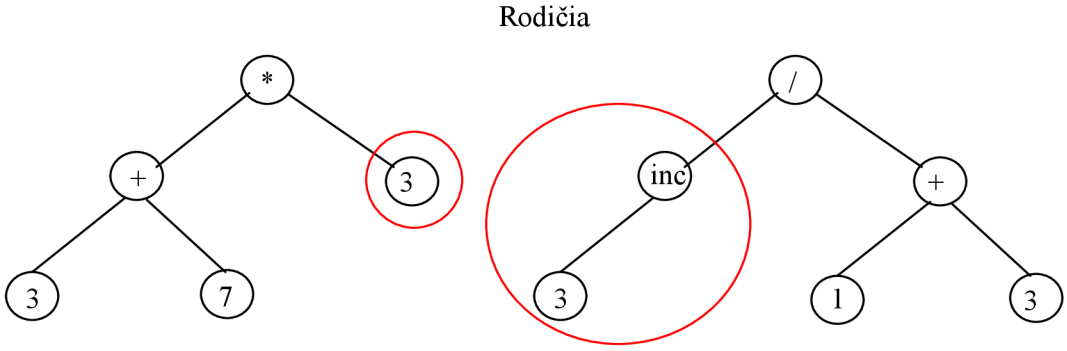
Pracuje obdobne ako u genetického algoritmu (viď kapitola 2.6).

2.6.8 Kríženie a mutácia

Pracujú obdobne ako u GA, avšak z dôvodu odlišnej reprezentácie chromozómov sa s nimi pracuje trochu iným spôsobom. Uvažujme iba stromovú reprezentáciu, potom pri krížení dostávame jedince rozličnej hĺbky stromu. To vedie aj k určitým problémom genetického programovania (viď kapitola 2.7.10).

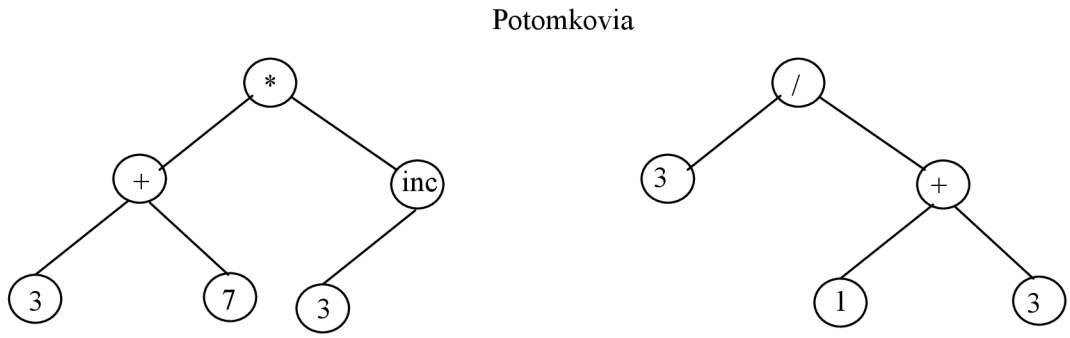
2.6.8.1 Kríženie

Je považované za základný operátor GP, pričom existuje niekoľko variant (v závislosti na type aplikácie GP), zostaňme u uvažovanej symbolickej regresie. Operátor kríženia kombinuje genetický materiál 2 rodičov. Na základne nejakej selekčnej metódy podobne ako u genetického algoritmu sú vybrané rodičovské jedince, a v nich sú náhodne vybrané podstromy v rodičovských jedincoch, a tieto sú medzi sebou vymenené. Takýmto spôsobom dostaneme dvoch potomkov, ktoré sú vložené do populácie.



Obrázok č. 10: Prvý bod kríženia v GP – výber podstromov

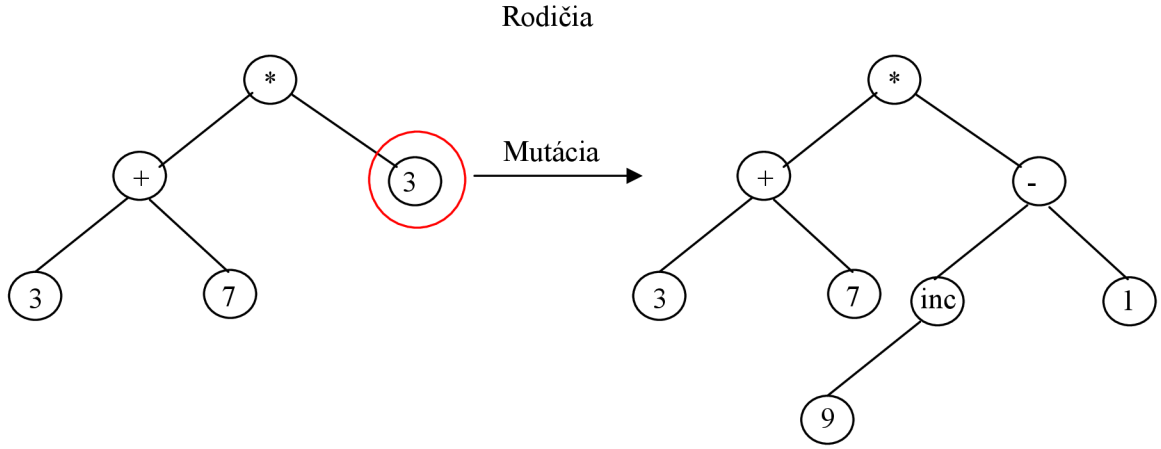
Po procese kríženia dostávame:



Obrázok č. 11: Druhý bod kríženia v GP – výmena podstromov

2.6.8.2 Mutácia

Pravdepodobnosť mutácie u GP býva okolo 5%. Existuje zase niekoľko variánt (napríklad na základe toho, či obsahuje uzol terminál alebo funkcia, môžeme zmeniť náhodne jeho hodnotu), najčastejšie sa používa ten spôsob, kedy sa náhodne vyberie podstrom, a ten je nanovo vygenerovaný nejakou metódou (viď úvod kapitoly 2.7).



Obrázok č. 12: Mutácia v GP

2.6.9 Obnova populácie

Prebieha obdobne ako u štandardného GA.

2.6.10 Problémy GP

Patrí medzi ne problém s konvergenciou populácie (J. Koza tvrdí, že ak sa nenájde riešenie do 50 generácií, spravidla už sa nenájde, pričom pracuje s tisíckami jedincov), potom je to príliš veľká množina terminálov, funkcií, alebo ich nevhodná voľba, a takisto problémy prameniace zo spôsobu reprezentácie chromozómov (bloat).

2.6.10.1 Introny a bloat

Introny sú také časti programu, ktoré nemajú sémantický význam – napríklad a=a+1-1. Počas evolúcie narastá ich počet a vedie k ďalšiemu problému – bloat (nafúknuť sa), čo znamená, že chromozómy príliš rastú a majú značné ako pamäťové nároky, tak aj časové na ich ohodnotenie fitness funkciou. Jedným z čiastočných riešení môže byť penalizácia príliš veľkých chromozómov.

2.6.10.2 Škálovateľnosť

Nie je doposiaľ možné vyvíjať príliš zložitý program alebo komplikovaný predpis matematickej funkcie (prípadne iné), pretože toto vedie k nutnosti použitia veľkých chromozómov, z čoho ale vyplýva nutnosť prehľadávania veľkého stavového priestoru, čo vedie k značnej neefektivite.

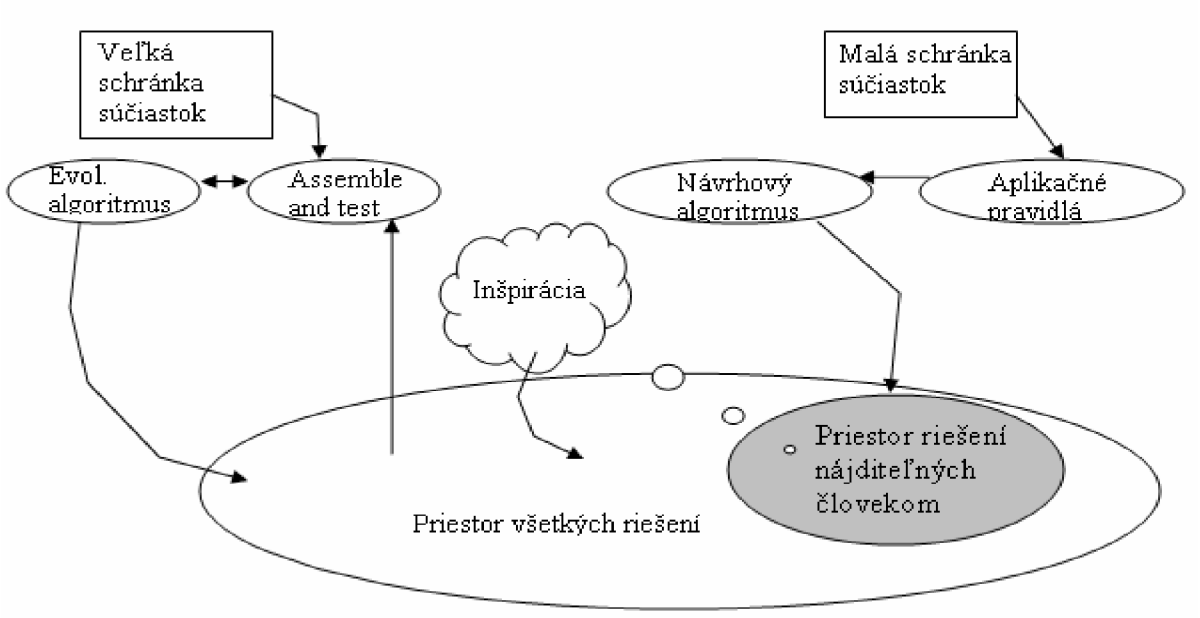
2.6.10.3 Zovšeobecňovanie

Požadujeme ju pri väčšine metód umelej inteligencie, kedy chceme, aby naše riešenie bolo schopné dostatočne zovšeobecňovať pre všetky testovacie dáta. Ak výsledok evolúcie pracuje dobre pre trénovaciu množinu, ale nie pre testovaciu (hoci má tento jedinec vysokú hodnotu fitness), dochádza k tzv. overfitting – u (obdobné pretrénovaniu z neurónových sietí).

3 Kartézske genetické programovanie

3.1 Návrh digitálnych obvodov pomocou evolučných algoritmov

Tradičné číslicové systémy akými sú napríklad počítače, smerovače, či mobilné telefóny, boli navrhnuté pomocou komplexných pravidiel a zásad pre tvorbu číslicových zariadení. Tento spôsob návrhu je nazývaný zhora nadol, keď sa začína so všeobecnou a abstraktnou špecifikáciou funkcie systému a rozdelením na jednotlivé podsystémy, pričom pohybom smerom nadol dochádza k zjemňovaniu návrhu a presnejšej špecifikácii architektúry a konkrétnych funkcií podsystémov. Tento spôsob je v príkrom rozpore s tým, čo je použité pri evolučnom návrhu číslicových obvodov. Na začiatku dizajnu sú vygenerované konkrétne jedince, zakódované istým spôsobom, ktorý reprezentuje číslicový obvod. Vhodnosť tohto jedinca je určená tým, ako dobre plní svoju požadovanú funkciu. Schopnosť prežitia tohto organizmu je možné vidieť ako proces skladania väčšieho systému z jednotlivých menších komponent a testovania organizmu v prostredí, v ktorom sa nachádza. Tento proces je v tomto dokumente (vypracovanom na základe [6]) nazývaný zostaviť a testovať (assemble-and-test), viď obrázok č. 13. Tento proces umožňuje prehľadať oveľa väčší priestor ako ten, ktorý je možné preskúmať pomocou striktných návrhových pravidiel (priestor zobrazený šedou farbou), a to práve z dôvodu absencie týchto obmedzení.



Obrázok č. 13: Konvenčný návrh vs. evolučný návrh s využitím zlož-a-testuj [6]

Koncept assemble-and-test spolu s evolučnými algoritmami postupne zlepšoval kvalitu návrhu bol do značnej miery prijatý aj novovznikajúcim odvetvím Evolvable Hardware, ktorého úlohou je zostaviť elektronický obvod. Obvody sú zakódované do genetypov, z ktorých sú konštruované konkrétne obvody (jedince) alebo fenotypy. Výskum v oblasti Evolvable Hardware môže byť rozdelený do dvoch kategórií: vnútorná (intrinsic) evolúcia a vonkajšia (extrinsic) evolúcia. Vnútorná evolúcia odkazuje na evolučný proces, v ktorom je zostavený konkrétny jedinec (fenotyp)v hardvéri a je testovaný (umožňuje podstatne zrýchliť proces evolúcie). Druhá používa softvérový model hardvéru, a konkrétne jedince sú ohodnotené softvérovo. Každá z týchto dvoch kategórií je deliteľná na ďalšie dve podkategórie, a to analógovú a digitálnu (ktorou sa budem zaoberať aj v štúdii).

Zásadnou otázkou je, či pri evolučnom vývoji obvodov dokážeme abstrahovať všeobecné princípy, a objaviť nové metódy návrhu obvodov. V literatúre [6] je uvádzané, že je to možné, pričom je však potrebné brať aj ohľad na veľkosť takto produkovaných obvodov, keďže napríklad pomocou kartézskeho genetického programovania dostávame rozumné výsledky iba pre obvody s maximálnym počtom približne 10 vstupov, ďalej sa už evolúcia stáva časovo veľmi náročnou z dôvodu extrémneho nárastu prehľadávaného stavového priestoru. Proces objavovania nových návrhových pravidiel je možné chápať ako formu dolovania dát [6].

3.2 Popis CGP

Kartézske genetické programovanie po prvýkrát predstavil vo svojom článku J. Miller v roku 1999 a o rok neskôr v spolupráci s P. Thomsonom. V štandardnej forme kartézskeho genetického programovania (CGP) je problém reprezentovaný vo forme acyklického orientovaného grafu (spomínané genetické programovanie v kapitole 2.7 používa ako reprezentáciu stromy rozličnej hĺbky), ktorý je zakódovaný pomocou reťazca celých čísel pevnej dĺžky. Toto je základný rozdiel oproti genetickému programovaniu, kde môžu mať jednotlivé chromozómy rozličnú veľkosť. Uzly grafu sú usporiadané v dvojrozmernej mriežke a predstavujú jednoduché logické funkcie (AND, XOR, OR…). Tento spôsob reprezentácie do značnej miery zodpovedá odpovedá štruktúre FPGA (obvody FPGA sú zostavené z matice logických blokov).

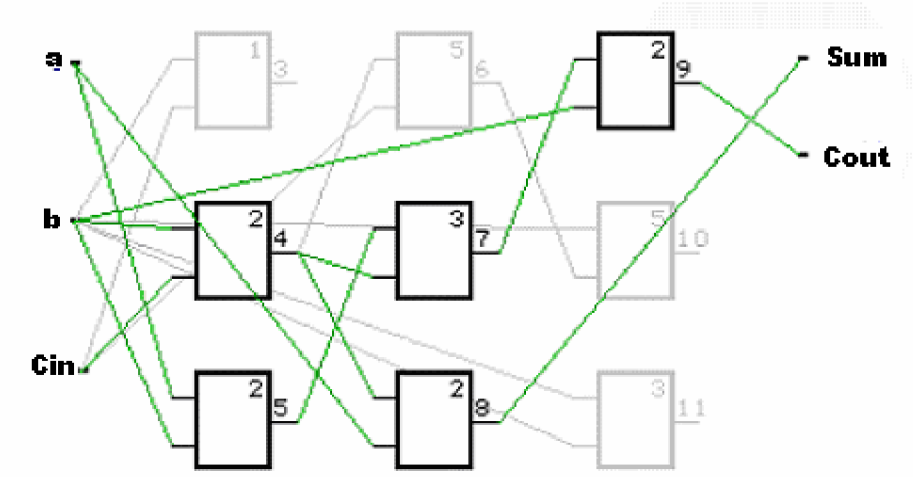
3.2.1 Reprezentácia problému v CGP

V CGP je programovateľný obvod reprezentovaný ako 2-D pole n_c (stĺpcov) x n_r (riadkov) programovateľných elementov. Rovnako, ako počet vstupov (n_i) a výstupov (n_o) obvodu, je i rozmer mriežky napevno daný. Vstup uzlu logického obvodu môže byť pripojený buď k vstupu, alebo ktorémukoľvek výstupu uzlu zo stĺpcov pred stĺpcom, v ktorom sa daný uzol nachádza. Nie je možné prepájanie sa v rámci jedného stĺpca alebo za daný stĺpec (kvôli komplikovanosti realizácie spätnej väzby). Vznikajú takto teda výlučne kombinačné logické obvody. Ďalej je potrebné definovať počet vstupov uzlu (n_n štandardne 2) a množinu F, ktorá obsahuje logické členy (funkcie), ktoré môžu

jednotlivé programovateľné elementy (uzly) realizovať. Kardinalita množiny F je označovaná ako n_f. Posledným a veľmi dôležitým parametrom, určujúcim prepojiteľnosť jednotlivých elementov v rámci obvodu je tzv. *l-back* parameter (L), ktorý určuje, v rámci koľko predchádzajúcich stĺpcov je možné pripájať vstupy elementov daného stĺpca k ich výstupom. Napríklad, ak je L = 1, je možné pripájať sa iba k výstupom predchádzajúceho stĺpca. Pre L = n_c, je možné prepojenie v rámci všetkých predchádzajúcich stĺpcov. Pri L = 1 je možné jednoduché zreťazenie linky a urýchlenie výpočtu až n_c+1-krát, na druhej strane väčšia hodnota l-back parametru umožňuje väčšiu variabilitu a možnosť prehľadať väčší priestor možných riešení (v pokusov výplyva, že toto je vhodné najmä pri evolúcii násobičiek [7,8]). V závislosti od implementácie, niektoré z nich dovoľujú pripojiť vstup elementu priamo na vstup obvodu, hoci toto nie je umožnené parametrom L. Celková dĺžka chromozómu ΔCGP (počet génov) je vyjadriteľná ako:

$\Delta_{CGP} = n_r.n_c.(n_n+1)+n_o$.

Na obr. č. 14 je uvedená schéma úplnej jednobitovej sčítačky s prenosom.



Obrázok č. 14: Schéma úplnej jednobitovej sčítačky s prenosom vykreslená zobr. programom [7]

Tento obrázok odpovedá chromozómu:

Genotyp: (1,2,1) (1,2,2) (0,1,2) (4,2,5) (5,4,3) (4,0,2) (7,1,2) (1,6,5) (1,1,3) (8,9)

Chromozóm obsahuje v našom prípade n_r x n_c trojíc, pričom každá táto trojica reprezentuje konkrétny programovateľný element. Prvé číslo udáva číslo prvého vstupu, druhé číslo druhého vstupu a posledný tretí parameter určuje index funkcie realizovanej daným elementom. Na konci chromozómu je n-tica (v našom prípade dvojica), ktorá má rovnaký počet členov ako je počet výstupov daného obvodu. Označuje čísla výstupov obvodu, ktoré vedú priamo na výstup. Číslovanie výstupov začína od nuly, pričom index 0 dostáva prvý vstup, výstup prvého elementu má hodnotu n_i+1. Takto sa v číslovaní pokračuje po stĺpcoch.

Fenotyp

Hoci má genotyp v CGP fixnú dĺžku, fenotyp má dĺžku premenlivú; tá závisí od počtu použitých elementov. Spôsob vytvorenia fenotypu je jednoduchý – postupuje sa smerom od výstupov obvodu

po indexoch uzlov smerom k vstupom. Ako je vidieť, v tomto prípade má fenotyp dĺžku 5, pretože sú aktívne iba uzly s indexami výstupov: 4, 5, 7, 8, 9 (zvýraznené na obrázku).

Pravdivostná tabuľka daného obvodu:

Vstupy			Výstupy	
x	y	carry	out1	out0
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Tabuľka č. 1: Pravdivostná tabuľka úplnej jednobitovej sčítačky s prenosom

Hodnoty jednotlivých parametrov CGP (pre daný príklad):

$n_r = 3$ (počet riadkov)

$n_c = 3$ (počet stĺpcov)

$n_i = 3$ (počet vstupov)

$n_o = 2$ (počet výstupov)

$n_n = 2$ (počet vstupov elementu)

L = 3 (l-back parameter)

F = {NAND (0), NOR (1), XOR (2), AND (3), OR (4), NOT (5)} (množina funkcií)

$n_n = 6$ (kardinalita množiny F)

3.2.1.1 Vlastnosti kódovania problému v CGP

Kódovanie v CGP spôsobuje redundanciu, a to z dôvodu:

- Niektoré uzly nemusia byť použité (šedé na obrázku č. 13),
- niektoré uzly nemusia používať všetky vstupy (NOT, WIRE - prepojka),
- funkcie určitej skupiny uzlov môžu byť nahradené menším počtom (dvojitá negácia, NOT (AND (A)) = NAND (A) atp.), je to obdobné ako u GP.

Ďalej tu existuje neutralita – tj. ak dva fenotypy sú vzájomne neutrálne, ak majú rovnakú hodnotu fitness.

3.2.2 Výpočet hodnoty fitness

V priebehu evolúcie je potrebné ohodnotiť vhodnosť zapojenia konkrétneho jedinca (viď kapitola 3.2.6), aby bolo možné určiť jeho ďalší osud v procese evolúcie. Fitness funkcia tu teda slúži ako funkcia, ktorá určuje mieru funkčnosti zapojenia [7]. V našom prípade je fitness funkcia u CGP hodnota u konkrétneho jedinca počet správne vypočítaných bitov výstupu. Predstavme si úplnú

definíciu napr. úplnej jednobitovej sčítačky, kedy pri troch vstupoch dostávame 2*2*2 – 8 možných kombinácií vstupných vektorov. Tieto sú postupne prikladané na vstupy testovaného obvodu, a na základe zapojenia konkrétneho jedinca (fenotypu), je vypočítaná hodnota fitness, ako počet správne vypočítaných bitov (vypočítaná hodnota sa porovná s tou zadanou v pravdivostnej tabuľke). V tomto prípade (2 výstupy), by bola maximálna možná fitness jedinca 2*8, tj. 16. Tento výpočet sa v praxi urýchľuje tak, že na vstupy obvodu nie sú prikladané jednotlivé bity, ale naraz celé čísla. Vzhľadom na to, že väčšina „bežných" programovacích jazykov ako je C, Java obsahujú bitové operátory, je možné pracovať s celými číslami na bitovej úrovni. To znamená, že na vstup obvodu privedieme naraz jeden bit, ale napr. 32bitov (veľkosť jedného čísla integer v 32 – bitovej aritmetike), prípadne 64. Týmto je znížený rád zložitosti o 2^5(2^6)a výpočet je urýchlený 32 – násobne (64).

Táto jednoduchá fitness zohľadňovala iba fukčnosť zapojenia, no nie také parametre akými sú napríklad spotreba, počet použitých elementov a iné. V praxi sa zväčša postupuje tým spôsobom, že najskôr je nájdený požadovaný obvod, a po jeho nájdení sa ďalej evolvuje obvod s ohľadom na ďalšie požadované parametre.

3.2.3 Inicializácia populácie

V kapitole 3.2.6 je možné sa dočítať, že algoritmus CGP pracuje na podobnom princípe ako evolučné stratégie (1+λ). Pričom je využívaný iba operátor mutácie. Počiatočná generácia je vygenerovaná úplne náhodne o počte (1+λ) jedincov (zvyčajne 1+4); prípadne je možné použiť ako prvotnú populáciu už existujúce riešenia a tie sa pokúšať ďalej v procese evolúcie vylepšovať.

