# Genetické programovanie, symbolická regresia a boolovské funkcie

Ján Borovský
borovsky@pobox.sk

March 28, 2000

## Genetické programovanie

Genetické programovanie je metóda na riešenie problémov, nezávislá na obore, pri ktorej sa vyvíjajú počítačové programy na riešenie alebo čiastočné riešenie problémov. Je založená na Darwinovskom princípe evolúcie a je vlastne modifikáciou genetického algoritmu, ktorú navrhol John Koza.

Genetické programovanie (GP) a genetické algoritmy (GA) majú preto veľa spoločných čŕt. Oba prístupy udržiavajú množinu nezávislých riešení, ktoré sú reprezentované ako jedince v populácii. Oba bežia v cykle, v ktorom vytvárajú nové generácie kopírovaním alebo modifikáciou pôvodných členov populácie, pričom túto modifikáciu realizuje množina genetických operátorov (reprodukcia, kríženie, mutácia). Na určenie, ktoré jedince budú použité v ďalšej generácii sa používa operátor selekcie, zvýhodňujúci jedince s väčšou silou (fitness). Fitness je počítaná na základe účelovej funkcie, založenej na vyhodnotení úspešnosti jednotlivých členov populácie. Obyčajne vyjadruje odchýlku medzi požadovaným a dosiahnutým výsledkom. Čím viac sa blíži k nule, tým je príslušný program úspešnejší (lepší). Fitness môže taktiež zohľadňovať ďalšie parametre ako čas, efektívnosť, náročnosť na výpočtové zdroje a podobne.

Všimnime si teraz najdôležitejší rozdiel medzi GP a GA. Ako reprezentačná štruktúra sa pri genetickom programovaní používa strom, pričom väčšina genetických algoritmov používa na reprezentáciu jednotlivcov v populácii (binárne) reťazce. Tieto reťazce (chromozómy) sú pasívne kódy pre riešenia daného problému. Pri genetickom programovaní sú však adaptujúce sa štruktúry (stromy) "aktívne", pretože reprezentujú program, ktorý môže byť vykonaný vo svojej aktuálnej forme. Dĺžka reťazcov pri genetických algoritmoch býva obvykle pevná, avšak veľkosť stromu sa pri GP mení. Každý počítačový program je vlastne kompozíciou funkcií z určitej množiny funkcií $\mathcal{F}$ a terminálov z množiny terminálov $\mathcal{T}$, ktoré reprezentujú vstupy pre hľadaný program. Každá funkcia z $\mathcal{F}$ by mala byť schopná akceptovať ako argumenty ľubovoľnú hodnotu a dátový typ, ktorý vracia iná funkcia z tejto množiny a ľubovoľnú hodnotu a dátový typ ktoréhokoľvek z terminálov. Teda zvolené množiny funkcií a terminálov by mali spĺňať vlastnosť uzavretosti, aby ľubovoľná kombinácia funkcií a terminálov dávala platný program. Funkcie (operácie) sú reprezentované vnútornými vrcholmi stromu a premenné alebo konštanty zase listami stromu. Takýto syntaktický strom možno interpretovať ako program, ktorý popisuje správanie sa určitého objektu (napr. umelého mravca) vo svojom prostredí alebo jednoducho ako nejakú funkciu. Táto práca sa zaoberá špeciálnym prípadom genetického programovania, v ktorom syntaktické stromy reprezentujú boolovské funkcie.

## Readov lineárny kód

Keďže sa budeme zaoberať funkciami reprezentovanými pomocou syntaktických stromov, potrebujeme zvoliť ich počítačovú reprezentáciu. Alternatívu ku klasickej "pointerovej" reprezentácii

stromu v procedurálnych jazykoch predstavuje Readov lineárny kód, ktorý umožňuje vyjadriť stromové štruktúry pomocou reťazcov. Readov kód pozostáva z postupnosti číslic, korešpondujúcich so stupňom koreňa alebo stupňom zníženým o jednotku pri ostatných vrcholoch stromu. Formálnejšie:

$$code(T(v)) =' q' + code(T_1(v_1)) + code(T_2(v_2)) + ... + code(T_n(v_n))) \tag{1}$$

kde $v$ je koreň stromu $T$, $q = deg(v)$ je stupeň vrchola $v$, $v_i$ sú synovia vrcholu $v$ a $T_i$ príslušné podstromy s koreňmi $v_i$. Pretože potrebujeme generovať náhodné stromy a tiež hľadať ich podstromy, je dôležité si uvedomiť, kedy postupnosť kladných číslic korešponduje s Readovým lineárnym kódom. Táto korešpodencia súvisí s pojmom *grafovej postupnosti*. Postupnosť kladných číslic $\alpha = (\alpha_1, \alpha_2, ..., \alpha_p)$ je grafová, ak platia nasledujúce podmienky:

$$\sum_{i=1}^{j} \alpha_i \geq j (j = 1, 2, \ldots, p-1) \tag{2}$$

$$\sum_{i=1}^{p} \alpha_i = p - 1 \tag{3}$$

Na základe týchto (ne)rovností možno vytvoriť jednoduchý algoritmus na generovanie stromov a testovanie, či je daná postupnosť kódom nejakého stromu. Syntaktické stromy možno reprezentovať jednoduchým rozšírením Readovho lineárneho kódu tak, že ku každému vrcholu priradíme funkčný alebo terminálny symbol. Podrobnejšie informácie možno nájsť napr. v [1].