3.2.4 Technika výberu rodičovského jedinca

Rovnako ako u ES je z jedného vybratého jedinca vygenerovaných pomocou operátoru mutácie jeho λ mutantov. Jednou zásadnou zmenou u CGP je to, že ak existuje jedinec s rovnakou hodnotou fitness, ktorý ešte nebol použitý pri generovaní novej generácie, použije sa ten, a nie už použitý. Toto zabezpečuje diverzitu populácie a pomáha predchádzať predčasnej konvergencii procesu evolúcie.

3.2.5 Mutácia

V CGP je mutácia používaná ako jediný genetický operátor. Jej veľkosť je voliteľná, čo vyžaduje istú dávku skúsenosti a citu. Rozoznávame dva typy mutácie: adaptívnu a neutrálnu.

3.2.5.1 Adaptívna mutácia

Adaptívna pracuje na aktívnych génoch, pričom využíva nahromadeného užitočného genetického materiálu. Náhodne je vybraný gén, a tento je zmutovaný, pričom keďže chromozóm obsahuje jednotlivé gény a na konci pripojenie výstupov, je možné mutovať:

- Pripojenie vstupov elementu (uzlu obvodu),

- funkciu elementu (tu teda i prípadne jeho aritu) a
- výstupy obvodu.

Je teda zrejmé, že po mutácii sa z niektorých uzlov môžu stať uzly aktívne, alebo naopak neaktívne.

3.2.5.2 Neutrálna mutácia

Neutrálna mutácia je taká, ktorá nemení fitness hodnotu jedinca. Môže sa stať, že dostávame iný fenotyp, s rovnakou fitness hodnotou, alebo je zmutovaný neaktívny gén, ktorý teda samozrejme rovnako nemá vplyv na fitness hodnotu jedinca; napriek tomu aj tento typ mutácie má významný vplyv na zachovanie diverzity, a môže sa užitočne prejaviť najmä v budúcich generáciách.

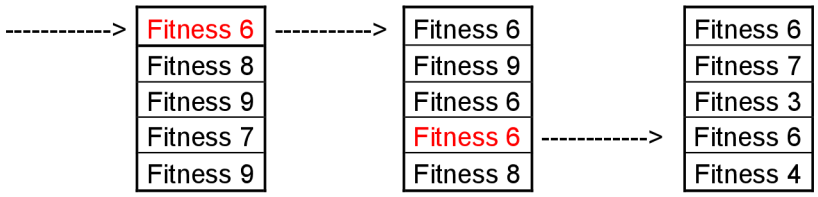
3.2.6 Algoritmus CGP

Algoritmus kartézskeho genetického programovania je založený na evolučnej stratégii typu (1+λ), zvyčajne (1+4). Drvivá väčšina štúdií objasňuje, že operátor uniformného kríženia je pre CGP nevhodný, a z toho dôvodu sa používa iba mutácia, avšak v literatúre je možno nájsť aj CGP využívajúce kríženie, ale aby bolo účinné je potrebné zmeniť reprezentáciu chromozómov [11]. Počet zmutovaných génov je udaný riadiacim parametrom určujúcim tento počet, zvyčajne sa však udáva ako prínosné 2 – 3 zmutovaných génov v každej mutácii chromozómu. Treba mať na pamäti to, že nie všetky hodnoty v génoch sú prípustné (treba brať ohľad napr. na l-back parameter a podobne).
Samotný algoritmus CGP prebieha nasledovne [6]:

- Náhodné vygenerovanie 1+λ jedincov (alebo použitie už existujúcich riešení) pre inicializáciu populácie.
- Ohodnotenie všetkých jedincov populácie pomocou fitness funkcie.
- Skopírovanie najlepšieho jedinca do novej populácie.
- Vygenerovanie λ potomkov tohoto jedinca pomocou mutácie.
- Pokým nie je splnená podmienka pre ukončenie algoritmu, pokračuj krokom 2.

3.2.7 Obnova populácie

Z danej generácie je teda vybraný jedinec s najlepšou hodnotou fitness. Ak sú dva s rovnakou hodnotou, použije sa ten, ktorý ešte nebol mutovaný, ak je takýchto viac, vyberá sa náhodne. Tento proces je možné vidieť na obrázku č. 14 (tu nižšia numerická hodnota znamená vyššiu fitness) [8].



Obrázok č. 14: Vývoj fitness hodnôt 5 jedincov [8]

3.2.8 Varianty CGP

Okrem klasického CGP je možné nájsť informácie aj o CGP využívajúcom moduly, viacero chromozómov alebo CGP s možnosťou spätnej väzby [8].

3.2.9 Zhodnotenie

Pomocou evolučných algoritmov (a teda aj CGP) môžeme prehľadávať oveľa väčší priestor možných riešení ako nám ponúkajú klasické návrhové metódy. CGP môže byť úspešne aplikované na rôzne problémy, okrem tu zmieňovaného návrhu kombinačných logických obvodov to je symbolická regresia (keď vstupy nie sú binárne, ale nadobúdajú hodnoty konštánt a premenných, pričom uzly „obvodu" realizujú rozličné matematické funkcie), ďalej je to návrh číslicových filtrov, obrazových operátorov, kontrolérov pre robotov a iné. CGP umožňuje objaviť nové (alebo „staré") patentovateľné riešenia (napr. najlepšie zapojenia obvodov pre medián, radiacu sieť alebo násobičky). Samozrejme trpí aj problémami, súvisiace najmä s maximálnym počtom vstupov (rozumné výsledky dáva CGP do asi 10 vstupov, potom narastá doba evolúcie na dni až týždne a podstatne sa znižuje schopnosť nájsť požadované riešenie), alebo s voľbou parametrov, pričom ich vhodná voľba môže značne urýchliť proces evolúcie. Pre urýchlenie výpočtu je možné výhodne používať paralelizáciu. Ďalšie informácie, najmä o výsledkoch experimentov s CGP, je možné nájsť napr. v [6][7][8].

4 Technika Age Layered Population Structure

Táto pomerne nová technika (článok, v ktorom je popísaná bol uverejnený Gregorym S. Hornbym v roku 2006), má za úlohu predchádzať predčasnej konvergencii, problému týkajúceho sa aj evolučných algoritmov.

V prvej podkapitole je objasnený problém predčasnej konvergencie a jeho možné príčiny, ako aj spôsoby, ako jej je možné zabrániť. Ďalej nasleduje popis techniky ALPS a jej využitie v evolučných algoritmoch.

4.1 Konvergencia

Úlohou algoritmu je nájsť globálne maximum (alebo minimum) danej funkcie pri prehľadávaní stavového priestoru. Ak nevieme presne, kedy algoritmus dosiahol globálneho extrému (napr. pri funkciách s viacerými premennými a rozoklaným priebehom), evolučný algoritmus ukončí svoj priebeh buď pri vyčerpaní maximálneho počtu generácií, alebo napríklad vtedy, keď väčšina jedincov populácie si je vzájomne veľmi podobná, alebo rovnaká. Majú tzv. vysokú uniformitu. Vtedy hovoríme, že populácia skonvergovala, ak toto platí pre isté percento jedincov populácie – jednotlivé gény chromozómov majú rovnaké hodnoty a ich vzájomné kríženie alebo mutácia neprispievajú k ďalšiemu vývinu. [10]

4.1.1 Predčasná konvergencia

Predčasná konvergencia je problém vyskytujúci sa aj v evolučných algoritmoch. Hovoríme o ňom vtedy, ak populácia skonverguje do lokálneho extrému, a nedokáže nájsť extrém globálny. Existuje viacero príčin predčasnej konvergencie:

- **Vysoký selekčný tlak** – pri výbere jedincov, ktoré sa uplatňujú v reprodukčnom procese, musíme vyberať najmä tie s vysokou hodnotou fitness, na druhej strane musíme zachovať dostatočnú mieru diverzity v populácii. Pri vysokom selekčnom tlaku populácia rýchlo skonverguje, no existuje tu vysoké riziko toho, že sa uprednostnia najmä jedince s vysokou hodnotou fitness, čo môže znamenať že sa dostaneme do lokálneho extrému, a algoritmu sa nepodarí dostať sa ďalej v prehľadávanom priestore. Viac napr. v [2]
- **Selekčný šum (genetický drift) –** vplyvom genetických operátorov sa jedince spočiatku rovnomerne rozložené v rámci stavového priestoru zhlukujú v okolí jedného bodu, čo môže ale nemusí byť globálny extrém.

- **Účinok genetických operátorov, poškodzujúcich genetický materiál** – pôsobia tak proti konvergencii.

Existuje niekoľko metód potlačujúcich predčasnú konvergenciu [10]:

1. Ovplyvňovanie veľkosti populácie (vo väčšej populácii dlhšie trvá kým sa genetický materiál eliminuje);
2. potlačenie príčin - nastavenie selekčného tlaku a pravdepodobnosti kríženia (zníženie);
3. infúzia nového genetického materiálu (genetic load, odstránenie duplikácií, reinicializácia);
4. podpora rozmiestnenia v priestore (úprava vhodnosti - snaha o rovnomernú distribúciu v celom priestore, napr. zdieľanie - čím viac rovnakých jedincov, ich vhodnosť sa znižuje);
5. podpora subpopulácií (obmedzenie párenia - pária sa iba podobné druhy, preselection - neprežije najpodobnejší teda rodič, crowding - neprežije najpodobnejší zo vzorky, prahovanie - nejakou metrikou sa určí vzdialenosť resp. podobnosť k, aj prah pre k, čo je podmienka výberu).

Techniku ALPS je do istej miery podobná 5. bodu, teda podpore subpopulácií, kedy sú vzájomne krížené iba podobné druhy (v prípade ALPS podobné vekom).

4.2 Možné riešenia predčasnej konvergencie

Informácie v tejto kapitole sú čerpané z článku [9]. Predčasná konvergencia je bežným problémom evolučných algoritmov, s ktorým sa stretneme najmä pri zložitejších úlohách. Nastáva vtedy, keď pomocou genetických operátorov už nie sme schopní produkovať nové jedince, ktoré by sa dokázali „vyliezť“ z lokálneho extrému a ďalej pokračovať v prehľadávaní stavového priestoru. Na potlačenie predčasnej konvergencie bolo zavedených niekoľko metód (viď kapitola 4.1.1), ktoré sú viac či menej úspešné. Napriek týmto pokusom sa nepodarilo zostrojiť robustný evolučný algoritmus, ktorý by dokázal nájsť vždy riešenie blízko optima (globálneho extrému).

Najjednoduchšou cestou za účelom redukcie pravdepodobnosti predčasnej konvergencie je zvýšenie percento mutácie, zvýšenie počtu mutácií na jednotlivých chromozómoch alebo väčšia populácia. Zvýšené percento mutácie ale rozbíja stavebné bloky (viď [2]), zvýšenie počtu mutácií (ak budeme teraz uvažovať napr. CGP) zasa nevytvára riešenia blízko rodiča, a „poskakuje“ v stavovom priestore, pričom je problém nájsť optimálne riešenie (toto je najmä znateľné pri menšej štruktúre 2-D mriežky, kedy každá mohutná mutácia jedinca znamená značný rozvrat jeho genetického materiálu). Použitie väčšej populácie dáva pomerne slušnú záruku toho, že bude dostatočne pokrytý prehľadávaný priestor a a zvýši sa tým pravdepodobnosť nájdenia riešenia blízko globálneho optima. Problém ale je, že príliš veľká populácia dlho konverguje, zatiaľ čo menšia môže ľahšie skĺznuť do lokálneho extrému.

Okrem týchto možností existuje niekoľko variácií všeobecného evolučného algoritmu, akými sú napr. predvýber, crowding, deterministický/multimodálny crowding, alebo zdieľaná fitness funkcia (viac v [2]). Na týchto technikách sú založené aj mnohé novšie metódy, ktorých nevýhodou ale je, že

zvyčajne pracujú lepšie s bitovou reprezentáciou chromozómov ako napríklad s reprezentáciou pri genetickom programovaní.

Jednou z možností, ako udržať dostatočnú diverzitu v populácii, je zavádzanie nových indivíduí do procesu evolúcie, čo umožňuje opustiť lokálne extrémy a prehľadávať inú časť priestoru riešení. Toto sa implicitne vykonáva pri reštartovaní evolučného algoritmu – vždy iné náhodné rozmiestnenie počiatočnej generácie umožňuje rozmiestniť jedince a zvyšuje tak šancu, že sa priblížime globálnemu optimu. Avšak je značne problematické určiť, po koľkých generáciách by mal byť algoritmus reštartovaný. Malý počet generácií môže vyústiť v to, že zastavíme evolúciu napriek tomu, že nedosiahla svoj koniec, pričom sme mohli dostať lepší výsledok. Na druhej strane môžeme stráviť dlhý čas na jednom mieste, pričom sme uviazli v lokálnom extréme.

Technika ALPS (Age Layered Population Structure) bola zavedená za účelom lepšej integrácie nových jedincov. ALPS používa istý spôsob merania veku za účelom obmedzenia súťaženia rozlične starých jedincov. ALPS meria vek genetického materiálu, t.j. ako dlho je prítomný v procese evolúcie. Náhodne vygenerovaný jedinec dostáva „časové razítko“ – vek 0. Každá generácia, v ktorej je použitý genetický materiál tohto jedinca, má vek vyšší o 1. To znamená, že akýkoľvek jedinec vytvorený krížením alebo mutáciou dostáva pridelené časové razítko o 1 vyššie, ako je vek jeho (naj)staršieho rodiča. Týmto spôsobom je populácia rozvrstvená do niekoľkých časových vrstiev, pričom každá z vrstiev má dovolený určitý maximálny vek, ktorý keď daný jedinec prekročí, prechádza do vyššej vrstvy. Jedince v počiatočnej vrstve sú pravidelne nahradené novými, náhodne vygenerovanými.

Rozvrstvením populácie dosiahneme toho, že kríženie a mutácia prebiehajú iba v konkrétnych vrstvách, tj. s približne rovnako starými jedincami, čo umožňuje novým chromozómom prehľadať stavový priestor, pričom sú chránené od starších jedincov, ktoré by mohli byť dominantné a spôsobiť výpadok genetického materiálu. Dôsledkom tohto metóda zabraňuje predčasnej konvergencii a uviaznutiu v lokálnom optime.

Technika ALPS je v podstate podobná iným klasickým typom evolučných algoritmov, ale tieto rozdiely spôsobujú, že dosahuje výrazne lepšie výsledky. Pred použitím ALPS existovalo už niekoľko algoritmov, ktoré používali niektorý spôsob merania veku. Tieto systémy sa od ALPS odlišujú tým, že vek konkrétneho jedinca bol vždy meraný od jeho vzniku (tj. po krížení alebo mutácii) a dostal vždy časové razítko 0 alebo 1. Oproti tomu teda ALPS meria vek genetického materiálu, ako dlho je prítomný v procese evolúcie. Navyše, tieto metódy generujú nové jedince iba na počiatku algoritmu, teda sú limitované nájdením riešenia iba v tzv. oblasti atraktoru (je to možné si predstaviť ako skupinu bodov v istej oblasti).

Iný evolučný algoritmus oddeľuje populáciu do rozličných vrstiev a pravidelne generuje náhodne nové jedince do počiatočnej vrstvy – tzv. HFC model. Rozdiel medzi HFC a ALPS je ale ten, že HFC používa fitness na rozdeľovanie jedincov do vrstiev. Oddelenie jedincov na základe fitness sa rovnako ukázalo ako prínosné vzhľadom na to, že jedince s vysokou hodnotou fitness sú vo

vyššej vrstve a teda nemajú možnosť dominovať nad jedincami z nižších vrstiev, ktoré takto dostávajú možnosť preskúmať stavový priestor a majú viac času na nájdenie lepšieho riešenia. Boli použité rozličné variácie HFC, napr. AFC (Adaptive Hierarchical Fair Competition) alebo CHFC (Continuous Hierarchical Fair Competition), no neukázali sa byť výrazne úspešnejšími ako HFC.

4.3 Popis ALPS

ALPS sa líši od tradičných evolučných algoritmov tým, že segreguje jedince použitím nového kritéria – veku genetického materiálu jedincov za účelom kontroly použitia genetických operátorov (kríženia, mutácie). Vek daného jedinca je definovaný v ALPS tým, ako dlho sa vyskytuje tento genetický materiál v procese evolúcie. Náhodne vygenerované nové jedince majú pridelené číslo 0, takto je ich genetický materiál zavedený do populácie. Jedince, ktoré vzniknú krížením alebo mutáciou, majú vek o 1 vyšší, ako má ich najstarší rodič (pri mutácii je samozrejme iba jeden). Ak je daný jedinec skopírovaný do ďalšej generácie (napr. využitím elitizmu), je takisto jeho vek zvýšený o 1, a zostáva uchovaný do tej doby, pokým nie je použitý ako rodič. Pri elitizme je vek zvýšený napriek tomu, že jedinec nebol použitý v reprodukčnom procese, a to z toho dôvodu, že jeho genetický materiál bol použitý v danej generácii, a súťažil tak s jeho potomkami. Aj keď je jedinec použitý v reprodukčnom procese niekoľkokrát, jeho vek je zvýšený vždy iba o 1 pri kopírovaní do novej generácie, pretože vek hovorí o tom, ako dlho je genetický materiál používaný, nie ako je rozšírený.

Za účelom obmedzenia súťaženia a reprodukcie medzi jednotlivými jedincami je populácia rozdelená do niekoľkých vekových vrstiev, ktoré sú podobné ostrovčekom, kde sa jedince rozmnožujú iba v rámci ostrovčekov, nie medzi nimi. Každá táto vrstva má vekový limit, po ktorý v nej môže zostať jedinec, okrem poslednej vrstvy, ktorá obsahuje jedince rozličného veku (samozrejme vyššieho ako maximálny vek jedinca predposlednej vekovej vrstvy). Pri určovaní maximálneho veku v danej vrstve môžeme vychádzať z rozličných vekových schém, a to napr. lineárna, Fibonacciho, polynomiálna alebo exponenciálna:

	Maximálny vek vo vrstve						
Veková schéma	0	1	2	3	4	5	6
Lineárna	1	2	3	4	5	6	7
Fibonacciho	1	2	3	5	8	13	21
Polynomiálna (n^2)	1	4	9	16	25	36	49
Exponenciálna (2^n)	1	2	4	8	16	32	64

Tabuľka č. 2: Rozličné spôsoby nastavenia vekových limitov v jednotlivých vrstvách

Aby sme oddelili jednotlivé vekové vrstvy a umožnili počet jedincov v nich nejakým spôsobom manažovať, používa sa tzv. age-gap parameter. Toto číslo násobíme konštantami v tabuľke pre

zvolenú vekovú schému. Teda napr. ak zvolíme age-gap 20 a použijeme polynomiálnu schému, jednotlivé vekové vrstvy (počnúc od prvej) budú obsahovať nasledovný počet jedincov: 20, 40, 80, 180, 320... Toto umožňuje jedincom v prvej vrstve nájsť vhodnú oblasť, v ktorej by sa mohlo nachádzať globálne optimum. Teoreticky môže ALPS používať teoreticky nekonečný počet vrstiev, obmedzený sme iba pamäťou počítača.

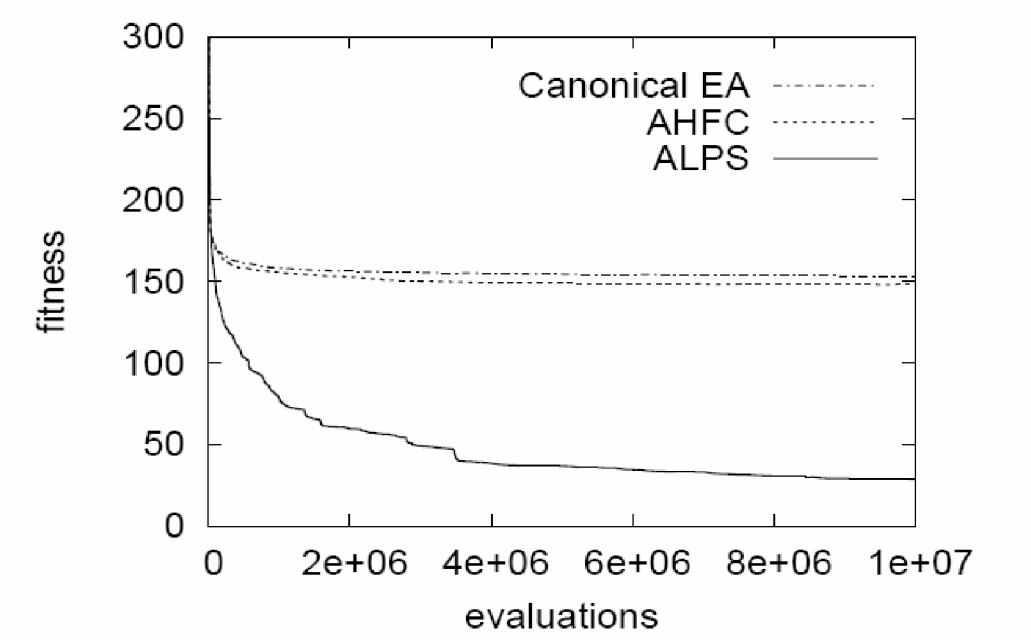
V podstate je evolúcia s využitím techniky ALPS podobná bežnému evolučnému algoritmu, avšak s niekoľkými podstatnými odchýlkami. Poprvé, jedince vstupujú do reprodukčného procesu iba s jedincami v rovnakej vekovej vrstve, alebo vrstve o 1 nižšou. To znamená, že rodičia pre vrstvu 0 sú vybraní iba z vrstvy 0, rodičia pre vrstvu 1 sú vyberaní z vrstiev 0 a 1, a tak ďalej až rodičov pre vrstvu n vyberáme z vrstiev n a n-1. Keďže vyberáme rodičov z dvoch vrstiev, takýmto spôsobom môžu potomkovia hladko postupovať v rámci vrstiev vyššie. Po druhé, jedince vo vrstve 0 sú v pravidelných intervaloch (vždy po age-gap generáciách) vždy kompletne nahradené náhodne vygenerovanými jedincami. Pre age-gap 5 je to teda v generáciách 0, 5, 10, 15,... A po tretie, jedince sú v danej vekovej vrstve iba po tú dobu, po ktorú spĺňajú vekový limit. To znamená, že ak použijeme vekovú schému lineárnu, a age-gap 10, vo vrstve 0 je aktívna iba prvých 10 generácií, potom je použitá vrstva 1, vrstva 2 začína od generácie 20 vyššie. Prvá vrstva je vždy kompletne nahradená náhodne vygenerovanými jedincami. Keď jedince nespĺňajú vekový limit, je ich fitness hodnota porovnaná s fitness hodnotou jedincov z vyššej vrstvy, a ak je lepšia aspoň ako jeden chromozóm z vyššej vrstvy, jedinec prechádzajúci z nižšej vrstvy nahradí toho najhoršieho z vrstvy vyššej.

Zvyšovanie veku jedinca po každej generácii, v ktorej je použitý ako rodič, mu umožňuje postúpiť veľmi vysoko v rámci vrstiev, až do poslednej, v ktorej sa nachádzajú najstaršie jedince, teda tie s najvyššou hodnotou fitness. Avšak jedinec ostáva v populácii iba za predpokladu, že je zároveň globálnym optimom, alebo sa nachádza veľmi blízko neho. Toto je zásadný rozdiel od techniky HFC, kde môže jedinec ostať v danej vrstve dovtedy, pokým je jeho fitness hodnota maximom v danej fitness vrstve.

4.4 Dosiahnuté výsledky

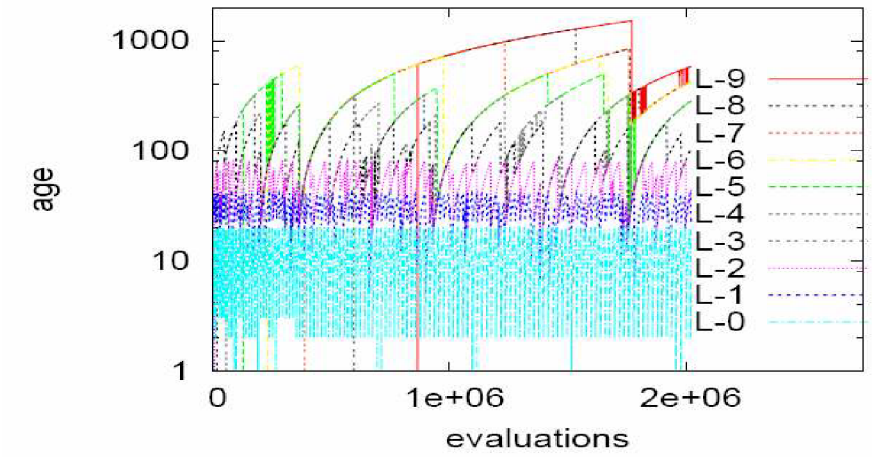
V porovnaní s inými typmi evolučných algoritmov, či už klasickým kanonickým EA, alebo algoritmom využívajúcim obmenu AHFC [9] dosahuje dobré výsledky a významne ich prekonáva (viď obrázok č. 15). Je to dosiahnuté tým, že ALPS neustále zavádza do procesu evolúcie nové náhodne generované riešenia, a týmto zvyšuje pravdepodobnosť, že sa algoritmu podarí vymaniť z lokálnych optím a nájsť globálne riešenie. Problém dominancie starších jedincov javiacich sa ako lepšie prispôsobených prostrediu (majúcich teda vyššiu hodnotu fitness), je ošetrený obmedzením možnosti pohlavnej reprodukcie v rámci jednotlivých (maximálne dvoch susedných) vekových vrstiev jedincov. Pri experimentoch bolo ďalej zistené, že všetky riešenia nájdené využitím techniky

ALPS boli lepšie v porovnaní s inými (konkrétne kanonický EA a EA s použitím techniky AHFC), čo takisto naznačuje, že globálne optimum by malo byť nájdený s využitím menšieho počtu generácií.



Obrázok č. 15: Graf znázorňujúci fitness hodnotu najlepšieho jedinca u rozl. EA [9]

Neustály prísun nových jedincov do populácie je síce podobný reštartovaniu EA, ale umožňuje reštartovať algoritmus „čiastočne“, pričom je umožnené novým jedincom evolvovať a prehľadávať stavový priestor, hľadajúc v ňom globálne optimum bez toho, aby bol ich genetický materiál potlačený riešeniami javiacimi sa ako lepšie, hoci sú tieto iba niekde blízko lokálneho extrému. Na obrázku č. 15 je vidieť postupný prechod najlepších jedincov skrz jednotlivé vekové vrstvy, čím je umožnené čiastkové preskúmanie novými jedincami odlišných oblastí atraktorov (basin of attraction), ktoré delia celkový stavový priestor.



Obrázok č. 16: Graf znázorňujúci fitness hodnoty a vek najlepších jedincov počas prvých 2 miliónov generácií algoritmu využívajúceho ALPS [9]

4.5 Hybridné formy ALPS

Napriek tomu, že ALPS poskytuje pravidelným zavádzaním nových jedincov do populácie značne lepšie výsledky oproti kanonickému EA alebo iným jeho vylepšeniam, nie vždy je táto technika schopná nájsť riešenie blízko globálneho optima. Preto sa ukázalo ako vhodné [9] kombinovať ALPS s inou technikou podporujúcou dostatočnú diverzitu v populácii, konkrétne bol použitý deterministický crowding (viď kapitola 4.1.1).

5 Návrh začlenenia ALPS do CGP

V rámci tejto kapitoly bude navrhnutý spôsob začlenenia technika ALPS do kartézskeho genetického programovania. V ďalšom semestri bude na základe návrhu implementované CGP využívajúce ALPS. Keďže na zadaných testovacích úlohách budú vykonané testy a porovnania klasického CGP a CGP využívajúceho ALPS, je určite nutné navrhnúť viacero spôsobov začlenenia techniky do kartézskeho genetického programovania.

Relatívne najväčším problémom pri návrhu sa javí problém absencie genetického operátora kríženia v CGP. Uvedené dva spôsoby návrhu sú si značne podobné a používajú iba mutáciu, s tým rozdielom, že v druhom návrhu je vždy pri generovaní novej populácie použitých viacero „simultánnych" skupín. Využitie techniky ALPS v kartézskom genetickom prináša isté problémy, ako je napríklad rovnako rýchle starnutie populácie v jednej vrstve, čo neumožňuje plynulý presun genetického materiálu medzi vrstvami, ale „skokovitý". Ďalším problémom je, že po zostarnutí jednej vrstvy nemusí byť práve aktuálny prechod z nižšej vrstvy (tá ešte nedosiahla potrebný maximálny vek), a teda takáto vrstva „idluje".

V nasledujúcich podkapitolách sú uvedené návrhy dvoch možných spôsobov začlenenia ALPS do CGP.

5.1 1. spôsob začlenenia ALPS do CGP

Prvý spôsob vychádza z použitej techniky ALPS popísanej v literatúre [9], avšak vzhľadom na absenciu kríženia budú jednotlivé subpopulácie putovať v závislosti na zvolenej vekovej schéme relatívne postupne vekovými vrstvami. Keďže kríženie jedincov v rámci dvoch susedných vekových vrstiev nie je možné, z toho dôvodu nie je umožnený ani kontinuálny presun genetického materiálu vyššie medzi jednotlivými vrstvami (vek jedinca je vek najstaršieho rodiča zvýšený o jedničku). Toto neplatí doslovne pre iné vekové schémy ako lineárna, kedy môže byť pri dosiahnutí maximálneho veku jedinca v nižšej vekovej vrstve najlepší jedinec vo vyššej vrstve ním nahradený. V poslednej vekovej vrstve budú vzájomne interagovať jedince z rozličných subpopulácií, t.j. na základe porovnania fitness hodnoty mladší jedinec buď nahradí staršieho jedinca s nižšou hodnotou fitness, alebo bude vymazaný z populácie (toto platí pre všetky vekové schémy).

Príklad: Uvažujme tri vekové vrstvy, parameter age-gap=20, polynomiálnu vekovú schému. Počet jedincov bude v jednej subpopulácii štandardne 5 (4+1). Potom priebeh niekoľkých vybraných krokov behu evolučného algoritmu ju:

V generácii č. 1 bude zaplnená prvá veková vrstva, kde budú nagenerované náhodné riešenia (čím vyššia numerická hodnota fitness, tým vhodnejší jedinec):

Veková vrstva 0
Fitness 15
Fitness 54
Fitness 53
Fitness 3
Fitness 7

Veková vrstva 1
-
-
-
-
-

Veková vrstva 2
-
-
-
-
-

Obrázok č. 17: 1. krok algoritmu využívajúceho ALPS v CGP

V generácii č. 11 sa presunie najlepší jedinec z prvej vekovej vrstvy vyššie, a subpopulácia vo vekovej vrstve 0 bude kompletne nahradená novými náhodne generovanými jedincami (posledná veková vrstva ostáva stále prázdna):

Veková vrstva 0
Fitness 13
Fitness 43
Fitness 64
Fitness 32
Fitness 23

Veková vrstva 1
Fitness 64
Fitness 54
Fitness 83
Fitness 34
Fitness 19

Veková vrstva 2
-
-
-
-
-

Obrázok č. 18: 2. krok algoritmu využívajúceho ALPS v CGP

V generácii č. 21 sa porovnajú hodnoty fitness najlepších dvoch jedincov z vekových vrstiev 0 a 1, ak bude jedinec z prvej vrstvy lepší (alebo bude mať rovnakú hodnotu fitness), nahradí jedinca vo vyššej vrstve, subpopulácia v nižšej vekovej vrstve je znovu nahradená novými riešeniami. Z vekovej vrstvy 1 zatiaľ neprechádza žiadny jedinec, kvôli použitej polynomiálnej vekovej schéme:

Veková vrstva 0
Fitness 2
Fitness 71
Fitness 56
Fitness 12
Fitness 34

Veková vrstva 1
Fitness 73
Fitness 52
Fitness 76
Fitness 34
Fitness 47

Veková vrstva 2
-
-
-
-
-

Obrázok č. 18: 3. krok algoritmu využívajúceho ALPS v CGP

V generácii č. 31 najlepší jedinec z druhej vrstvy prechádza do vyššej vekovej vrstvy 2 a sú z neho mutované nové riešenia, nasleduje porovnanie najlepších riešení vrstiev 0 a 1, prípadné presunutie sa najlepšieho jedinca z vrstvy 0 a generovanie novej subpopulácie vo vrstve 0:

Veková vrstva 0
Fitness 4
Fitness 56
Fitness 37
Fitness 95
Fitness 12

nahrádza

Veková vrstva 1
Fitness 17
Fitness 54
Fitness 83
Fitness 41
Fitness 7

Veková vrstva 2
Fitness 83
Fitness 97
Fitness 53
Fitness 42
Fitness 65

5.2 2. spôsob začlenenia ALPS do CGP

Druhý spôsob je podobný prvému, takisto s absenciou kríženia. Avšak tu nebude v prvej vekovej vrstve generovaná jediná subpopulácia, ale bude ich viacero (3-5). To znamená, že v jednotlivých vrstvách sa bude postupne evolvovať viacero subpopulácií rovnakého veku. Hoci toto je značne podobné prvému spôsobu (tu je iba simultánne spustených viacero subpopulácií), chcel by som porovnať výsledky, kedy bude najlepšia subpopulácia (resp. najlepší jedinec z danej subpopulácie danej vekovej vrstvy) z nižšej vekovej vrstvy nahrádzať subpopuláciu z vyššej vrstvy, pričom budú tieto subpopulácie porovnávané na základe fitness hodnoty najlepšieho jedinca z danej populácie (z ktorého sa generujú nové riešenia).

Tento spôsob viac zodpovedá použitiu ALPS v klasickom EA, kedy z viacerých jedincov vyberiem do vyššej vrstvy toho najlepšieho z niekoľkých.

6 Implementácia kartézskeho genetického programovania využívajúceho techniku ALPS

6.1 Výber programovacieho jazyka

Výber programovacieho jazyka je jedným z najdôležitejších rozhodnutí. Tento výber môže vo významnej miere ovplyvniť vývoj celej aplikácie a jej vlastnosti.

V zadaní projektu je formulovaná požiadavka na vytvorenie implementácie CGP využívajúceho techniky ALPS. Túto možnosť nám samozrejme poskytuje každý netriviálny vyšší programovací jazyk; avšak každý z jazykov sa vyznačuje rozličnou mierou vhodnosti pre daný projekt. Z „klasických“, či „modernejších“ programovacích jazykov je možné uvažovať napríklad o jazykoch ako je Java, C#, C++, C, prípadne iné. Objektovo orientované jazyky, kam patria aj prvé tri uvedené, zažívajú veľký rozmach, najmä kvôli možnosti urýchlenia vývoja aplikácie, modulárnemu návrhu, a to pri súčasnom zvýšení kvality a bezpečnosti vytváraných softvérových produktov (výnimky, kontrola hraníc polí v Jave, atď.). Naproti tomu, daňou za tento programátorský prepych býva práve rýchlosť aplikácie, ktorá je jedným z najdôležitejších parametrov pri evolučnom návrhu. Aj z tohto dôvodu som sa rozhodol pre jazyk C, ktorý umožňuje okrem iného aj rýchlu prácu s poľami, čo je pre tento typ aplikácie zrejme jedným z najväčších kritérií. Pri použití štandardných knižníc bude takisto možné preložiť projekt pod rôznymi operačnými systémami.

6.2 Cieľ implementácie

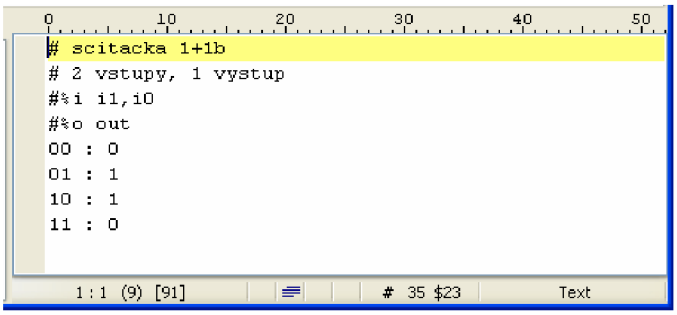
Hlavným cieľom práce je vytvoriť funkčnú implementáciu techniku ALPS v kartézskom genetickom programovaní. Nastavenie ideálnych parametrov evolúcie je pomerne náročný proces, v ktorom nie je možné nájsť jednoznačne najlepšiu alebo najhoršiu kombináciu parametrov ako je veľkosť mriežky CGP, počet mutovaných génov, veľkosť populácie a podobne. Pri implementácii a overovaní výsledkov na jednotlivých zadaných úlohách budem väčšinou vychádzať z bežne dostupných výsledkov a známych nastavení, keďže cieľom práce je porovnať výsledky klasického CGP oproti tomu využívajúcemu ALPS. Je predpoklad, že zdrojový kód môže byť v budúcnosti modifikovaný alebo rozširovaný niekým pokračujúcim v práci, takisto je zrejmé, že výsledný produkt nebude určený úplne bežnému užívateľovi PC. Z týchto dôvodov, takisto z nutnosti veľkého množstva nastavení jednotlivých parametrov, fakticky zbytočnosti grafického rozhrania (väčšie problémy

s prenositeľnosťou, nutnosť inštalovania špeciálnej knižnice; som sa rozhodol implementovať program ako konzolovú aplikáciu. Jednotlivé parametre budú nastaviteľné prostredníctvom hlavičkového súboru. Na zobrazovanie jednotlivých chromozómov bude použitý program CGPViewer od p. Ing. Vašíčka).

6.3 Postup pri implementácii

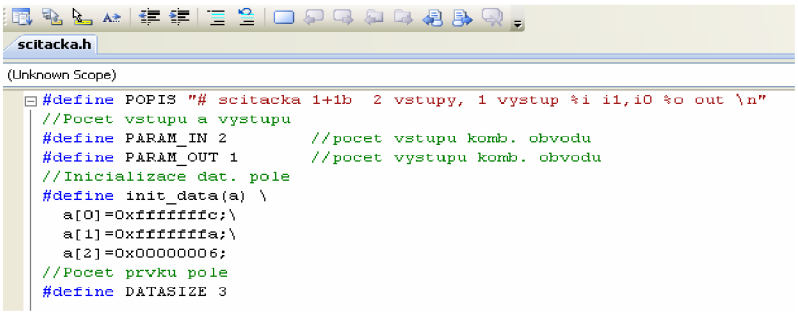
6.3.1 Zakódovanie riešeného problému

Funkcia kombinačného logického obvodu býva zadaná vo forme pravdivostnej tabuľky (na aké vstupy dostávam aké výstupy, viď obrázok č. 19):



Obrázok č. 19: Pravdivostná tabuľka jednobitovej sčítačky

Na obrázku č. 19 je znázornená pravdivostná tabuľka jednobitovej sčítačky. Pre urýchlenie vyhodnocovania fitness hodnoty daného jedinca bude použitá tzv. „paralelné“ vyhodnocovanie v tom zmysle, že vstupy jednotlivých chromozómov nebudú reprezentované bitovo, ale ako celé čísla typu integer, čo umožní (v mojom prípade) pri použití 32 – bitovej počítačovej architektúry až 32 – násobné urýchlenie výpočtu. Na zakódovanie takejto tabuľky do vhodnej reprezentácie je použitý program tab2h.exe od Ing. Vašíčka. Ten po spracovaní takého vstupné súboru vygeneruje hlavičkový súbor, ktorý vyzerá nasledovne:



Obrázok č. 20: Zakódovaná pravdivostná tabuľka jednobitovej sčítačky

Program do hlavičkového súboru nageneruje parametre ako sú počet vstupov, výstupov kombinačného obvodu, a definuje makro pre inicializáciu poľa so zakódovanými prvkami.

Toto platí pre kombinačné obvody – pre iné testovacie úlohy je v prílohe A uvedený postup definovania potrebných parametrov pre evolučný návrh.

6.3.2 Náčrt postupu implementácie

6.3.3 1. fáza implementácie

Prvým cieľom bolo vytvorenie implementácie klasického CGP, aby bolo možné s ním získané výsledky porovnať s výsledkami dosiahnutými po implementovaní vyšie uvedených dvoch návrhov začlenenia techniky ALPS do CGP. Tento program bude tvoriť kostru pre implementáciu ďalších dvoch navrhnutých spôsobov. V tejto časti návrhu boli navrhnuté základné hlavičkové súbory, ktoré aplikácia využíva, kam patrí najmä spôsob kódovania chromozómu a hlavičkový súbor, prostredníctvom ktoré budú nastavované užívateľom aplikácie jednotlivé parametre evolučného procesu. Implementácia projektu obsahuje hlavičkové súbory *cgp.h* (tu sa nastavujú parametre evolúcie), *chromozom.h* (definícia chromozómu), *rozsahy.h* (definícia štruktúry, ktorá slúži pri generovaní look-up tabuliek pre minimálnu alebo maximálnu hodnotu vstupu génu). V zdrojovom súbore *cgp.cpp* je implementovaný celý algoritmus CGP (alebo jeho dvoch modifikácií).

Hlavičkový súbor popisujúci spôsob zakódovania jedinca (*chromozom.h*):

```
/**
Struktura reprezentujuca jedinca,
konkretne riesenie.
*/
typedef struct chromozom {
      /*Vek jedinca*/
      unsigned int vek;

      /**
      Smernik na dynamicky alokovane pole.
      Tu su zakodovane jednotlive geny(vstup 1, vstup 2, funkcia hradla)
            (stlpce * riadky * (POCET_VSTUPOV_blocku +
      POCET_VYSTUPOV_bloku) + POCET_VYSTUPOV)*/
      int * krabicka;

      /**
      Smernik na dynamicky alokovane pole.
      tu budu ulozene vysledky jednotlivych blockov v chromozome, pocet je
      POCET_STLPCOV * POCET_RIADKOV*/
      int * vysledky;
} chrom;

/**Definicia smernika na chromozom*/
typedef chrom * CHROMOZOM;
```

Príklad definície hlavičkového súboru umožňujúceho nastavenie parametrov evolúcie (*cgp.h*) sa nachádza v prílohe A.

6.3.4 2. fáza implementácie

V druhej fáze implementácie bola do klasického CGP doimplementovaná funkčnosť podľa prvého spôsobu návrhu začlenenia techniky ALPS do CGP. Zostalo zachované kódovanie chromozómu, bol doplnený hlavičkový súbor *cgp.h* o definíciu konštánt počtu vekových vrstiev, použitej vekovej schémy a parametru age-gap, ktorý definuje maximálny možný vek v jednotlivých vrstvách. V tejto fáze boli doimplementované funkcie:

- `void inicializujVrstvy(int *vrstvy);`

    => slúžiaca na vytvorenie look – up tabuľky pre definovanie maximálneho možného veku jedincov v jednotlivých vekových vrstvách na základe definovaného parametru age – gap a použitej vekovej schémy

- `int krokEvolucie(CHROMOZOM * pop, int cisloVrstvy, int cisloIteracie);`

    => funkcia, ktorá oddeľuje evolučný proces pre jednotlivé vrstvy

- `int zistiPrechod(int iteracia, int * vrstvy);`

    => funkcia zisťujúca, či v danej generácii evolučného algoritmu nie je potrebné vykonať prechod do vyššej vrstvy z dôvodu zostarnutia genetického materiálu

- `int porovnajAvymen(CHROMOZOM * spodnaV, CHROMOZOM * hornaV, int cisloSpodnejVrstvy, int iter);`

    => táto funkcia porovná hodnotu najlepších jedincov, z ktorých sú generované nové riešenia medzi dvomi susednými vrstvami, v prípade lepšej hodnoty fitness najlepšieho jedinca v spodnej vrstve tento nahradí najlepšieho jedinca vo vyššej vrstve

- `void ohodnotPopulaciu(CHROMOZOM * pop, int cisloVrstvy);`

    => funkcia ohodnocuje novo generovanú populáciu za účelom získania potrebných informácií (index najlepšieho jedinca, maximálna hodnota fitness v danej vrstve atď.)

6.3.5 3. fáza implementácie

V tretej fáze bola doimplementovaná funkčnosť algoritmu podľa druhého spôsobu návrhu začlenenia techniky ALPS do CGP. V tejto fáze prišlo k jedinej zmene v definícii hlavičkového súboru *cgp.h*, kde pribudla definícia počtu subpopulácií v jednotlivých vrstvách. Ďalej bolo potrebné modifikovať vyššie uvedené funkcie a logiku algoritmu vzhľadom na pribudnutie ďalšieho rozmeru (počtu subpopulácií).

6.3.6 Ďalšie informácie

Keďže testovanie CGP využívajúceho techniku ALPS bolo vykonané nielen na čisto kombinačnom obvode ako je napr. násobička, ale napríklad i na probléme „násobenia čísiel konštantou“ (kde je možné používať iba bitové operácie posunu, sčítanie a odčítanie), bolo potrebné mierne modifikovať hlavičkový súbor definujúci parametre počtu vstupov , výstupov a iné, v prílohe A je možné nájsť návod na kroky potrebné pre spustenie evolučného procesu kombinačných logických obvodov a iné.
Celá implementácia CGP ako i mnou navrhnutých 2 spôsobov začlenenia techiky ALPS do CGP bola vytvorená v jazyku C, bez použitia špeciálnych knižníc, ktoré by znemožňovali prenositeľnosť implementácie medzi rôznymi OS. Preložiteľnosť programu bola testovaný na systémoch Windows XP a Linux (distribúcia Gentoo, jadro 2.6.12.6). Preklad bol vykonaný Microsoft Visual Studiom a MinGW (Windows XP) a prekladačom „g++“ (Linux) Jediným väčším obmedzením je použitie výhradne 32 – bitovej počítačovej architektúry, je možné odstrániť niekoľkými jednoduchými úpravami zdrojového kódu (jedná sa tu najmä o výpočet fitness funkcie, kde sa ráta iba s 32 – bitovými číslami).

7 Experimentálne výsledky

Pri nastavovaní parametrov evolúcie som nerobil špeciálne testy určujúce vhodnosť nastavenia veľkosti mriežky CGP, počtu mutovaných génov, maximálneho počtu generácií pre daný problém a iné. Zväčša som vychádzal z parametrov, ktoré boli dostupné pre daný typ problému (napríklad pri násobení čísiel konštantou som použil nastavenia z [12]). Bolo tak konané z dôvodu možnosti porovnania mnou získaných výsledkov v rámci klasického CGP (ktoré ako bude ukázané sú naozaj porovnateľné so štatistikami uvedenými v literatúre), a ďalej aby tak bolo možné ukázať vplyv použitia techniky ALPS v CGP. Dôsledok jej použitia je znázornený v jednotlivých podkapitolách vo forme štatistických tabuliek, ako i graficky. V grafoch sú znázornené priemerné hodnoty fitness najlepšieho jedinca získané zo všetkých behov evolúcie.

Avšak musel som revidovať (najmä pri testovaní druhého spôsobu začlenenia ALPS do CGP) 2 parametre evolučného procesu, a to maximálny počet generácií a takisto počet behov. Bolo to spôsobené najmä veľkou časovou náročnosťou 2. a 3. časti implementácie, kedy sa vlastne časom znásoboval počet populácií (ako sa postupom času stávali aktívne jednotlivé vrstvy); testy boli vykonávané na 50 – tich behoch evolúcie. Na experimentovanie bol použitý jednojadrový 32 - bitový procesor Intel Centrino s maximálnou frekvenciou 1,73 GHz. V tabuľke č. 3 sú uvedené priemerné časy evolúcie pre 50 behov pre jednotlivé experimenty.