## Symbolická regresia

Problém je daný nasledovne: máme funkciu definovanú regresnou tabuľkou a úlohou je nájsť syntaktický strom, ktorý aproximuje danú funkciu, teda minimalizuje rozdiel vypočítaných a požadovaných hodnôt. Takýto prístup k regresii nazval John Koza symbolickou regresiou. Riešenie sa tu hľadá na ohraničenej množine funkcií, ktorých funkcionálny tvar je pevný, pretože sú dané elementárne funkcie z ktorých môže byť výsledná funkcia zložená. V ďalšom sa budeme zaoberať boolovskými funkciami, ktoré sú zložené z elementárnych funkcií *and*, *or*, *xor* a *not*.

Majme regresnú tabuľku (tréningovú množinu) pozostávajúcu z $n$ dvojíc:

$$A = \{[x_i, y_i] | i = 1, 2, \ldots, n\} \tag{4}$$

kde $x_i$ reprezentujú vstupy a $y_i$ požadované výstupy. Cieľom symbolickej regresie je nájsť také riešenie, ktoré minimalizuje nasledovnú účelovú funkciu:

$$E(t) = \sum_{i=1}^{n} |t(x_i) - y_i| \tag{5}$$

kde $t$ reprezentuje syntaktický strom pre príslušnú funkciu. Riešenie má potom tvar:

$$t_{opt} = \arg\min_{t \in T} E(t) \tag{6}$$

## Implementácia

Základnými parametrami charakterizujúcimi beh genetického programovania (algoritmu) sú veľkosť populácie a maximálny počet generácií, ktoré vygeneruje. Je potrebné tiež určiť kritérium ukončenia a spôsob určenia výsledku. Často používanou metódou na určenie výsledku genetického programovania je tzv. *best-so-far individual*, čo je najlepšie riešenie získané počas celého behu programu. Počiatočná populácia je inicializovaná vygenerovaním náhodných jedincov. Existuje viacero možností ako generovať syntaktické stromy. Určiť pevnú hĺbku a vygenerovať kompletný

strom danej hĺbky, náhodne voliť vrcholy a rekurzívne vygenerovať nasledovníkov, alebo kombinovaná metóda predchádzajúcich s rôznym percentuálnym zastúpením oboch. Pre jednoduchosť som použil generovanie stromov nie podľa hĺbky, ale podľa počtu vrcholov.

Zaujímavou vlastnosťou genetického programovania je variabilita výsledných programov. Často je náročné alebo neprirodzené pokúšať sa dopredu špecifikovať alebo obmedzovať tvar a veľkosť eventuálneho riešenia. Navyše sa môže pri niektorých problémoch stať, že takéto obmedzenie zabráni nájdeniu riešenia. Pre efektívnu implementáciu generovania syntaktických stromov však potrebujeme poznať ich veľkosť. Takže dostávame ďalšie parametre pre algoritmus, konkrétne minimálny a maximálny počet vrcholov stromu, prípadne jeho hĺbku.

Na výber jedincov (riešení) z populácie, ktorý vstupujú do procesu reprodukcie sa používa operátor selekcie. Tento je založený na ich hodnote sily (fitness). Čím je fitness vyššia, tým väčšia je šanca, že dané riešenie bude použité aj v ďalšej generácii. Najčastejšie používanou metódou selekcie je tzv. *ruleta* (použitá aj v tejto práci). Každý člen populácie dostane časť ruletového kolesa podľa veľkosti fitness, takže čím väčšia fitness, tým väčšia je pravdepodobnosť výberu, ale popritom každý jedinec má šancu byť vybraný. Pri veľkých populáciách je však často nutné uspokojiť sa s jednoduchšími - časovo menej náročnými metódami. Jedna z nich sa nazýva *overselection*. Populácia sa rozdelí na dve skupiny (ľubovoľne alebo podľa fitness) a výber sa potom uskutočňuje s rôznymi pravdepodobnosťami z jednej alebo z druhej skupiny ruletovým algoritmom. Koza používa v štandardnej formulácii pravdepodobnosť 80% na výber z prvej skupiny a 20% z druhej skupiny. Ďalšou metódou je tzv. *tournament selection*, pri ktorej sa z populácie náhodne vyberie $n$ jedincov a z nich sa potom vyberie najlepší podľa fitness.