Násobička 3x3b	
Použitá technika a parametre	*Priemerný čas pre 50 behov evolúcie*
Klasické CGP, limit 1M generácií	0:11:40
Klasické CGP, limit 2M generácií	0:23:20
CGP s využitím ALPS (1. spôsob), lineárna veková schéma, age-gap 110000, 2M generácií	1:02:30
CGP s využitím ALPS (2. spôsob), lineárna veková schéma, age-gap 100000, 1M generácií	1:35:50
Multiplierless Multiple Constant Multiplication	
Použitá technika a parametre	*Priemerný čas pre 50 behov evolúcie*
Klasické CGP, limit 1M generácií	0:06:40
Klasické CGP, limit 3M generácií	0:19:10
CGP s využitím ALPS (1. spôsob), lineárna veková schéma, age-gap 70000, 3M generácií	0:55:50
CGP s využitím ALPS (2. spôsob), lineárna veková schéma, age-gap 100000, 1M generácií	0:58:20

Tabuľka č. 3: Priemerné časy pre evolúciu pri jednotlivých nastaveniach (50 behov)

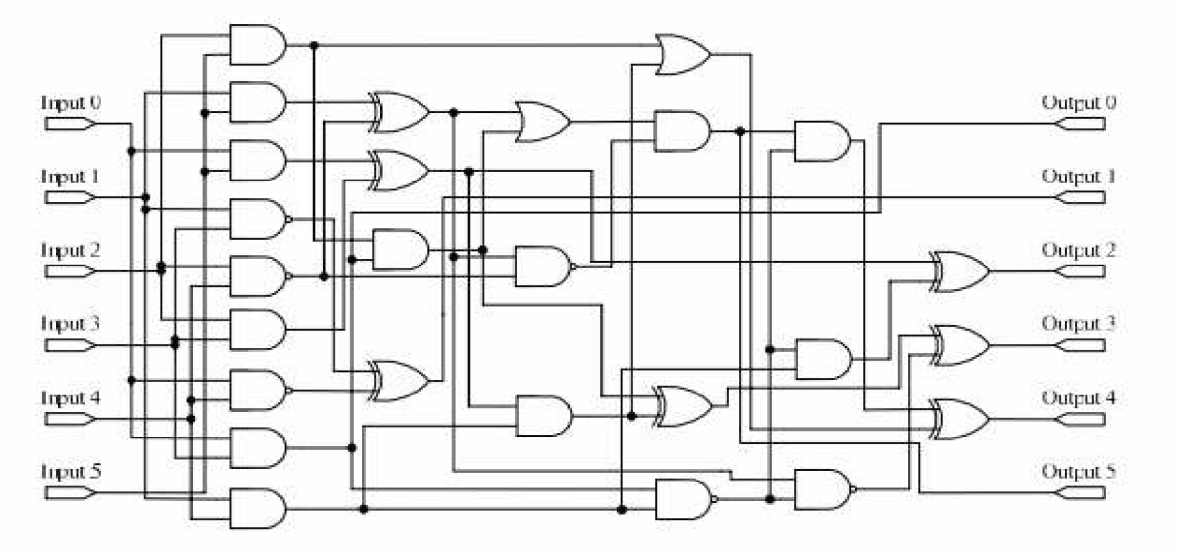
Druhý spôsob vzhľadom na pribudnutie nových subpopulácií do jednotlivých vrstiev bol časovo ešte náročnejší, bol preto znížený maximálny počet generácií.

7.1 Experiment 1 – evolučný návrh násobičky 3x3 bity

V ďalších podkapitolách bude popísaný evolučný návrh násobičky 3x3 bity; v prvej podkapitole budú porovnané mnou získané výsledky pri použití klasického CGP so známym riešením od autorov [11]. V ďalších dvoch podkapitolách nasleduje popis a výsledky experimentov za použitia mnou navrhnutých spôsobov začlenenia techniky ALPS do CGP (pri rozličných použitých vekových schémach). V závere kapitoly sú uvedené tabuľky a grafy zhodnocujúce dosiahnuté výsledky.

7.1.1 Známe riešenie

Uvedené známe riešenie bolo opísané v článku [11], pri použití metódy SAMRC (technika umožňujúca adaptibilnú zmenu veľkosti mutácie počas evolúcie). Bola použitá štruktúra veľkosti 6x9, pričom bol vyevolvovaný jedinec s 26 hradlami a oneskorením 6. Princíp algoritmu bol podobný ako v klasickom CGP, čiže najprv bolo nájdené riešenie s maximálnou hodnotou fitness, a ďalej bol znižovaný počet použitých hradiel (tento spôsob som použil aj ja vo všetkých spôsoboch implementácie CGP). Výsledok je na obr. č. 21:



Obrázok č. 21: Najlepšie nájdené riešenie 3 – bitovej sčítačky podľa [11]

Ako je vidieť, bola použitá maximálna hodnota parametru l-back. Ďalšie informácie o evolučnom návrhu násobičky (v tomto prípade 3x4 bity) je možné získať v [7]. Tam sa autori zaoberali aj nastavením parametru l-back a jeho vplyvu na úspešnosť evolučného návrhu násobičiek. Je zjavné, že najväčšiu úspešnosť má návrh násobičky pri povolení maximálneho parametru l-back (napr. pri použití l-back = 1 bola úspešnosť návrhu nulová, a to ako v prípade násobičky 3x4 bity, tak aj v prípade 3x3 bity).

Na základe takto získaných informácií som sa rozhodol nastaviť veľkosť mriežky CGP na 7x6, s maximálnym počtom generácií 2 milióny. Pri návrhu násobičky boli použité iba funkcie: WIRE (prepojka), AND, OR a XOR. Ďalšie podrobné informácie o jednotlivých nastaveniach sú uvedené pri konkrétnych experimentoch. Konštantné nastavenia pre evolučný návrh násobičky 3x3 bity:

```
#define VELKOST_POPULACIE 5          /*pocet jedincov populacie*/
#define POCET_MUT_GENOV 3            /*max pocet genu, ktory MOZE zmutovat*/

#define POCET_STLPCOV 7              /*pocet stlpcov*/
#define POCET_RIADKOV 6              /*pocet riadkov*/

#define POCET_VSTUPOV_BLOKU 2
#define POCET_VYSTUPOV_BLOKU 1

#define L_BACK 7                     /*parameter l-back*/

#define POCET_FUNKCII 4              /*počet pouzitych funkcii*/
```

7.1.2 Experiment 1A

Za použitia klasického CGP na evolučný návrh násobičky 3x3 bity boli vykonané 2 experimenty, líšiace sa iba maximálnym počtom generácií. Bolo tak vykonané z toho dôvodu, že kým pri prvom spôsobe implementácie ALPS v CGP bol použitý maximálny počet generácií 2 milióny, v druhom spôsobe bol vzhľadom na výrazné zvýšenie výpočtovej náročnosti pri použití druhého spôsobu začlenenia techniky ALPS do CGP použitý limit 1 milión. Za použitia týchto dvoch limitov bolo vykonané porovnanie klasického ALPS s jedným a druhým spôsobom začlenenia ALPS.

7.1.2.1 Experiment 1Ai

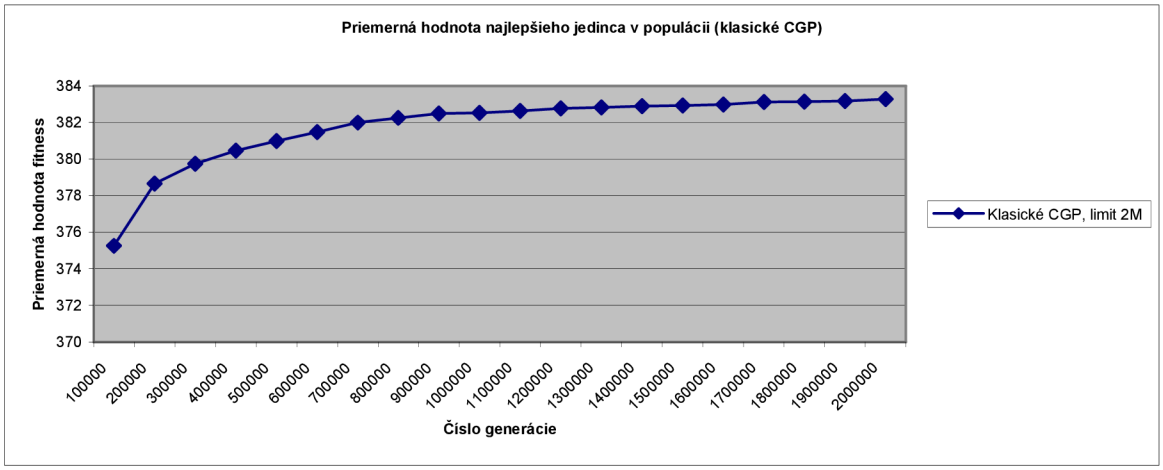
V tomto experimente bol použitý maximálny limit počtu generácií rovných 2 milióny. Ostatné nastavenia parametrov evolúcie:

```
#define POCET_GENERACII 2000000     /*max. pocet generacii evolucie*/
#define POCET_BEHOV_EVO 50          /*max. pocet behov evolucie*/
```

Priemerná úspešnosť nájdenia násobičky bola v tomto prípade 50%, čo približne zodpovedá výsledkom uvádzaným v literatúre (41% - 58%) [11] Štatistické výsledky a graf priemernej hodnoty fitness najlepšieho jedinca počas evolúcie sú uvedené v tabuľke č. 3 a na obrázku č. 22.

Priemerný počet generácií potrebných na nájdenie vhodného riešenia	827 043,19
Priemerný počet použitých hradiel v riešení	29,04
Minimálny počet použitých hradiel v riešení	26
Maximálny počet použitých hradiel v riešení	32

Tabuľka č. 4: Získané štatistické výsledky



Obrázok č. 22: Priemerná hodnota fitness najlepšieho jedinca počas evolúcie

7.1.2.2 Experiment 1Aii

V tomto experimente bol použitý maximálny limit počtu generácií rovných 1 milión. Bolo to z toho dôvodu, aby bolo možné vykonať vhodné porovnanie do stanoveného počtu generácií s druhým spôsobom použitia techniky ALPS v CGP, kde musela byť maximálna horná hranica znížená kvôli vysokej časovej náročnosti riešenia.

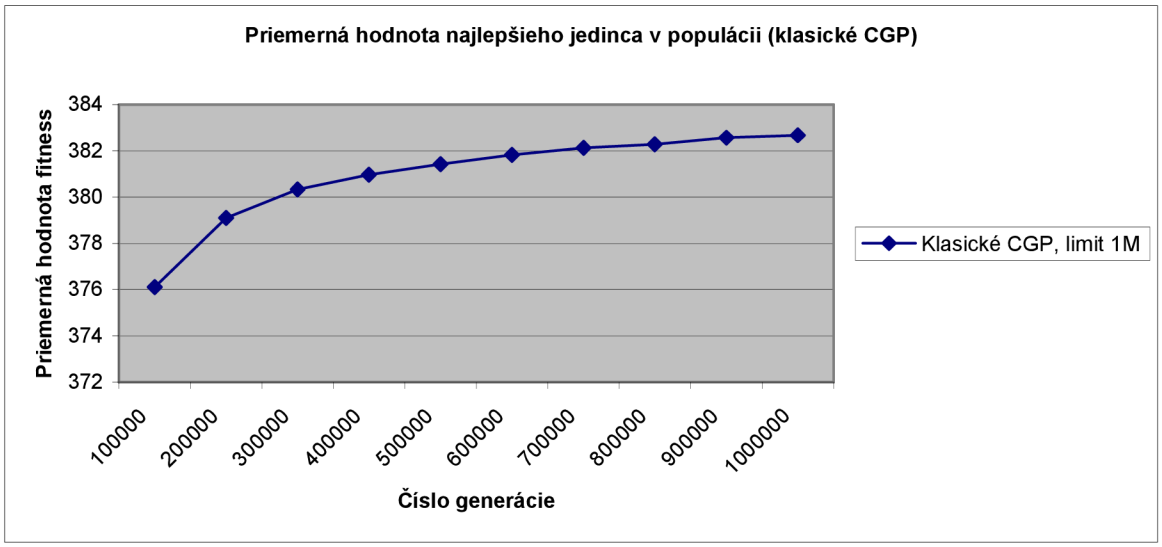
Ostatné nastavenia parametrov evolúcie:

```
#define POCET_GENERACII 1000000     /*max. pocet generacii evolucie*/
#define POCET_BEHOV_EVO 50          /*max. pocet behov evolucie*/
```

Priemerná úspešnosť nájdenia násobičky bola v tomto prípade 24%, teda približne polovičná úspešnosť evolúcie oproti experimentu 1Ai. Štatistické výsledky a graf priemernej hodnoty fitness najlepšieho jedinca počas evolúcie sú uvedené v tabuľke č. 4 a na obrázku č. 23.

Priemerný počet generácií potrebných na nájdenie vhodného riešenia	616 105,69
Priemerný počet použitých hradiel v riešení	28,50
Minimálny počet použitých hradiel v riešení	26
Maximálny počet použitých hradiel v riešení	32

Tabuľka č. 5: Získané štatistické výsledky



Obrázok č. 23: Priemerná hodnota fitness najlepšieho jedinca počas evolúcie

Zatiaľčo minimálny a maximálny počet použitých hradiel ostáva v obidvoch prípadoch (experimenty 1Ai a 1Aii) rovnaký, pri zníženom počte generácií na jeden milión dosahujeme značné zlepšenie priemerného počtu generácií potrebných pre nájdenie vhodného riešenia a takisto sa znížil priemerný počet použitých hradiel v riešení. Avšak toto je za cenu zníženia úspešnosti algoritmu až

o polovicu – teda zrejme dostáva možnosť „vyniknúť“ práve tá lepšia polovica riešení, ktoré boli nájdené za kratší čas a mali i napriek zníženému počtu generácií možnosť v priemere znížiť počet použitých hradiel (ďalšie navýšenie počtu generácií väčšinou nemáva už veľký vplyv na zníženie počtu hradiel, po čase väčšinou ich počet stagnuje a ďalej sa neznižuje).

7.1.3 Experiment 1B

V tejto podkapitole budú zobrazené výsledky experimentov za použitia prvého spôsobu začlenenia techniky ALPS do CGP. Všetky nastavenia ostávajú rovnaké ako v prípade použitia klasického CGP, ďalej bol dodefinovaný počet vekových vrstiev, ktorý ostáva pri všetkých testoch použitia rozličných vekových schém (lineárna, fibonacciho, polynomiálny a exponenciálna) rovnaký, avšak bol menený parameter age–gap, vždy v závislosti na použitej vekovej schéme. V jednotlivých podkapitolách budú uvedené dosiahnuté štatistické výsledky a grafy priemernej hodnota fitness najlepšieho jedinca v celej populácii riešení v porovnaní s klasickým CGP.

Použité parametre:

```
#define POCET_GENERACII 2000000     /*max. pocet generacii evolucie*/
#define POCET_BEHOV_EVO 50          /*max. pocet behov evolucie*/
```

7.1.3.1 Experiment 1Bi

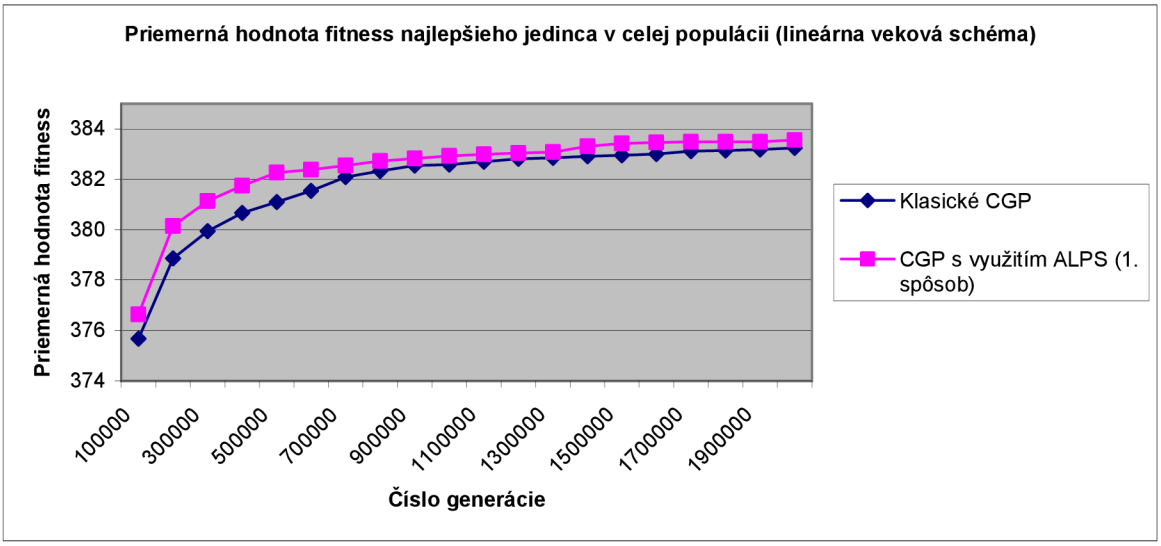
V tomto experimente bola použitá lineárna veková schéma, ktorá viedla k úspešnosti nájdenia vhodné riešenia až 64%, čo je oproti klasickému CGP pomerne slušné zlepšenie (podobné, o niečo horšie výsledky dosahovali aj iné metódy usilujúce sa vylepšiť klasické CGP, napr. spomínaný SAMRC). Takisto je grafe (obr. č. 24) možné pozorovať zlepšenie priemernej fitness hodnoty najlepšieho jedinca v populácii. Na druhej strane zo štatistík je vidno, že sa výraznejšie zvýšil priemerný počet generácií potrebných na nájdenie vhodného riešenia, ďalej maximálny počet použitých hradiel. Toto je zrejme spôsobené zavádzaním nového genetického materiálu, čo umožňuje vymaniť sa z lokálnych extrémov a zvýšiť úspešnosť evolúcie; na druhej strane tie riešenia, ktoré sú objavené tesne pred stanoveným limitom počtu generácií zvyšujú ich priemerný počet a majú taktiež menej času na zníženie počtu použitých hradiel. Napriek tomu bolo dosiahnuté zníženie priemerného počtu použitých hradiel.

Použité parametre:

```
#define AGE_SCHEME 1                /*linearna vekova schema*/
#define AGE_GAP 100000
```

Priemerný počet generácií potrebných na nájdenie vhodného riešenia	918 205.38
Priemerný počet použitých hradiel v riešení	28.75
Minimálny počet použitých hradiel v riešení	26
Maximálny počet použitých hradiel v riešení	34

Tabuľka č. 6: Získané štatistické výsledky



Obrázok č. 24: Priemerná hodnota fitness najlepších jedincov počas evolúcie

7.1.3.2 Experiment 1Bii

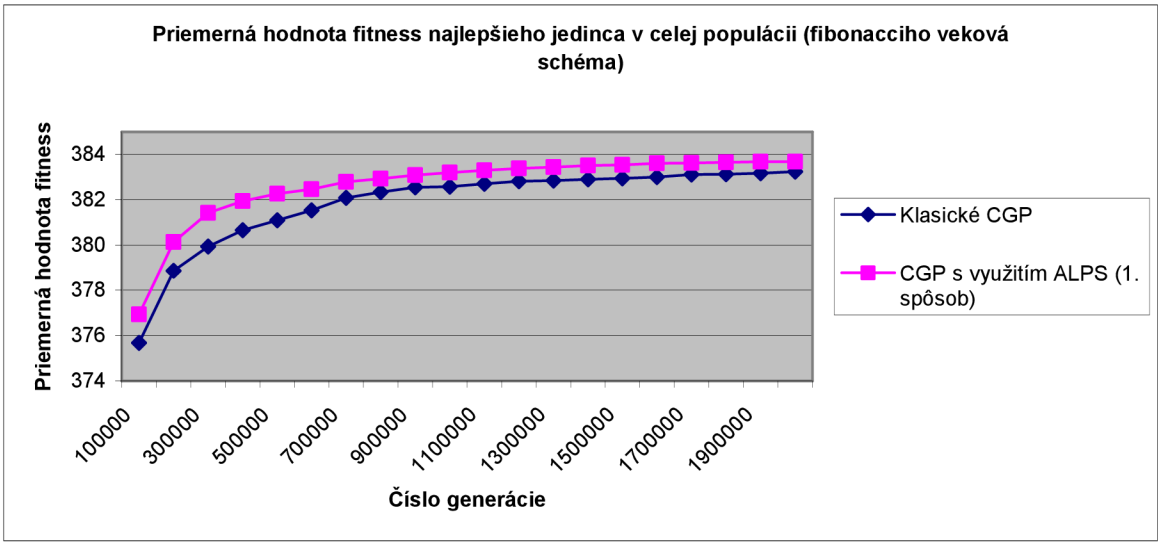
V tomto experimente bola použitá fibonacciho veková schéma, ktorá viedla k úspešnosti nájdenia vhodné riešenia až 68%, čo je rovnako významné zlepšenie, o niečo viac ako v lineárnej schéme. Takisto je grafe (obr. č. 25) možné pozorovať zlepšenie priemernej fitness hodnoty najlepšieho jedinca v populácii. Je vidno, že sa pri tejto použitej vekovej schéma zároveň zvýšil oproti klasickému CGP priemerný počet generácií potrebných na nájdenie vyhovujúceho riešenia, takisto stúpol o 2 maximálny počet použitých logických hradiel (zrejme kvôli neskoršie objaveným riešeniam, rovnako, ako u lineárnej schémy), ale znížil sa ich priemerný počet. Vyššia úspešnosť je zrejme možná pripísať tomu, že oproti lineárnej schéme sú tu väčšie rozostupy medzi jednotlivými vrstvami, a tak je nahradené dlhšie stagnujúce riešenie.

Použité parametre:

```
#define AGE_SCHEME 2                    /*fibonacciho vekova schema*/
#define AGE_GAP 100000
```

Priemerný počet generácií potrebných na nájdenie vhodného riešenia	927 803.27
Priemerný počet použitých hradiel v riešení	28.50
Minimálny počet použitých hradiel v riešení	26
Maximálny počet použitých hradiel v riešení	34

Tabuľka č. 7: Získané štatistické výsledky



Obrázok č. 25: Priemerná hodnota fitness najlepších jedincov počas evolúcie

7.1.3.3 Experiment 1Biii

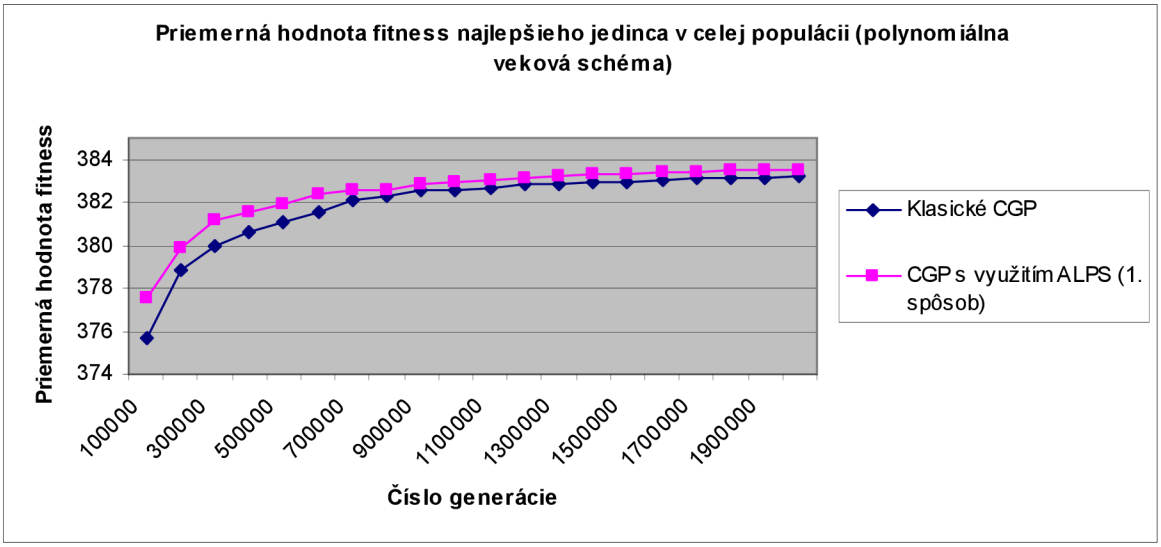
V tomto experimente bola použitá polynomiálna veková schéma, ktorá viedla k úspešnosti nájdenia vhodné riešenia 62%, čo je výsledek približne zrovnateľný s doposiaľ získanými z predchádzajúcich dvoch vekových schém. Oproti prechádzajúcej fibonacciho vekovej schéma sa znížil priemerný počet generácií potrebných na nájdenie vhodného jedinca, stále však ostáva o niečo vyšší ako pri použití klasického CGP. Na druhej strane je tu takisto oproti prechádzajúcim použitým vekovým schémam zvýšený priemerný počet použitých hradiel, čo zrejme takisto súvisí s častejším zavádzaním nového genetického materiálu.

Použité parametre:

```
#define AGE_SCHEME 3                    /*polynomialna vekova schema*/
#define AGE_GAP 50000
```

Priemerný počet generácií potrebných na nájdenie vhodného riešenia	897 214.29
Priemerný počet použitých hradiel v riešení	29.06
Minimálny počet použitých hradiel v riešení	26
Maximálny počet použitých hradiel v riešení	34

Tabuľka č. 8: Získané štatistické výsledky



Obrázok č. 26: Priemerná hodnota fitness najlepších jedincov počas evolúcie

7.1.3.4 Experiment 1Biv

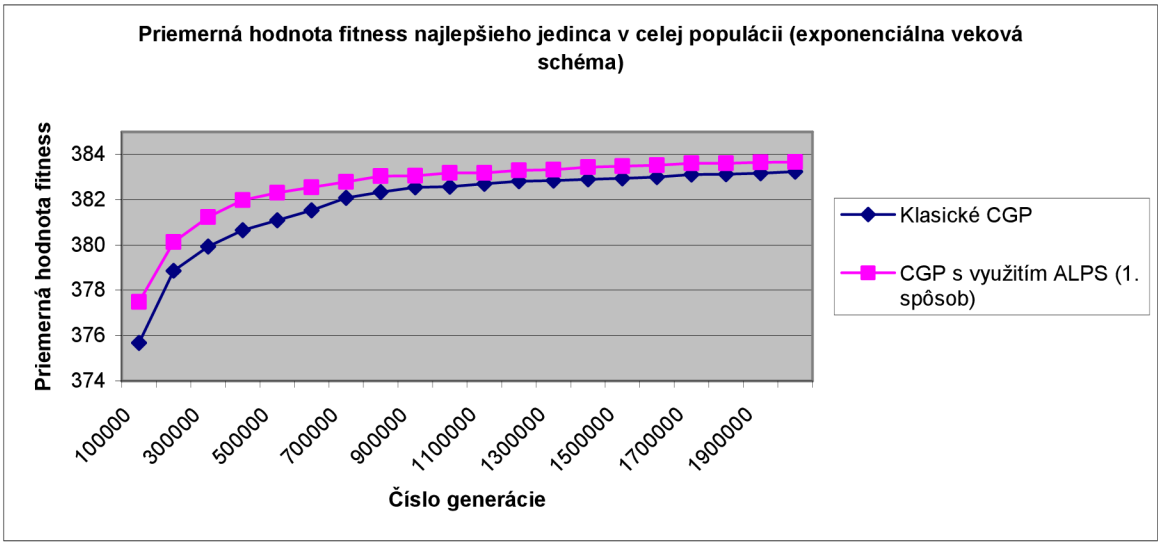
V tomto experimente bola použitá exponenciálna veková schéma, ktorá viedla k úspešnosti nájdenia vhodného riešenia až 70%. Avšak oproti ostatným vekovým schémam je tu možné pozorovať najvýznamnešie zvýšenie priemerného počtu generácií potrebných na nájdenie vhodného riešenia a zníženie priemerného počtu použitých hradiel. Toto je možné zrejme pripísať malému parametru age-gap, ktorý umožňuje veľmi intenzívne prehľadávanie stavového priestoru možných riešení, zatiaľčo na druhej strane má jedinec možnosť vo vyšších vekových vrstvách, medzi ktorými je značne väčší rozostup dokonvergovať ku globálnemu extrému.

Použité parametre:

```
#define AGE_SCHEME 4                    /*exponencialna vekova schema*/
#define AGE_GAP 50000
```

Priemerný počet generácií potrebných na nájdenie vhodného riešenia	938 545.62
Priemerný počet použitých hradiel v riešení	28.74
Minimálny počet použitých hradiel v riešení	26
Maximálny počet použitých hradiel v riešení	34

Tabuľka č. 9: Získané štatistické výsledky

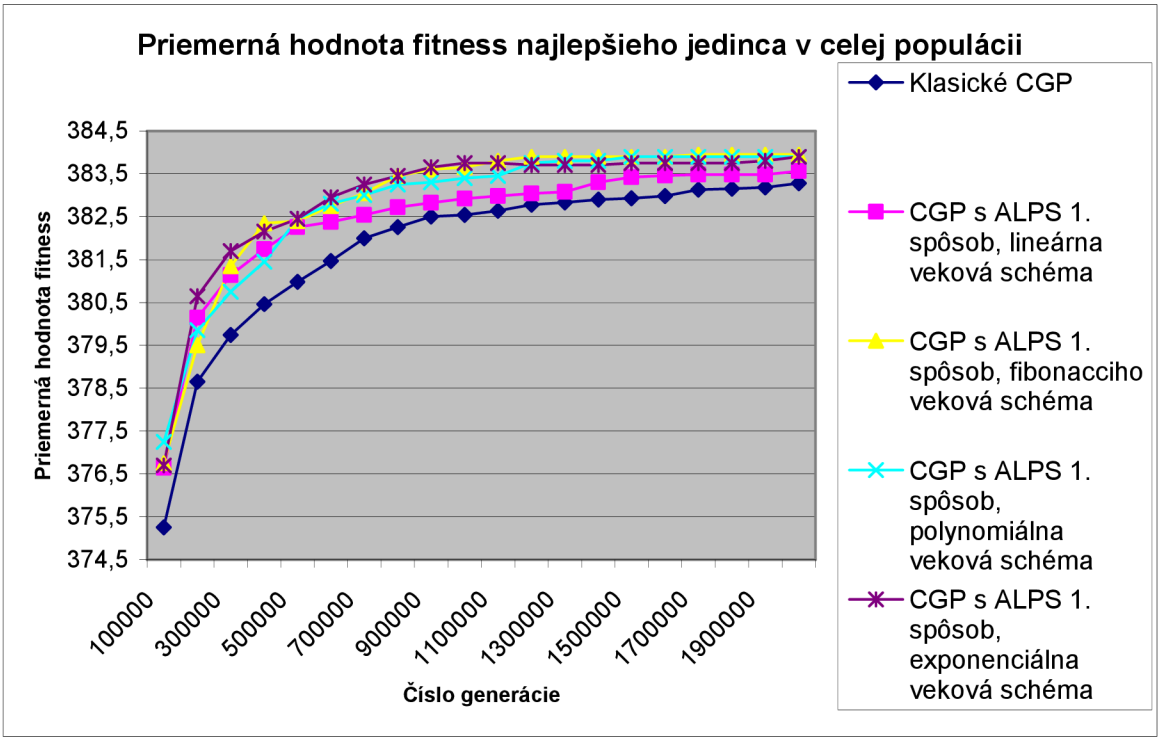


Obrázok č. 27: Priemerná hodnota fitness najlepších jedincov počas evolúcie

7.1.3.5 Zosumarizovanie výsledkov

	Priemerný počet generácií potrebných na nájdenie vhodného riešenia	Priemerný počet použitých hradiel v riešení	Minimálny počet použitých hradiel v riešení	Maximálny počet použitých hradiel v riešení
Klasické CGP	**827 043.19**	29.04	**26**	**32**
1. varianta začlenenia ALPS (lineárna vek. schéma)	918 205.38	28.75	**26**	34
1. varianta začlenenia ALPS (fibonacciho vek. schéma)	927 803.27	28.50	**26**	34
1. varianta začlenenia ALPS (polynomiálna vek. schéma)	897 214.29	29.06	**26**	34
1. varianta začlenenia ALPS (exponenciálna vek. schéma)	938 545.62	**28.74**	**26**	33

Tabuľka č. 10: Zosumarizované štatistické výsledky



Obrázok č. 28: Priemerná hodnota fitness najlepších jedincov počas evolúcie

7.1.3.6 Záver

Ako je vidieť, najlepšie výsledky pri evolučnom návrhu násobičky za použitia prvého spôsobu využitia techniky ALPS dosiahla exponenciálna veková schéma (úspešnosť, priemerný aj minimálny počet hradiel). Napriek tomu nemožno tvrdiť, že by použitie niektorej vekovej schémy malo v konečnom dôsledku (za využitia limitu 2 miliónov generácií) významný vplyv na výsledky oproti ostatným, čo je možné vidieť aj v sumarizačnom grafe (obr. č. 28).

7.1.4 Experiment 1C

V tejto podkapitole budú zobrazené výsledky experimentov za použitia druhého spôsobu začlenenia techniky ALPS do CGP. Všetky nastavenia ostávajú rovnaké ako v prípade použitia klasického CGP, ďalej bol ako v prípade experimentu 1B dodefinovaný počet vekových vrstiev, a v tomto prípade i počet subpopulácií nachádzajúcich sa v jednotlivých vekových vrstvách. Štruktúra kapitoly je obdobná ako aj u 1B, budú uvedené štatistické výsledky a grafy priemernej hodnoty fitness najlepšieho jedinca v celej populácii riešení. Toto bude porovnávané voči výsledkom získaným z experimentu 1Aii, pretože vzhľadom na pribudnutie ďalšieho rozmeru (počtu subpopulácií) prišlo k ďalšiemu zpomaleniu procesu evolúcie, preto bol maximálny počet generácií znížený na 1 milión. Počet behov evolúcie ostáva zachovaný ako v predošlom experimente 1B (50 behov), čo by malo poskytnúť dostatočne relevantné štatistické výsledky.

Použité parametre:

```
#define POCET_GENERACII 1000000      /*max. pocet generacii evolucie*/
#define POCET_BEHOV_EVO 50           /*max. pocet behov evolucie*/
#define POCET_SUBPOPULACII 3         /*pocet subpopulacii*/
```

7.1.4.1 Experiment 1Ci

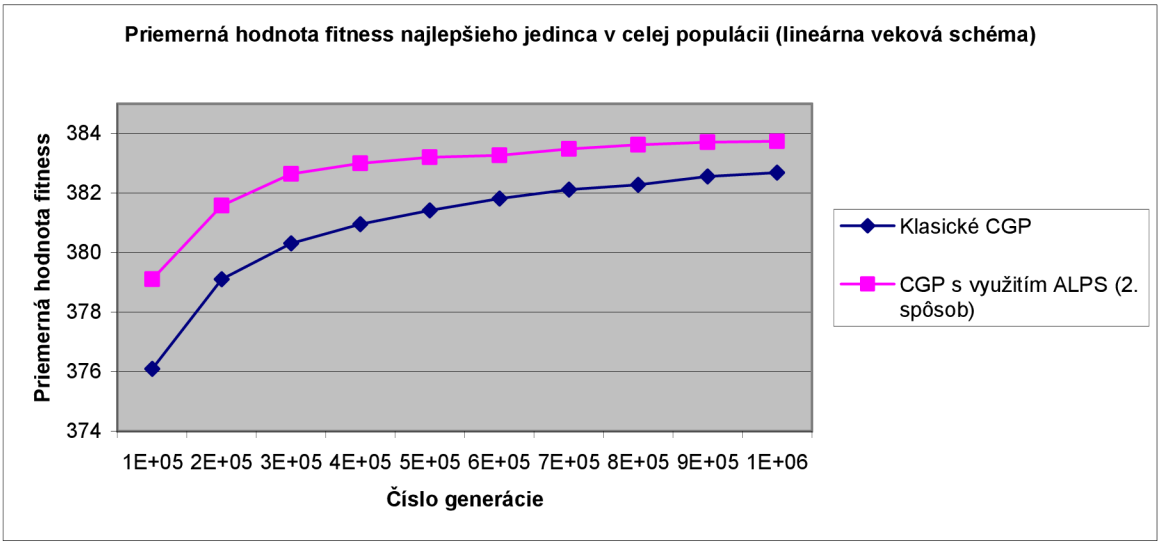
Pri tomto experimente bola použitá lineárna veková schéma, ktorá mala úspešnosť nájdenia vhodného riešenia až 74%; takisto prišlo k zníženiu priemerného počtu použitých hradiel a ich maximálneho počtu. Toto oproti klasickému CGP, ktorý pri danom limite 1 milión generácií malo úspešnosť iba 24% možno považovať za veľmi významné zlepšenie, samozrejme si treba uvedomiť, že sa tu simultánne vedľa seba vyvíjali 3 riešenia v každej aktívnej vrstve, a z nich bolo vybrané najvhodnejšie. Oproti klasickému CGP je možné v grafe pozorovať, že pri takomto spôsobe bol intenzívne prehľadávaný stavový priestor možných riešení a relatívne skoro prichádzalo k nájdeniu vhodného riešenia. Takisto prišlo k zníženiu priemerného počtu použitých hradiel.

Použité parametre:

```
#define AGE_SCHEME 1                 /*linearna vekova schema*/
#define AGE_GAP 100000
```

Priemerný počet generácií potrebných na nájdenie vhodného riešenia	501 473.73
Priemerný počet použitých hradiel v riešení	28.08
Minimálny počet použitých hradiel v riešení	26
Maximálny počet použitých hradiel v riešení	31

Tabuľka č. 11: Získané štatistické výsledky



Obrázok č. 29: Priemerná hodnota fitness najlepších jedincov počas evolúcie

7.1.4.2 Experiment 1Cii

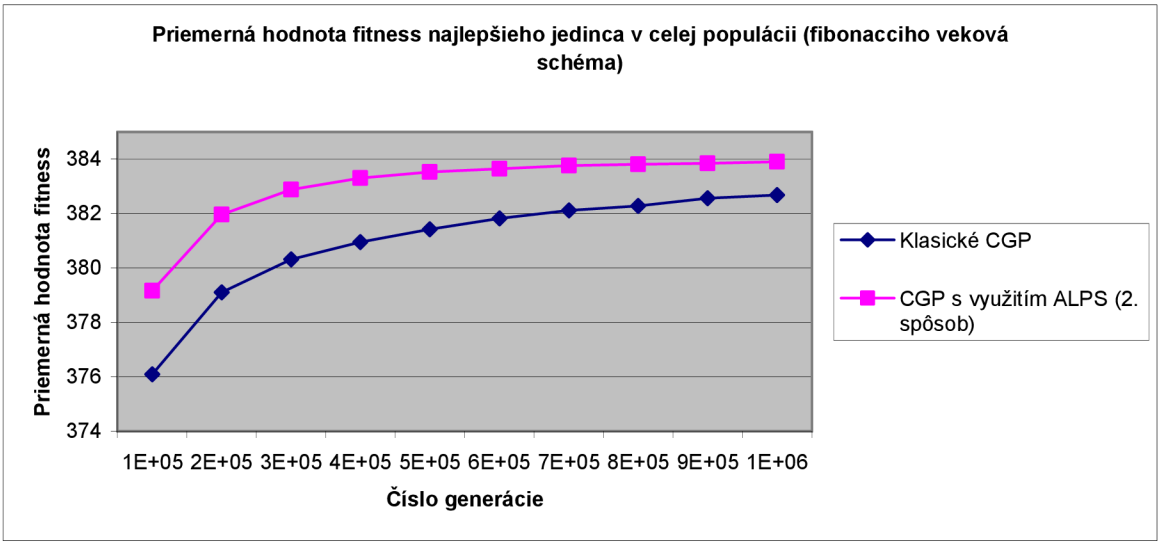
Za použitia fibonacciho vekovej schémy bola úspešnosť nájdenia vhodného riešenia až 90%. Súčasne sa znížil priemerný počet generácií potrebných na nájdenie vhodného riešenia a prišlo k miernemu zvýšeniu priemerného počtu použitých hradiel a takisto ich maximálneho počtu. Zníženie priemerného počtu generácií zrejme súvisí so znížením parametru age-gap, ktorý umožňuje rýchlejšie vyhľadávanie vhodných riešení.

Použité parametre:

```
#define AGE_SCHEME 2                    /*fibonacciho vekova schema*/
#define AGE_GAP 90000
```

Priemerný počet generácií potrebných na nájdenie vhodného riešenia	443 296.91
Priemerný počet použitých hradiel v riešení	28.22
Minimálny počet použitých hradiel v riešení	26
Maximálny počet použitých hradiel v riešení	34

Tabuľka č. 12: Získané štatistické výsledky



Obrázok č. 30: Priemerná hodnota fitness najlepších jedincov počas evolúcie

7.1.4.3 Experiment 1Ciii

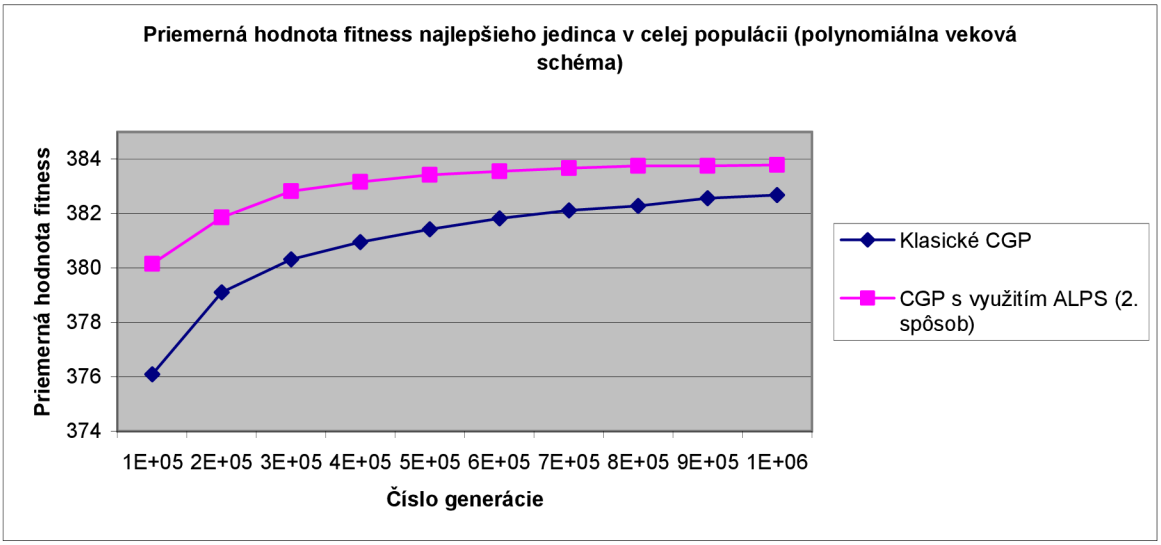
Polynomiálna veková schéma viedla k o niečo horším výsledkom (úspešnosť 82%) ako prechádzajúca fibonacciho, ale znížil sa priem. počet generácií potrebných na nájdenie vhodné riešenia (zrejme to znovu súvisí s opätovným znížením parametru age-gap). Ostatné sledované parametre ostávajú približne rovnaké ako u ostatných vekových schém.

Použité parametre:

```
#define AGE_SCHEME 3                    /*polynomialna vekova schema*/
#define AGE_GAP 50000
```

Priemerný počet generácií potrebných na nájdenie vhodného riešenia	429 388.46
Priemerný počet použitých hradiel v riešení	28.31
Minimálny počet použitých hradiel v riešení	26
Maximálny počet použitých hradiel v riešení	32

Tabuľka č. 13: Získané štatistické výsledky



Obrázok č. 31: Priemerná hodnota fitness najlepších jedincov počas evolúcie

7.1.4.4 Experiment 1Civ

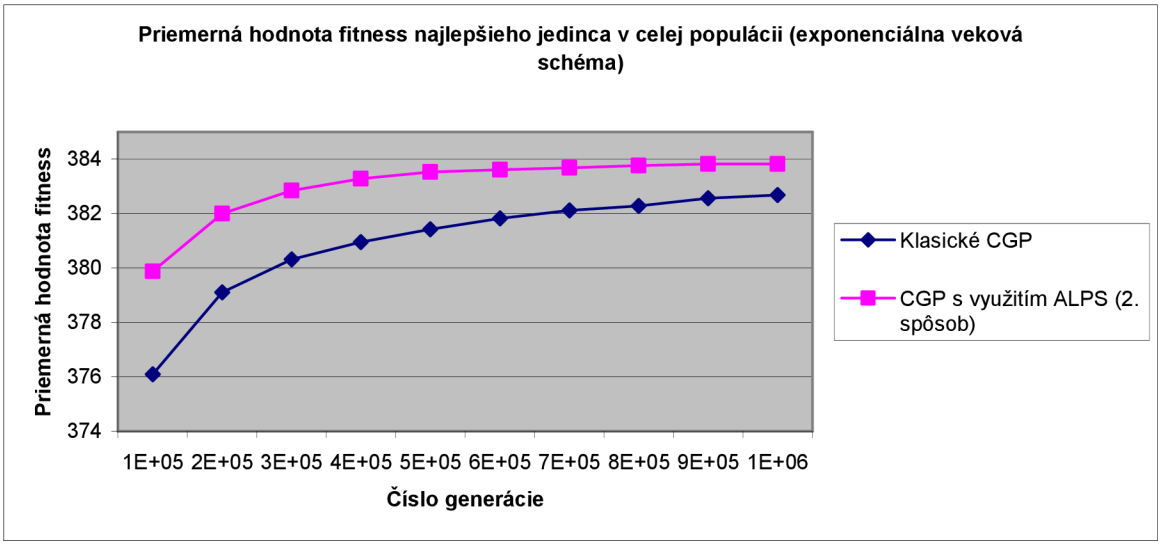
Použitie exponenciálnej vekovej schémy viedlo k podobným výsledkom ako u predchádzajúcich použitých schém, pričom so znížením parametru age-gap prišlo k ďalšiemu zníženiu priemerného počtu generácií potrebných na nájdenie vhodného riešenia. Úspešnosť riešenia bola 84%, čo je takmer identické s polynomiálnou vekovou schémou. Zrejme je je spôsobené aj podobnými rozostupmi medzi vekovými vrstvami (bolo ich použitých iba päť).

Použité parametre:

```
#define AGE_SCHEME 4                    /*exponencialna vekova schema*/
#define AGE_GAP 25000
```

Priemerný počet generácií potrebných na nájdenie vhodného riešenia	412 269.29
Priemerný počet použitých hradiel v riešení	28.60
Minimálny počet použitých hradiel v riešení	26
Maximálny počet použitých hradiel v riešení	33

Tabuľka č. 14: Získané štatistické výsledky

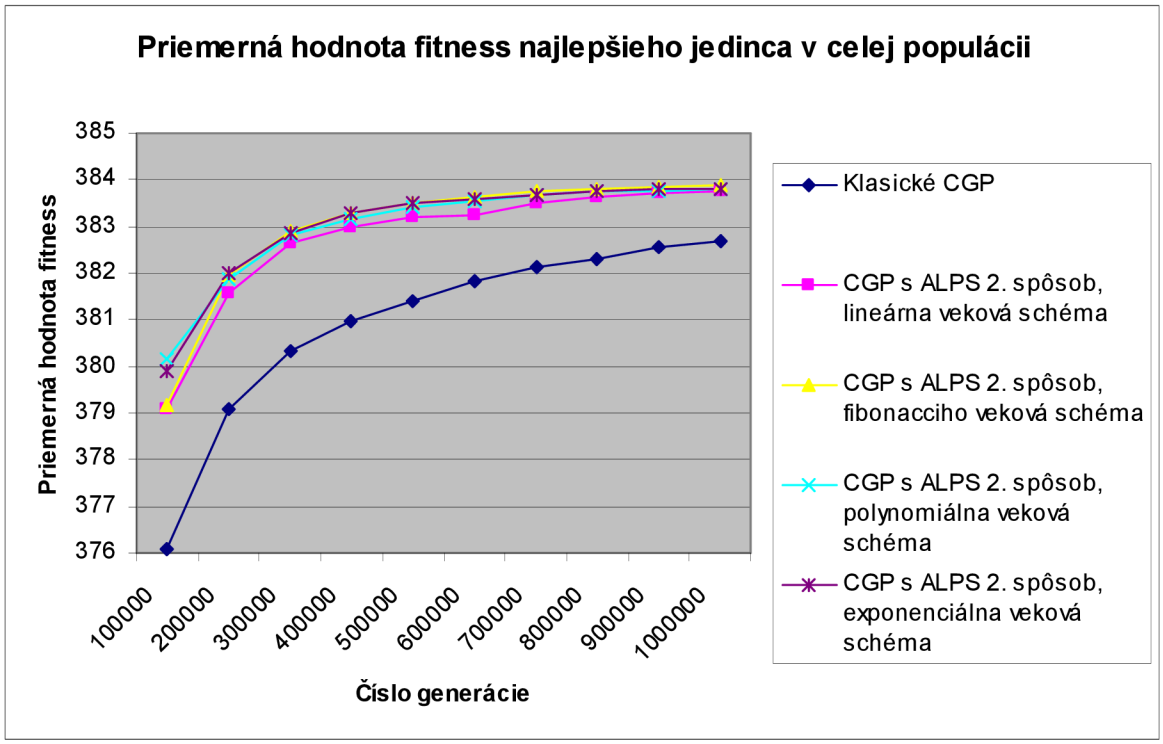


Obrázok č. 32: Priemerná hodnota fitness najlepších jedincov počas evolúcie

7.1.4.5 Zosumarizovanie výsledkov

	Priemerný počet generácií potrebných na nájdenie vhodného riešenia	Priemerný počet použitých hradiel v riešení	Minimálny počet použitých hradiel v riešení	Maximálny počet použitých hradiel v riešení
Klasické CGP	616 105.69	28.50	**26**	32
2. varianta začlenenia ALPS (lineárna vek. schéma)	501 473.73	**28.08**	**26**	**31**
2. varianta začlenenia ALPS (fibonacciho vek. schéma)	443 296.91	28.22	**26**	34
2. varianta začlenenia ALPS (polynomiálna vek. schéma)	429 388.46	28.31	**26**	32
2. varianta začlenenia ALPS (exponenciálna vek. schéma)	**412 269.29**	28.60	**26**	33

Tabuľka č. 15: Zosumarizované štatistické výsledky



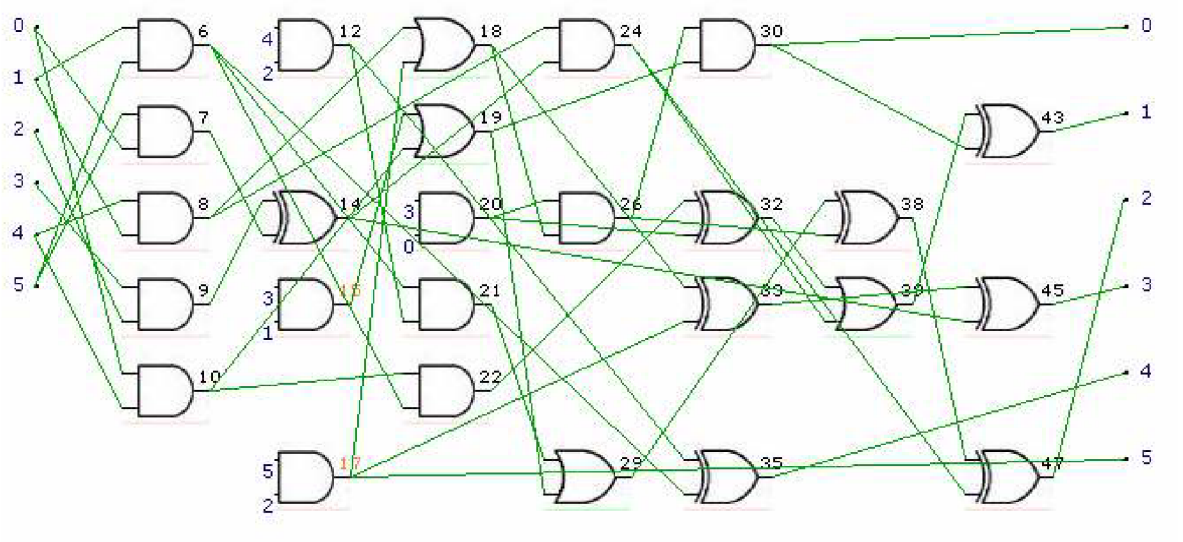
Obrázok č. 33: Priemerná hodnota fitness najlepších jedincov počas evolúcie

7.1.4.6 Záver

Nie je možné jednoznačne určiť, použitie ktorej vekovej schémy viedlo k lepším výsledkom. Čo sa týka rýchlosti nájdenia vhodného riešenia víťazí exponenciálna veková schéma, na druhej strane najvyššiu úspešnosť vykazuje fibonacciho. Ako je možné vidieť, väčšina ostatných dôležitých sledovaných parametrov (priemerný počet použitých hradiel, ich najmenší počet) ostáva zhruba rovnaký. Rovnako ako v prípade použitia 1. spôsobu využitia techniky ALPS teda neplatí, že by použitie niektorej vekovej schémy v konečnom dôsledku (za využitia limitu 1 miliónov generácií) viedlo k významne lepším výsledkom oproti ostatným, čo je možné vidieť aj v sumarizačnom grafe (obr. č. 33).

7.1.5 Najlepšie získané riešenie

Vo všetkých experimentoch malo najlepšie nájdené riešenie 26 použitých hradiel, rovnako ako v [11]. Podarilo sa mi teda vo všetkých experimentoch dosiahnuť ich relatívne nízky počet, pričom počet takýchto jedincov mierne kolísal od typu použitého spôsobu začlenenia ALPS do CGP a použitej vekovej schémy. Na obrázku 4. 34 je možné vidieť mnou získané najlepšie riešenie, s oneskorením 5T. Toto by zrejme v ďalších experimentoch mohlo byť eliminované použitím iného rozmeru mriežky pri evolúcii.



Obrázok č. 34: Najlepšie nájdené riešenie 3 – bitovej násobičky

7.2 Experiment 2 – násobenie čísiel konštantou (Multiplierless Multiple Constant Multiplication)

V rámci experimentu č. 2 budú vykonané testy CGP a jeho modifikácií využívajúce techniku ALPS na porovnanie výsledkov na probléme násobenia čísiel konštantou.

Premenná (x) môže byť násobená nielen pomocou násobičky, ale aj obvodom pozostávajúcim iba z operácií bitového posunu, sčítačiek a odčítačiek. Nájdenie optimálneho riešenia s minimálnym počtom sčítačiek a odčítačiek je NP – úplný problém, na riešenie ktorých sa používajú (ako bolo uvedené na začiatku práce) aj evolučné algoritmy.

Napríklad, premenná x môže byť násobená 71 ako:

71x = 10001112x = x << 6 + x << 2 + x << 1 + x [13]

Konkrétny problém, na ktorom budú vykonané experimenty bol prevzatý z [12], odkiaľ boli rovnako prevzaté parametre veľkosti mriežky CGP (upravený bol ale napríklad maximálny počet generácií). Vybraný bol relatívne „najľažší“ problém, s uvádzanou úspešnosťou nájdenia vhodného riešenia 68,5%.

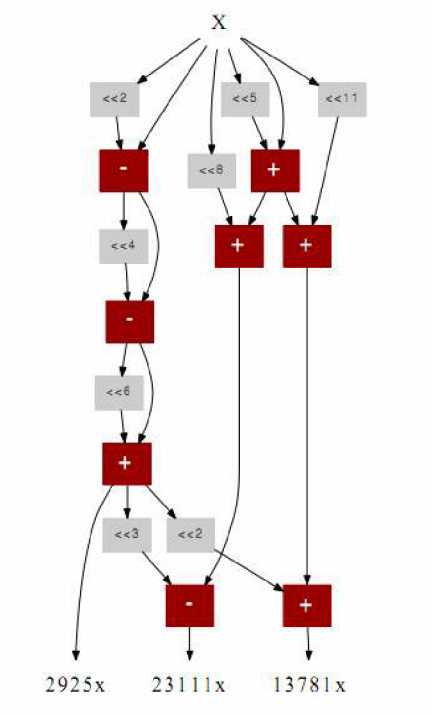
Budem teda hľadať násobky 2925, 23111 a 13781 premennej x, na mriežke o rozmeroch 5x6. Ďalšie parametre uvádzam nižšie:

```
#define VELKOST_POPULACIE 5          /*pocet jedincov populacie*/
#define POCET_MUT_GENOV 3            /*max pocet genu, ktory MOZE zmutovat*/

#define POCET_STLPCOV 5              /*pocet stlpcov*/
#define POCET_RIADKOV 6              /*pocet riadkov*/

#define POCET_VSTUPOV_BLOKU 2
#define POCET_VYSTUPOV_BLOKU 1

#define L_BACK 5                      /*parameter l-back*/

#define POCET_FUNKCII 11             /*počet pouzitych funkcii*/
```

Bolo použitých 11 funkcií, a to: WIRE (prepojka), operácie bitového posunu doľava (od 1 do 8bitov) a operácie sčítania a odčítania. Riešenie pozostáva z komponent (operácie bitového posunu, operácie sčítania a odčítania, prepojka), avšak v tejto kapitole budú nazývané hradlá, z dôvodu zachovania „názvoslovia“ z oblasti použitia CGP.

7.2.1 Známe riešenie

Uvedené najlepšie známe riešenie je prevzaté z [12] zverejnené bolo v [13]. Používa iba 8 operácií sčítania alebo odčítania (tento parameter je pri tomto probléme významný najmä preto, lebo operácie bitového posunu sú z hardvérového hľadiska veľmi „lacné“). Ako je vidieť, v tomto prípade boli použité posuny aj o šírke viac ako 8bitov.



Obrázok č. 35: Najlepšie známe riešenie problému [13], prevzaté z [12]

7.2.2 Experiment 2A

Za použitia klasického CGP na evolučný návrh obvodu na násobenie čísiel konštantou boli vykonané 2 experimenty, líšiace sa rovnako ako v experimente 1A iba maximálnym počtom generácií. Bolo tak vykonané z toho dôvodu, že kým pri prvom spôsobe implementácie ALPS v CGP bol použitý maximálny počet generácií 3 milióny (v literatúre[12] bol limit až 20 miliónov generácií), v druhom spôsobe bol vzhľadom na výrazné zvýšenie výpočtovej náročnosti pri použití druhého spôsobu začlenenia techniky ALPS do CGP použitý limit 1 milión. Za použitia týchto dvoch limitov bolo vykonané porovnanie klasického ALPS s jedným a druhým spôsobom začlenenia ALPS.

7.2.2.1 Experiment 2Ai

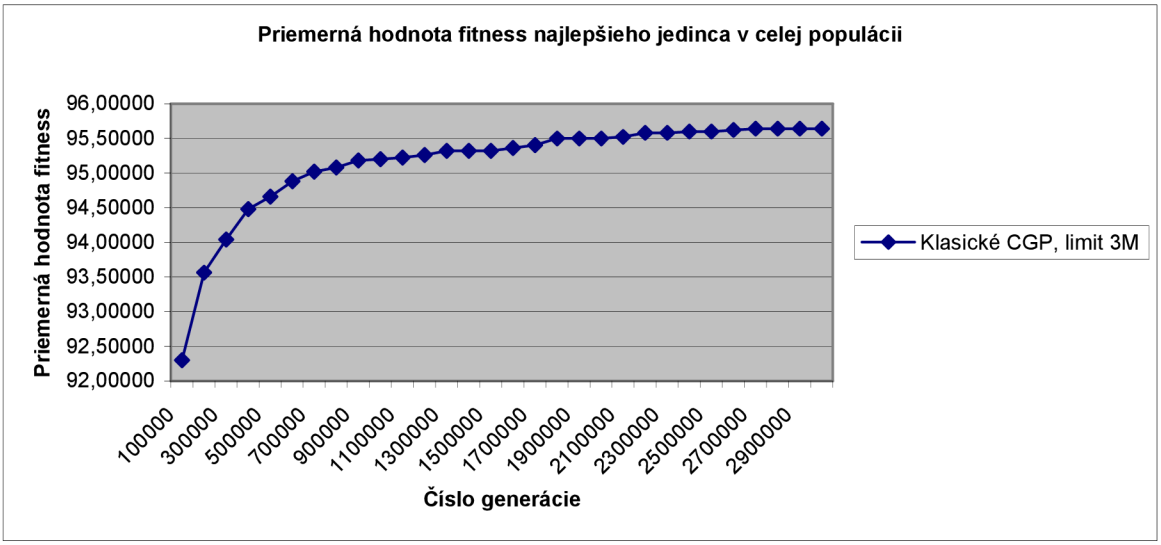
V tomto experimente bol použitý maximálny limit počtu generácií rovných 3 milióny. Ostatné nastavenia parametrov evolúcie:

```
#define POCET_GENERACII 3000000     /*max. pocet generacii evolucie*/
#define POCET_BEHOV_EVO 50          /*max. pocet behov evolucie*/
```

Priemerná úspešnosť nájdenia takéhoto obvodu bola v tomto prípade 76%, čo približne zodpovedá výsledkom uvádzaným v literatúre (68.5% [12]),pričom mierne vyššia úspešnosť mohla byť spôsobená menším počtom behov evolúcie, a taktiež zrejme inými použitými funkciami bitového posunu. Štatistické výsledky a graf priemernej hodnoty fitness najlepšieho jedinca počas evolúcie sú uvedené na v tabuľke č. 15 a na obr. č. 36.

Priemerný počet generácií potrebných na nájdenie vhodného riešenia	967 084.24
Priemerný počet použitých hradiel v riešení	22.31
Minimálny počet použitých hradiel v riešení	19
Maximálny počet použitých hradiel v riešení	25
Priemerný počet použitých ne-shiftových hradiel v riešení	13.07
Minimálny počet použitých ne-shiftových hradiel v riešení	11
Maximálny počet použitých ne-shiftových hradiel v riešení	16

Tabuľka č. 16: Získané štatistické výsledky



Obrázok č. 36: Priemerná hodnota fitness najlepšieho jedinca počas evolúcie

7.2.2.2 Experiment 2Aii

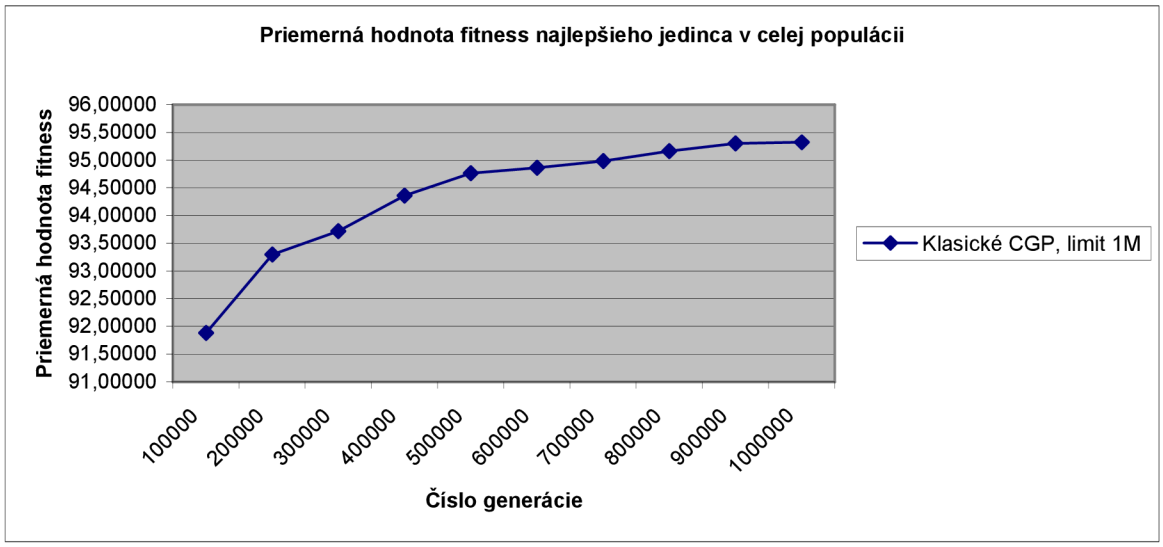
V tomto experimente bol použitý maximálny limit počtu generácií rovných 1 milión. Bolo to z toho dôvodu, aby bolo možné vykonať vhodné porovnanie do stanoveného počtu generácií s druhým spôsobom použitia techniky ALPS v CGP, kde musela byť maximálna horná hranica znížená kvôli vysokej časovej náročnosti riešenia. Ostatné nastavenia parametrov evolúcie:

```
#define POCET_GENERACII 1000000     /*max. pocet generacii evolucie*/
#define POCET_BEHOV_EVO 50          /*max. pocet behov evolucie*/
```

Priemerná úspešnosť nájdenia riešenia v tomto prípade bola 56%, čo nie je príliš razantné zníženie úspešnosti, ako tomu bolo v prípade násobičky. Ako je možné vidieť aj v literatúre, napr. [12], hľadanie obvodov na násobenie čísiel konštantou má pri použití CGP vysokú úspešnosť. Štatistické výsledky a graf priemernej hodnoty fitness najlepšieho jedinca počas evolúcie sú uvedené v tabuľke č. 16 a na obr. č. 37.

Priemerný počet generácií potrebných na nájdenie vhodného riešenia	548 792.29
Priemerný počet použitých hradiel v riešení	22.75
Minimálny počet použitých hradiel v riešení	20
Maximálny počet použitých hradiel v riešení	26
Priemerný počet použitých ne-shiftových hradiel v riešení	13.39
Minimálny počet použitých ne-shiftových hradiel v riešení	11
Maximálny počet použitých ne-shiftových hradiel v riešení	16

Tabuľka č. 17: Získané štatistické výsledky



Obrázok č. 37: Priemerná hodnota fitness najlepšieho jedinca počas evolúcie

Ako je vidieť, v oboch prípadoch sa stretávame s obdobnými javmi ako tomu bolo u evolučného návrhu násobičky 3x3 bity. A to, že pri použití limitu 1 milión generácií sa pomerne razantne zníži priemerný počet generácií potrebných na nájdenie riešenia (aj keď v tomto prípade bez takmer polovičného zníženia úspešnosti). Avšak v toto prípade neprišlo už k významnému zníženiu priemerného počtu použitých hradiel, a to ani v prípade sčítačiek/odčítačiek.

7.2.3 Experiment 2B

Experimenty vykonané v tejto časti práce sú obdobné ako u experimentu 1; teda všetky nastavenia ostávajú rovnaké ako v prípade použitia klasického CGP, ďalej bol dodefinovaný počet vekových vrstiev, ktorý ostáva pri všetkých testoch použitia rozličných vekových schém (lineárna, fibonacciho, polynomiálny a exponenciálna) rovnaký, avšak bol menený parameter age–gap, vždy v závislosti na použitej vekovej schéme. V jednotlivých podkapitolách budú uvedené dosiahnuté štatistické výsledky a grafy priemernej hodnota fitness najlepšieho jedinca v celej populácii riešení v porovnaní s klasickým CGP.

Použité parametre:

```
#define POCET_GENERACII 3000000     /*max. pocet generacii evolucie*/
#define POCET_BEHOV_EVO 50          /*max. pocet behov evolucie*/
```

7.2.3.1 Experiment 2Bi

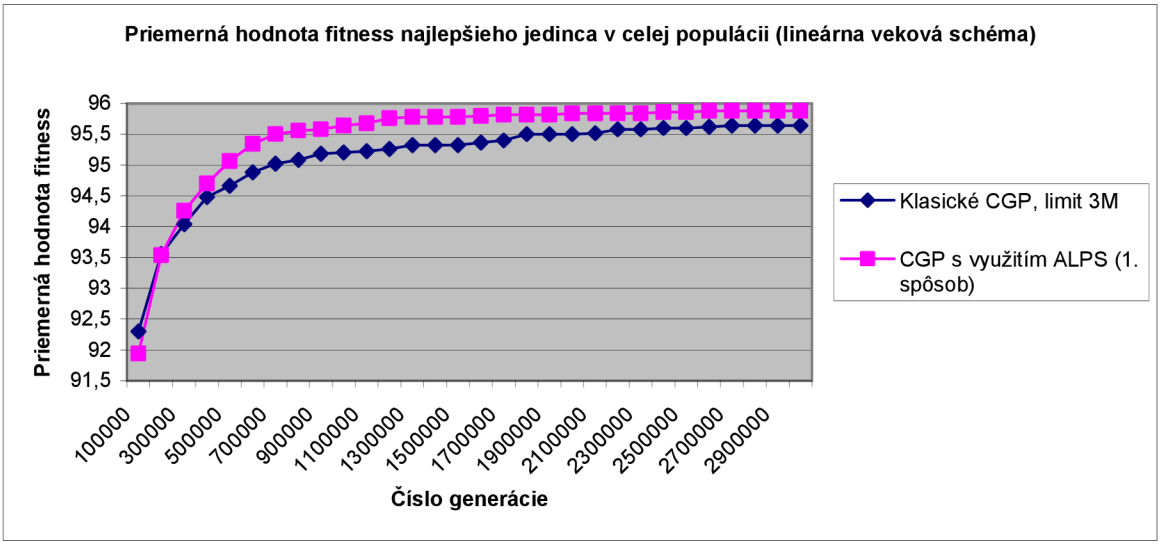
Pri použití lineárnej vekovej schémy bola dosiahnutá úspešnosť nájdenia riešenia až 88%, čo je oproti klasickému CGP značné zlepšenie. Takisto sa znížil priemerný počet potrebný na nájdenie vhodného riešenia, rovnako ako priemerný počet použitých hradiel. Toto všetko je zrejme spôsobené neustálym zavádzaním nového genetického materiálu na najnižšej vekovej vrstve a tým pádom udržovaním dostatočnej diverzity v populácii. Nižšie sú uvedené dosiahnuté štatistické výsledky a graf, kde je porovnaná priemerná hodnota fitness najlepšieho jedinca v celkovej populácii počas evolúcie s klasickým CGP.

Použité parametre:

```
#define AGE_SCHEME 1                    /*linearna vekova schema*/
#define AGE_GAP 70000
```

Priemerný počet generácií potrebných na nájdenie vhodného riešenia	764 183.48
Priemerný počet použitých hradiel v riešení	22.00
Minimálny počet použitých hradiel v riešení	18
Maximálny počet použitých hradiel v riešení	25
Priemerný počet použitých ne-shiftových hradiel v riešení	13.06
Minimálny počet použitých ne-shiftových hradiel v riešení	10
Maximálny počet použitých ne-shiftových hradiel v riešení	15

Tabuľka č. 18: Získané štatistické výsledky



Obrázok č. 38: Priemerná hodnota fitness najlepších jedincov počas evolúcie

7.2.3.2 Experiment 2Bii

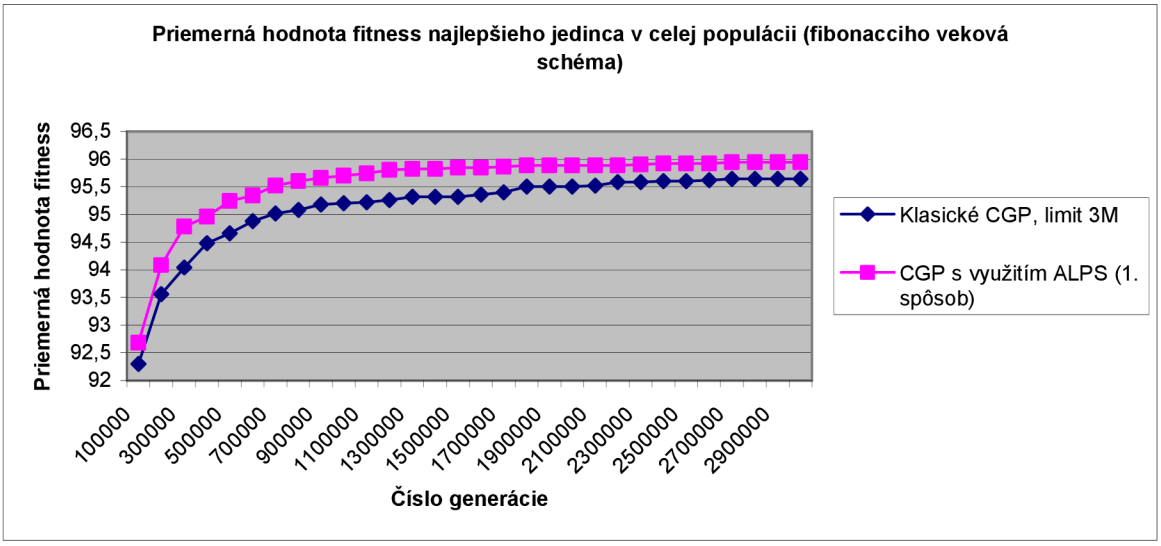
Pri použití fibonacciho vekovej schémy bola v tomto prípade dosiahnutá úspešnosť nájdenia riešenia až 94% (rovnako ako v predošlých schémach dosahuje fibonacciho veková schéma dobré výsledky). Prišlo tu (oproti lineárnej schéme alebo klasickému CGP) k zníženiu priemerného počtu generácií potrebných na nájdenie riešenia, takisto i k priemernému počtu použitých hradiel (všetkých a neshiftových).

Použité parametre:

```
#define AGE_SCHEME 2                    /*fibonacciho vekova schema*/
#define AGE_GAP 60000
```

Priemerný počet generácií potrebných na nájdenie vhodného riešenia	719 244.36
Priemerný počet použitých hradiel v riešení	21.74
Minimálny počet použitých hradiel v riešení	17
Maximálny počet použitých hradiel v riešení	25
Priemerný počet použitých ne-shiftových hradiel v riešení	12.79
Minimálny počet použitých ne-shiftových hradiel v riešení	11
Maximálny počet použitých ne-shiftových hradiel v riešení	15

Tabuľka č. 19: Získané štatistické výsledky



Obrázok č. 39: Priemerná hodnota fitness najlepších jedincov počas evolúcie

7.2.3.3 Experiment 2Biii

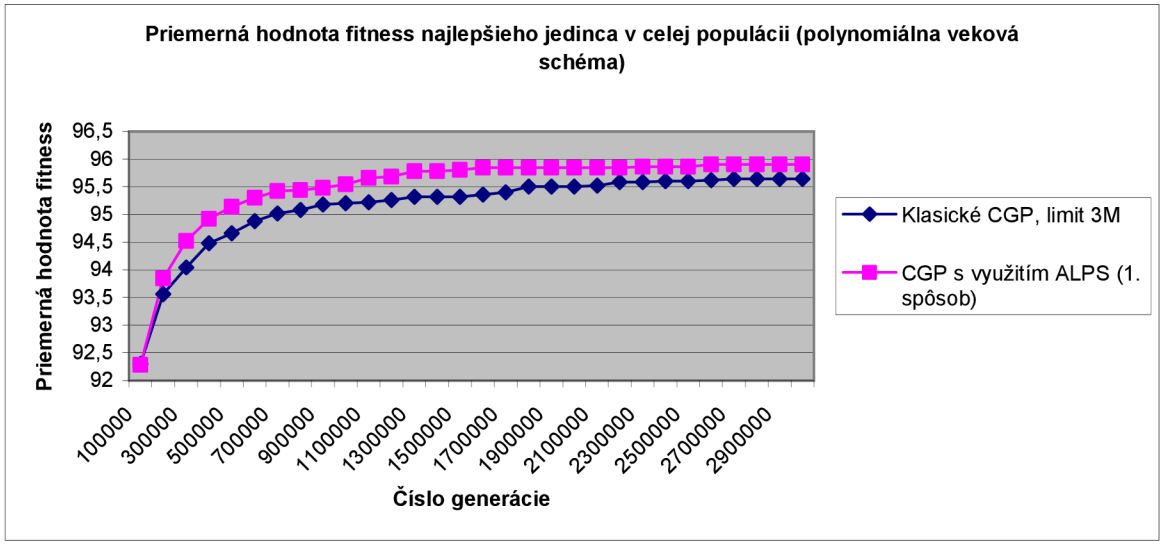
Pri použití polynomiálnej vekovej schémy sa oproti predchádzajúcim dvom vekovým schémam mierne zvýšil priemerný počet generácií potrebných na nájdenie vhodného riešenia (čo zrejme súvisí s väčšími rozostupmi medzi vekovými schémami, kde občas jedna vrstva „zostarne“, no zatiaľ neprišlo k prísunu nového gen. materiálu). Napriek vysokej úspešnosti sa ale mierne zvýšil priemerný počet použitých hradiel.

Použité parametre:

```
#define AGE_SCHEME 3                    /*polynomialna vekova schema*/
#define AGE_GAP 50000
```

Priemerný počet generácií potrebných na nájdenie vhodného riešenia	806 493.67
Priemerný počet použitých hradiel v riešení	22.27
Minimálny počet použitých hradiel v riešení	19
Maximálny počet použitých hradiel v riešení	26
Priemerný počet použitých ne-shiftových hradiel v riešení	13.22
Minimálny počet použitých ne-shiftových hradiel v riešení	10
Maximálny počet použitých ne-shiftových hradiel v riešení	16

Tabuľka č. 20: Získané štatistické výsledky



Obrázok č. 40: Priemerná hodnota fitness najlepších jedincov počas evolúcie

7.2.3.4 Experiment 2Biv

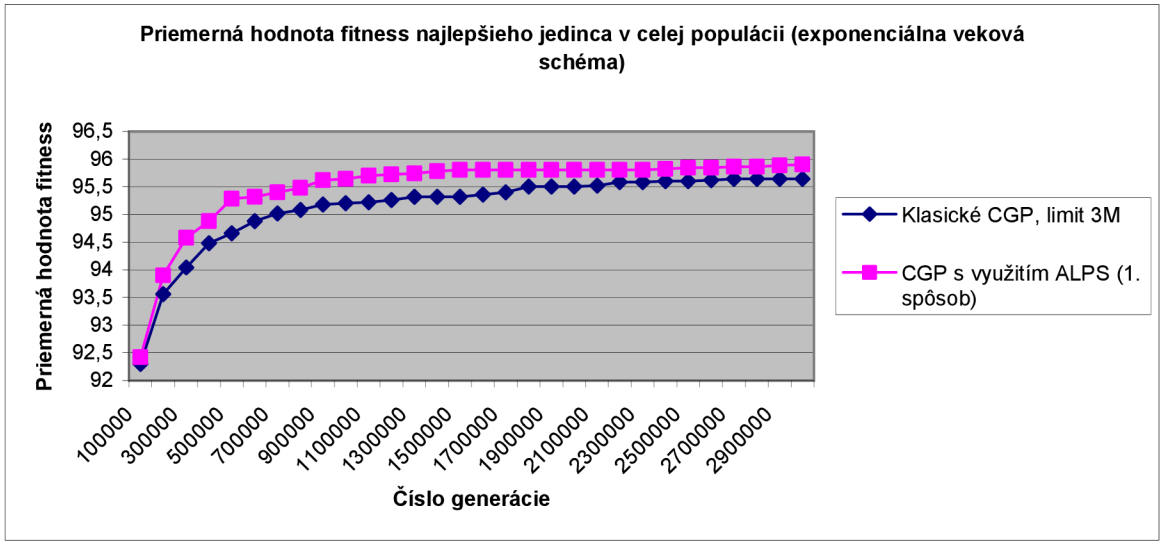
Pri použití exponenciálnej vekovej schémy, rovnako ako v experimente 1Biv, prišlo zvýšeniu priemerného počtu generácií potrebných na nájdenie vhodného riešenia, avšak táto použitá schéma dosahuje najlepšie priemerné ukazovatele počtu použitých hradiel (aj neshiftových). Úspešnosť bola 92%.

Použité parametre:

```
#define AGE_SCHEME 4                    /*exponencialna vekova schema*/
#define AGE_GAP 50000
```

Priemerný počet generácií potrebných na nájdenie vhodného riešenia	820 219.11
Priemerný počet použitých hradiel v riešení	21.54
Minimálny počet použitých hradiel v riešení	18
Maximálny počet použitých hradiel v riešení	25
Priemerný počet použitých ne-shiftových hradiel v riešení	12.63
Minimálny počet použitých ne-shiftových hradiel v riešení	10
Maximálny počet použitých ne-shiftových hradiel v riešení	16

Tabuľka č. 21: Získané štatistické výsledky

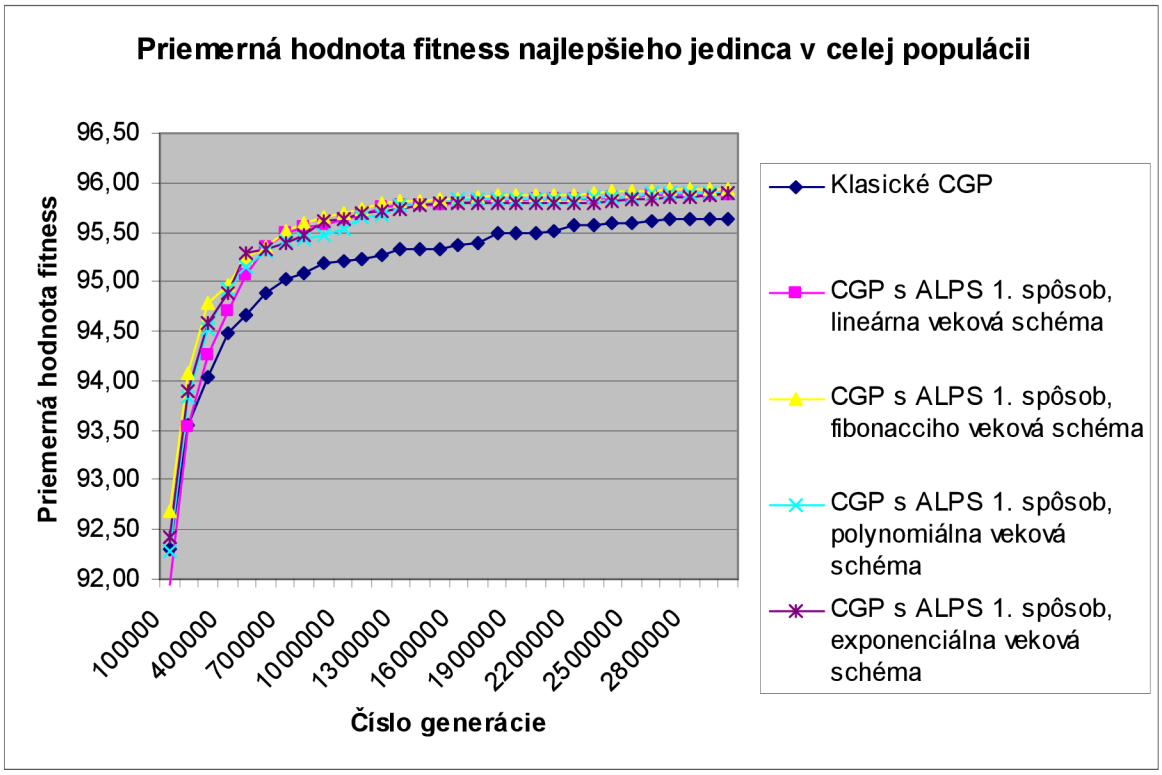


Obrázok č. 41: Priemerná hodnota fitness najlepších jedincov počas evolúcie

7.2.3.5 Zosumarizovanie výsledkov experimentu 2B

	Priem. počet generácií potrebných na nájdenie vhodného riešenia	Priem. počet použitých hradiel v riešení	Min. počet použitých hradiel v riešení	Max. počet použitých hradiel v riešení	Priem. počet použitých ne-shiftových hradiel v riešení	Min. počet použitých ne-shiftových hradiel v riešení	Max. počet použitých ne-shiftových hradiel v riešení
Klasické CGP	967084.24	22.32	19	**25**	13.08	11	16
1. varianta začlenenia ALPS (lineárna vek. schéma)	764183.48	22.00	18	**25**	13.06	**10**	**15**
1. varianta začlenenia ALPS (fibonacciho vek. schéma)	**719244.36**	21.74	**17**	**25**	14,20	11	**15**
1. varianta začlenenia ALPS (polynomiálna vek. schéma)	806493.67	22.27	19	26	13.22	**10**	16
1. varianta začlenenia ALPS (exponenciálna vek. schéma)	820219.11	**21.54**	18	25	**12.63**	**10**	16

Tabuľka č. 22: Zosumarizované štatistické výsledky



Obrázok č. 42: Priemerná hodnota fitness najlepších jedincov počas evolúcie

7.2.3.6 Záver

Ako je vidieť, nie je možné jednoznačne určiť, ktorá veková schéma bola pri návrhu obvodu na násobenie čísla konštantou najvhodnejšia. Ak by sme uvažovali parametre ako je rýchlosť nájdenia vhodného riešenia, či minimálny počet použitých hradiel s operáciami sčítania a odčítania, rovnako prevažovala by fibonacciho veková schéma.

7.2.4 Experiment 2C

V tejto podkapitole budú zobrazené výsledky experimentov za použitia druhého spôsobu začlenenia techniky ALPS do CGP. Všetky nastavenia ostávajú rovnaké ako v prípade použitia klasického CGP, ďalej bol ako v prípade experimentu 2B dodefinovaný počet vekových vrstiev, a v tomto prípade i počet subpopulácií nachádzajúcich sa v jednotlivých vekových vrstvách. Štruktúra kapitoly je obdobná ako aj u 2B, budú uvedené štatistické výsledky a grafy priemernej hodnoty fitness najlepšieho jedinca v celej populácii riešení. Toto bude porovnávané voči výsledkom získaným z experimentu 2Aii, pretože vzhľadom na pribudnutie ďalšieho rozmeru (počtu subpopulácií) prišlo k ďalšiemu zpomaleniu procesu evolúcie, preto bol maximálny počet generácií znížený na 1 milión. Počet behov evolúcie ostáva zachovaný ako v predošlom experimente 2B (50 behov), čo by malo poskytnúť dostatočne relevantné štatistické výsledky.

Použité parametre:

```
#define POCET_GENERACII 1000000     /*max. pocet generacii evolucie*/
#define POCET_BEHOV_EVO 50          /*max. pocet behov evolucie*/
#define POCET_SUBPOPULACII 3        /*pocet subpopulacii*/
```

7.2.4.1 Experiment 2Ci

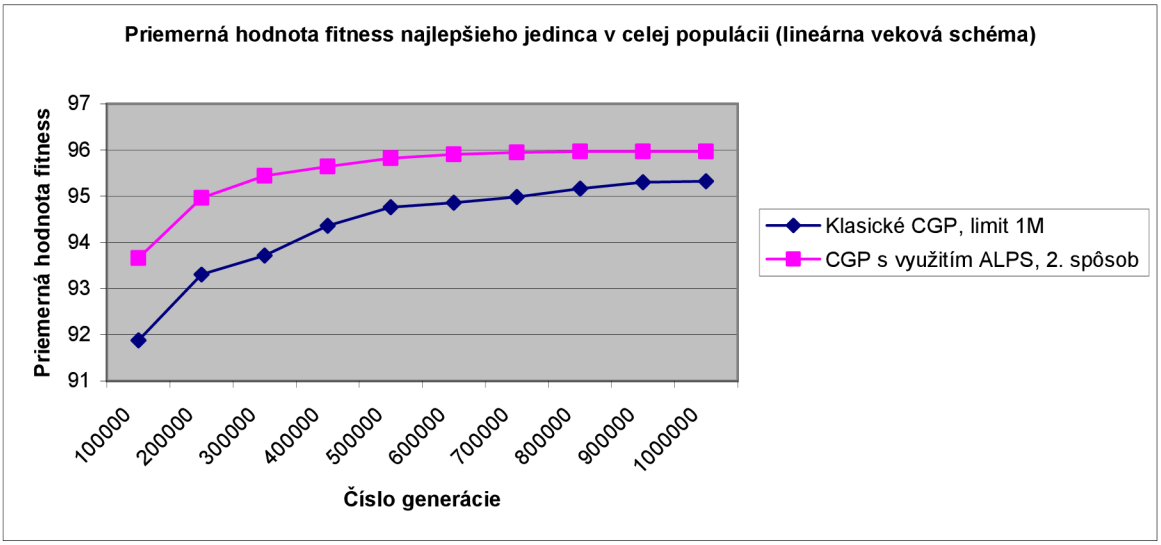
Pri použití lineárnej vekovej schémy sa znížili v podstate všetky sledované štatistické parametre oproti použitiu klasického CGP, najmä rýchlosť nájdenia vhodného kandidátneho riešenia. Úspešnosť nájdenia vhodného riešenia bola rovnako ako pri experimente 2Bi až 96%. Tu bolo nájdené aj najlepšie vyevolvované riešenie, ktoré je zrovnateľné s riešeniami dosiahnutými v literatúre [12, 13]

Použité parametre:

```
#define AGE_SCHEME 1                /*linearna vekova schema*/
#define AGE_GAP 100000
```

Priemerný počet generácií potrebných na nájdenie vhodného riešenia	322 902.83
Priemerný počet použitých hradiel v riešení	21.12
Minimálny počet použitých hradiel v riešení	17
Maximálny počet použitých hradiel v riešení	24
Priemerný počet použitých ne-shiftových hradiel v riešení	12.45
Minimálny počet použitých ne-shiftových hradiel v riešení	10
Maximálny počet použitých ne-shiftových hradiel v riešení	14

Tabuľka č. 23: Získané štatistické výsledky



Obrázok č. 43: Priemerná hodnota fitness najlepších jedincov počas evolúcie

7.2.4.2 Experiment 2Cii

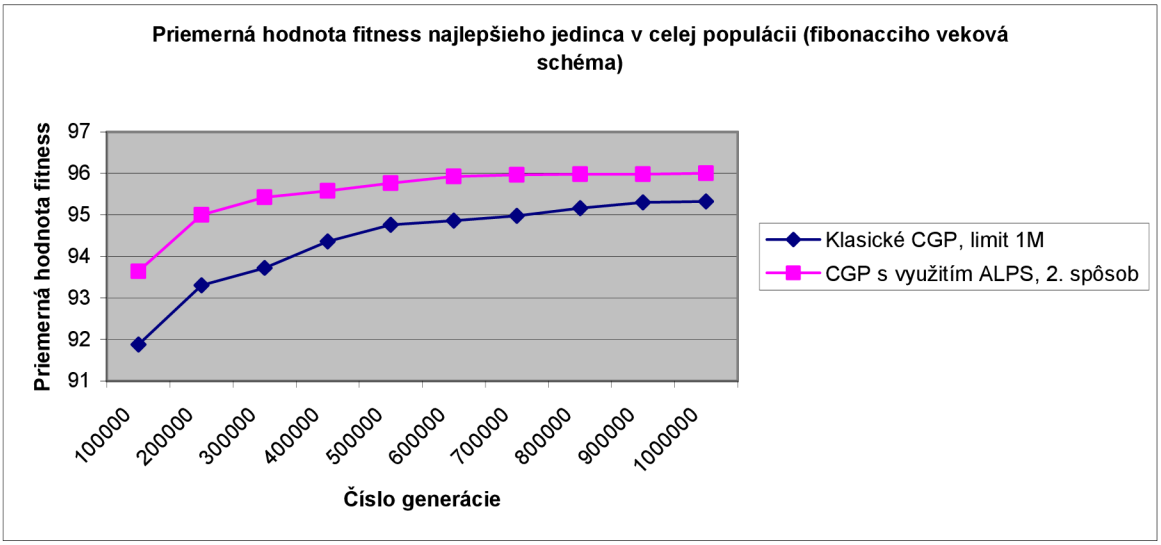
Úspešnosť nájdenia vhodného kandidátneho riešenia bola 100%, výsledky sú obdobné ako pri použití lineárnej vekovej schémy (pričom ale fibonacciho veková schéma znovu dosahuje vysokú úspešnosť). Ostatné sledované parametre sú zrovnateľné s lineárnou vekovou schémou, prišlo iba k miernemu zvýšeniu minimálneho počtu použitých hradiel.

Použité parametre:

```
#define AGE_SCHEME 2                    /*fibonacciho vekova schema*/
#define AGE_GAP 90000
```

Priemerný počet generácií potrebných na nájdenie vhodného riešenia	349 804.92
Priemerný počet použitých hradiel v riešení	21.18
Minimálny počet použitých hradiel v riešení	19
Maximálny počet použitých hradiel v riešení	24
Priemerný počet použitých ne-shiftových hradiel v riešení	12.44
Minimálny počet použitých ne-shiftových hradiel v riešení	11
Maximálny počet použitých ne-shiftových hradiel v riešení	15

Tabuľka č. 24: Získané štatistické výsledky



Obrázok č. 44: Priemerná hodnota fitness najlepších jedincov počas evolúcie

7.2.4.3 Experiment 2Ciii

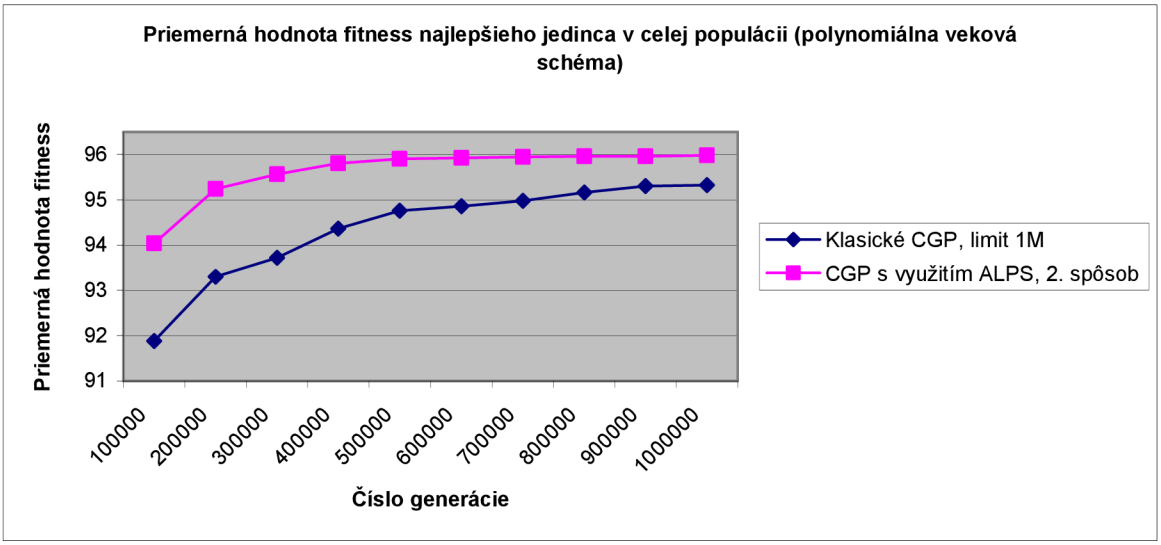
Pri použití polynomiálnej vekovej schémy bola úspešnosť nájdenia vhodného jedinca 98%, prišlo dokonca k zvýšeniu rýchlosti nájdenia vhodného kandidátneho riešenia (čo zrejme súvisí so zmenšením parametru age-gap a schopnosti rýchlejšie prehľadávať stavový priestor možných riešení). Ostatné sledované parametre ostávajú znovu porovnateľné s ostatnými vekovými schémami.

Použité parametre:

```
#define AGE_SCHEME 3                    /*polynomialna vekova schema*/
#define AGE_GAP 35000
```

Priemerný počet generácií potrebných na nájdenie vhodného riešenia	292 374.12
Priemerný počet použitých hradiel v riešení	21.39
Minimálny počet použitých hradiel v riešení	17
Maximálny počet použitých hradiel v riešení	24
Priemerný počet použitých ne-shiftových hradiel v riešení	12.69
Minimálny počet použitých ne-shiftových hradiel v riešení	10
Maximálny počet použitých ne-shiftových hradiel v riešení	15

Tabuľka č. 25: Získané štatistické výsledky



Obrázok č. 45: Priemerná hodnota fitness najlepších jedincov počas evolúcie

7.2.4.4 Experiment 2Civ

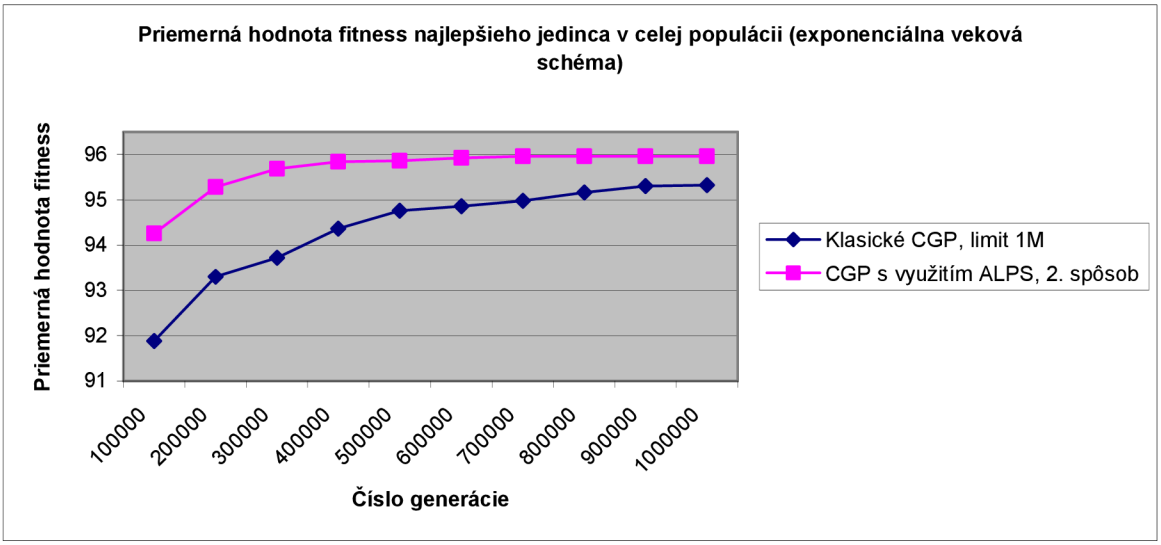
Pri použití exponenciálnej vekovej schémy ostala úspešnosť nájdenia vhodného riešenia na 98% (rovnako ako u polynomiálnej schémy), pričom ostatné parametre ostali znovu porovnateľné ako u ostatných vekových schém. Najvýraznejšie sa znížil priemerný počet potrebných generácií na nájdenie vhodného jedinca, čo je znovu spôsobené rýchlym prehľadávaním stavového priestoru na najnižšej vrstve pri generovaní nových riešení.

Použité parametre:

```
#define AGE_SCHEME 4                    /*exponencialna vekova schema*/
#define AGE_GAP 25000
```

Priemerný počet generácií potrebných na nájdenie vhodného riešenia	243 235.10
Priemerný počet použitých hradiel v riešení	21.53
Minimálny počet použitých hradiel v riešení	18
Maximálny počet použitých hradiel v riešení	25
Priemerný počet použitých ne-shiftových hradiel v riešení	12.84
Minimálny počet použitých ne-shiftových hradiel v riešení	10
Maximálny počet použitých ne-shiftových hradiel v riešení	15

Tabuľka č. 26: Získané štatistické výsledky

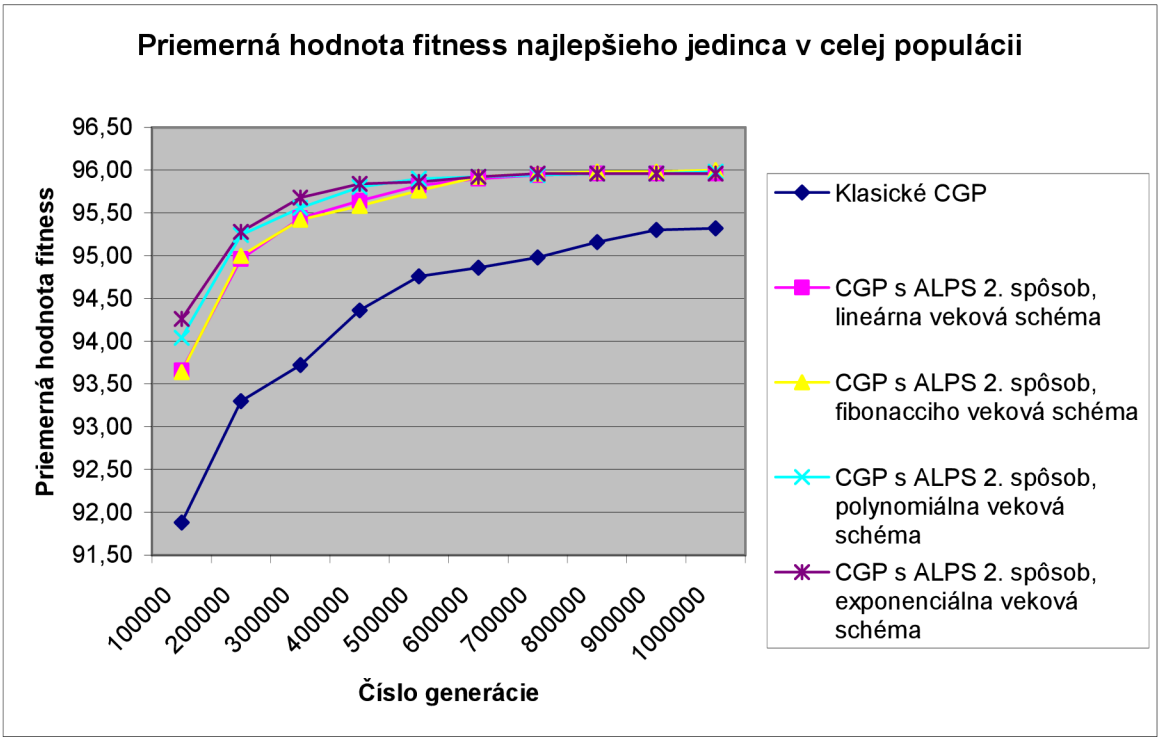


Obrázok č. 46: Priemerná hodnota fitness najlepších jedincov počas evolúcie

7.2.4.5 Zosumarizovanie výsledku experimentu 2C

	Priem. počet generácií potrebných na nájdenie vhodného riešenia	Priem. počet použitých hradiel v riešení	Min. počet použitých hradiel v riešení	Max. počet použitých hradiel v riešení	Priem. počet použitých ne-shiftových hradiel v riešení	Min. počet použitých ne-shiftových hradiel v riešení	Max. počet použitých ne-shiftových hradiel v riešení
Klasické CGP	548792.29	22.75	20	26	13.39	11	16
2. varianta začlenenia ALPS (lineárna vek. schéma)	322902.83	**21.13**	**17**	**24**	12.46	**10**	**14**
2. varianta začlenenia ALPS (fibonacciho vek. schéma)	349804.92	21.18	19	**24**	**12.44**	11	15
2. varianta začlenenia ALPS (polynomiálna vek. schéma)	292374.12	21.39	**17**	**24**	12.69	**10**	15
2. varianta začlenenia ALPS (exponenciálna vek. schéma)	**243235.10**	21.53	18	25	12.84	**10**	15

Tabuľka č. 27: Zosumarizované štatistické výsledky



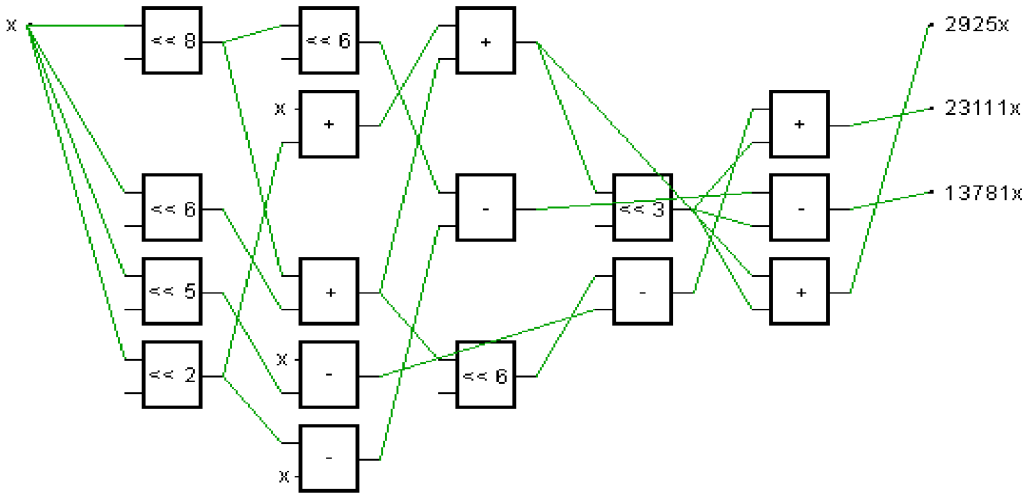
Obrázok č. 47: Priemerná hodnota fitness najlepších jedincov počas evolúcie

7.2.4.6 Záver

Použitie druhého spôsobu začlenenia techniky ALPS do CGP, s ktorým boli vykonané experimenty na probléme násobenia čísiel konštantou prinieslo dobré výsledky oproti použitiu klasického CGP, rozdiel je najmä výraznejší pri použití limitu jedného milióna generácií. Avšak rovnako ako v predchádzajúcich prípadoch nie je možné jednoznačne určiť, ktorá veková schéma podáva najlepšie výsledky, s blížiacim sa limitom konvergujú všetky k rovnakému maximu.

7.2.5 Najlepšie získané riešenie

V najlepšom známom riešení [13] bolo použitých iba 8 operácií sčítania a odčítania a 8 bitových posunov. Mnou získané najlepšie riešenie obsahovalo až 10 operácií sčítania a odčítania a 7 operácií bitových posunov. Toto bolo zrejme spôsobené tým, že boli použití relatívne vysoké hodnoty konštánt, pričom ja som vo vykonaných testoch použil maximálne 8 – bitový posun; na rozdiel od známeho riešenia, kde boli použité posuvy viacerých bitov. Napriek tomu sa domnievam, že sa mi podarilo dosiahnuť relatívne dobré riešenie blížiace sa optimu, ktoré je znázornené na obr. č. 42.



Obrázok č. 42: Najlepšie získané riešenie problému počas evolúcie

8 Záver

V rámci diplomového projektu som vypracoval štúdiu uvádzajúcu do problematiky softcomputingu, konkrétne časti evolučných algoritmov. V prvej časti sú rozobrané jednotlivé typy evolučných algoritmov. V ďalších kapitolách je vypracovaná štúdia na tému kartézskeho genetického programovania a novej techniky používanej pri evolučnom návrhu ALPS, ktorá zabraňuje predčasnej konvergencii a udržuje dostatočnú diverzitu populácie. Posledná časť obsahuje dva spôsoby návrhu začlenenia techniky ALPS do kartézskeho genetického programovania.

Nasledovala implementácia klasického CGP a jeho ďalších dvoch variánt obsahujúcimi začlenenie techniky ALPS navrhnutými v zimnom semestri v rámci semestrálního projektu. Na týchto implementáciách boli vykonané experimenty so zadanými testovacími úlohami, ktorými boli návrh 3 – bitovej násobičky a návrh obvodu na násobenie čísiel konštantou. V obidvoch prípadoch viedlo použitie techniky ALPS k významnému zlepšeniu evolučného procesu. Zvýšila sa najmä úspešnosť a rýchlosť nájdenia vhodného kandidátneho riešenia, čo je spôsobené zavádzaním nových riešení do populácie na najnižšej vekovej vrstve, a tým umožnenie intenzívnejšie prehľadávanie stavového priestoru možných riešení, čo umožňuje vymaniť sa z lokálnych miním. Takisto prišlo k miernemu zlepšeniu parametrov ako je priemerný počet použitých hradiel.

Oproti tomu ležia ale isté problémy. Sú nimi napríklad vyššia výpočtová náročnosť na hardvér vzhľadom na použitie viacerých vekových vrstiev (pri druhom spôsobe sa náročnosť zvýši použitím subpopulácií vo vrstvách), či problém nekontinuálneho presunu genetického materiálu medzi jednotlivými vrstvami, ktorý sa odohráva skokovito.

Experimenty boli vykonané za použitia rozličných vekových schém, z ktorých avšak nie je možné usúdiť, že by tá – ktorá bola vhodnejšia na riešenie niektorého typu problému. Dá sa povedať, že všetky použité vekové schémy dosahovali veľmi podobné výsledky pri jednom aj druhom spôsobe začlenenia techniky ALPS do CGP.

Na základe vykonaných experimentov a získaných výsledkov je možné povedať, že technika ALPS prináša zlepšenie výsledkov evolučného procesu aj pri jej použití v kartézskom genetickom programovaní, a to ako vo zvýšení úspešnosti evolúcie ako i vlastností jej výsledkov, viď napríklad zníženie počtu použitých hradiel.

Do budúcnosti by bolo určite vhodné naprogramovať začlenenie techniky ALPS do CGP viacvláknovo, prípadne vykonať hardvérovú implementáciu.

Literatúra

[1] Kvasnička, V., Pospíchal J., Tiňo P.: Evolučné algoritmy. Vydavatelství STU Bratislava, 2000, ISBN 80-227-1377-5, 215 s.

[2] Schwartz, J., Sekanina, L.: Aplikované evoluční algoritmy - studijní opora. FIT VUT v Brně, Brno, 2006.

[3] Sekanina, L.: Evolvable Components : From Theory to Hardware Implementations. Berlin : Springer, 2004, ISBN 3-540-40377-9, 194 s.

[4] Satola, R.: Evoluční strategie : ES. Referát na predmet Aplikované evoluční algoritmy. FIT VUT v Brně, Brno, 2005.

[5] Satola, R.: Genetické algoritmy : ES. Referát na predmet Aplikované evoluční algoritmy. FIT VUT v Brně, Brno, 2005.

[6] Miller, J. F., Job, D., Vassilev, V.K.: Principles in the Evolutionary Design of Digital Circuits—Part I. Genetic Programming and Evolvable Machines. 2000, vol. 1, no. 1-2, s. 7-35.

[7] Vašíček, Z., Sekanina, L.: Evoluční návrh kombinačních obvodů,In:Elektrorevue-www.elektrorevue.cz, roč. 2004, č. 43, Brno, CZ, s. 1-6, ISSN 1213-1539
Dostupné na URL: < http://www.elektrorevue.cz/clanky/04043/index.html>

[8] Sekanina, L. Kartézské genetické programování – prednáška na predmet Biologií inspirované počítače. FIT VUT v Brně, Brno, 2005.

[9] Hornby, Gregory S.: ALPS: the age-layered population structure for reducing the problem of premature convergence. In GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation. New York, USA : ACM, 2006. ISBN 1-59593-186-4, s. 815-822.

[10] Andrássyová, E.: Genetický algoritmus ako technika dolovania dát – písomná práca k dizertačnej skúške. Košice, 1999. 31 s. Dostupné na URL: <http://neuron-ai.tuke.sk/~andrassy/publikacie/minimovka/minimovka.doc>.

[11] Wang, J., Chong Ho, L.: Evolutionary design of combinational logic circuits using VRA processor. IEICE Electronics Express. 2009, vol. 6, no. 3, s. 141-147.

[12] Vašíček, Z., et al.: On Evolutionary Synthesis of Linear Transforms in FPGA. In Proc. Of Evolvable Systems: From Biology to Hardware Configuration. Lecture Notes in Computer Science, vol. 5216, Springer Berlin 2008, s. 141-152.

[13] Voronenko, Y., Puschel, M.: Multiplierless multiple constant multiplication. ACM Trans. Algorithms. Vol. 3, no. 2, s. 38.

Zoznam príloh

Dodatok A – Návod na prípravu spustenia evolúcie a nastavenie jej parametrov

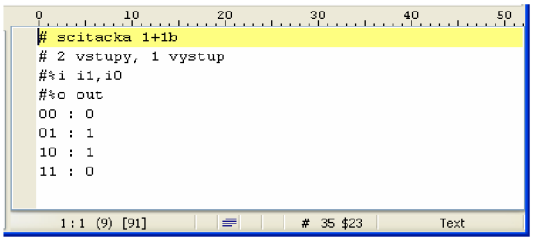
Dodatok B – Obsah priloženého CD

Dodatok A

Návod na prípravu spustenia evolúcie a nastavenie jej parametrov

Kombinačné logické obvody

Pre spustenie evolúcie je potrebné zakódovať konkrétny problém do požadovaného tvaru. K tomuto je použitý program od p. Vašíčka tab2h. Najprv je potrebné definovať pravdivostnú tabuľku požadovaného obvodu, viď obr. č. A1:



Obrázok č. A.1: Pravdivostná tabuľka jednobitovej sčítačky

Potom po zadaní príkazu ***tab2h pravdivostnatabulka.txt > pravdivostnatabulka.h*** dostaneme hlavičkový súbor, ktorý je potrebné prilinkovať do súboru *cgp.h*:

```
#include "pravdivostnatabulka.h"
```

Uvedený spôsob platí pre kombinačné logické obvody.

Násobenie čísiel konštantou

V prípade, že chceme pomocou CGP riešiť problém násobenia čísiel konštantou, musíme modifikovať (vytvoriť) hlavičkový súbor, nasledovným spôsobom:

```
#define POPIS "#MCM\n"

#define PARAM_IN 1                  /*pocet vstupov obvodu, tu 1*/
#define PARAM_OUT 3                 /*pocet vystupov kombinacneho obvodu*/
/**Inicializacia datoveho pola*/
#define init_data(a) \
  a[0]=0x00000001;\                 /*Konstanta, napriklad 1*/
  a[1]=0x00000B6D;\                 /*1. nasobok konstanty*/
  a[2]=0x00005A47;\                 /*2. nasobok konstanty*/
  a[3]=0x000035D5;                  /*3. nasobok konstanty atd...*/

/**Pocet prvkov pola --> PARAM_IN + PARAM_OUT*/
#define DATASIZE 4
```

Ako vidieť, je potrebné definovať počet vstupov (v tomto prípade 1), ďalej počet výstupov (počet násobkov konštanty) a veľkosť dátového poľa. Takto editovaný súbor je takisto potrebné v hlavičkovom súbore *cgp.h* prilinkovať:

```
include "mcm.h"
```

Nastavenie parametrov evolúcie

Kompletné nastavenie parametrov evolúcie je možné vykonať editovaním hlavičkového súboru *cgp.h*, kde je možné definovať maximálny počet behov evolúcie, počet generácií, pri CGP rozšírenom o ALPS počet vekových vrstiev, počet subpopulácií atď. Počas evolúcie sú takisto zaznamenávané štatistiky o priebehu evolúcie, a to do dvoch súborov:
textový, kam sa ukladajú informácie o čísle prvej generácie, kedy bolo nájdené vhodné kandidátne riešenie, na konci evolúcie doplnené o číslo minimálneho počtu použitých blokov. V prípade problému násobenia čísiel konštantou je doplnený ešte počet operácií sčítania a odčítania.
excelovský, kam sa ukladajú po stanovenom počte generácií informácie o maximálnej hodnote fitness najlepšieho jedinca v populácii.
Po každom úspešnom behu evolúcie je zaznamenaný najlepší jedinec do súboru vo formáte, ktorý umožňuje zobrazenie programom CGPViewer od p. Ing. Vašíčka, a to v tvare:
„*chromozom<číslo_behu_evolúcie>.chr*“.

Príklad definície hlavičkového súboru *cgp.h*:

```
#include "posuvy.h"

/*Nazvy suborov so statistikami*/
#define NAZOV_SUBORU_STATISTIK "statistiky.txt"
#define NAZOV_SUBORU_EXCEL "statistiky.xls"

#define VELKOST_POPULACIE 5          /*pocet jedincov populacie*/
#define POCET_MUT_GENOV 3            /*max. pocet mutovanych genov*/
#define POCET_FUNKCII 11             /*pocet pouzitych funkcii*/

#define POCET_STLPCOV 5              /*pocet stlpcov*/
#define POCET_RIADKOV 6              /*pocet riadkov*/

#define POCET_VSTUPOV_BLOKU 2
#define POCET_VYSTUPOV_BLOKU 1

#define L_BACK 5                     /*parameter l-back*/

#define POCET_GENERACII 1000000      /*max. pocet generacii evolucie*/
#define POCET_BEHOV_EVO 20           /*pocet behov evolucie*/

#define AGE_SCHEME 1                 /*pouzita vekova schema*/
#define AGE_GAP 100000               /*definicia parametru age-gap*/
#define POCET_VEKOVYCH_VRSTIEV 5     /*pocet vekovych vrstiev*/
#define POCET_SUBPOPULACII 3         /*pocet jednotlivych sub-populacii*/


#define POCET_STATISTIKY 100000      /*po kolko gen. statistika do excelu*/
```

Dodatok B

Obsah priloženého CD

Adresárová štruktúra priloženého dátového média:

- howto – obsahuje manuál k použitiu aplikácií
- prog – aplikácie pre zakódovanie problému/zobrazenie riešenia (priložené so zvolením p. Ing. Vašíčka)
  - *tab2h* - program pre zakódovanie kombinačného obvodu
  - *viewer* – program na zobrazovanie vyevolvovaných riešení
- src – zdrojové kódy pre klasické CGP a jeho 2 variánt s využitím ALPS
- stat – získané štatistické výsledky a uložené jednotlivé vyhovujúce jedince
- thesis – obsahuje text diplomovej práce vo formátoch *.doc* a *.pdf*