Kríženie je v genetickom programovaní hlavným operátorom pre rekombináciu starých na nové, potencionálne lepšie riešenia. Majme dvoch členov populácie $A$ a $B$. V oboch náhodne vyberieme krížiace body (vrcholy) $n_A$ a $n_B$. Kríženie potom pozostáva z výmeny podstromov začínajúcich v $n_A$ a $n_B$. Pri mutácii najskôr vyberieme mutačný bod (vrchol) a príslušný podstrom nahradíme náhodne vygenerovaným podstromom. Pri jeho generácii sa ponúka viacero možností. Prvou je vygenerovať podstrom s rovnakým počtom vrcholov, druhou možnosťou je vygenerovať podstrom s rovnakou hĺbkou ako pôvodný podstrom alebo možno vygenerovať podstrom ľubovoľnej veľkosti s následnou kontrolou, či výsledný strom spĺňa obmedzenie veľkosti alebo hĺbky. Pri výbere mutačného bodu je možné určiť pravdepodobnosti, ktoré charakterizujú jeho pozíciu v pôvodnom strome. Toto môže byť niekdy výhodné, pretože parameter *pravdepodobnosť mutácie* v genetickom programovaní len určuje pravdepodobnosť, či sa daný strom bude mutovať alebo nie, ale neovplyvňuje samotný proces mutácie ako pri genetickom algoritme pri mutácii chromozómu.

Klasickú účelovú funkciu zo syntaktickej regresie rozšírime o pokutový člen $\alpha$, ktorý bude zvýhodňovať riešenia s menším počtom vrcholov

$$E(t) = \sum_{i=1}^{n} |t(x_i) - y_i| + \alpha|t| \tag{7}$$

Pseudokód pre algoritmus použitý na implementáciu genetického programovania je na obrázku 1.

```
P = random_population();
while( !stop_criterion() ) {
  evaluate_population_with_fitness();
  Q = {};
  while( |Q| < |P| ) {
    ch1 = Select( P );
    ch2 = Select( P );
    if( random < P_cross )
      (ch1', ch2') = Crossover( ch1, ch2 );
      if( random < P_mut )
        ch1' = Mutation( ch1' );
      if( random < P_mut )
        ch2' = Mutation( ch2' );
    else
      (ch1', ch2') = (ch1, ch2);
    Q = Q + {ch1', ch2'};
  }
  P = Q;
}
```

Figure 1: Pseudokód pre genetické programovanie

## Testovanie a výsledky

Na otestovanie funkčnosti algoritmu boli použité boolovské funkcie pre paritu a symetriu vstupného vektora. Obe tieto funkcie majú pre štyri vstupy optimálne riešenia veľkosti osem vrcholov, preto bolo obmedzenie veľkosti generovaných syntaktických stromov nastavené na rozsah $5-20$, pri viacerých vstupoch na $5-30$ vrcholov. Na obrázkoch 2 a 3 sú zobrazené postupnosti riešení vygenerované genetickým programovaním pre paritu resp. symetriu binárnych vektorov. Použité parametre algoritmu boli získané experimentálne na základe niekoľkých pozorovaní. Pre paritu: veľkosť populácie $Pop = 100$, pravdepodobnosť kríženia $Pcross = 0.9$, pravdepodobnosť mutácie $Pmut = 0.5$, pokutový člen $\alpha = 0.01$; pre symetriu: $Pop = 500$, $Pcross = 0.9$, $Pmut = 0.75$, $\alpha = 0.01$.

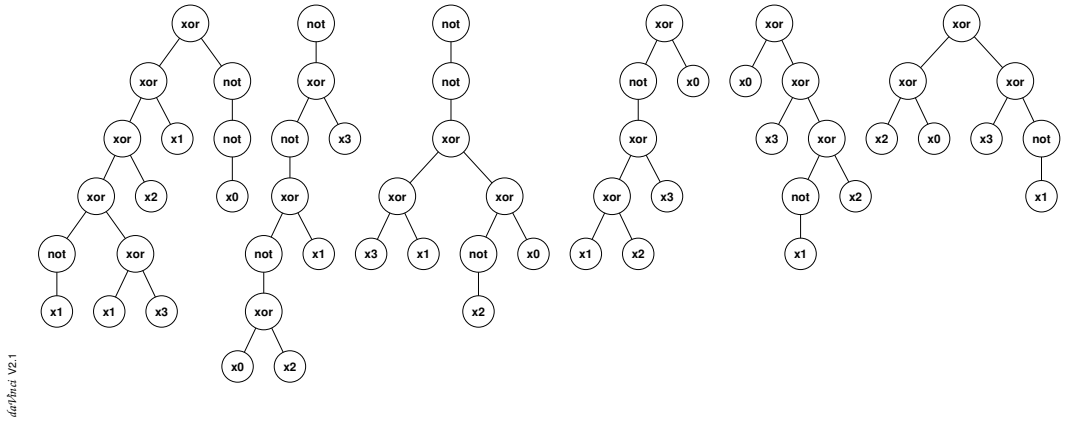


Figure 2: Syntaktické stromy pre paritu binárnych vektorov dĺžky n=4. Riešenia boli získané postupne v epochách: 10,50,110,370,430 a 510

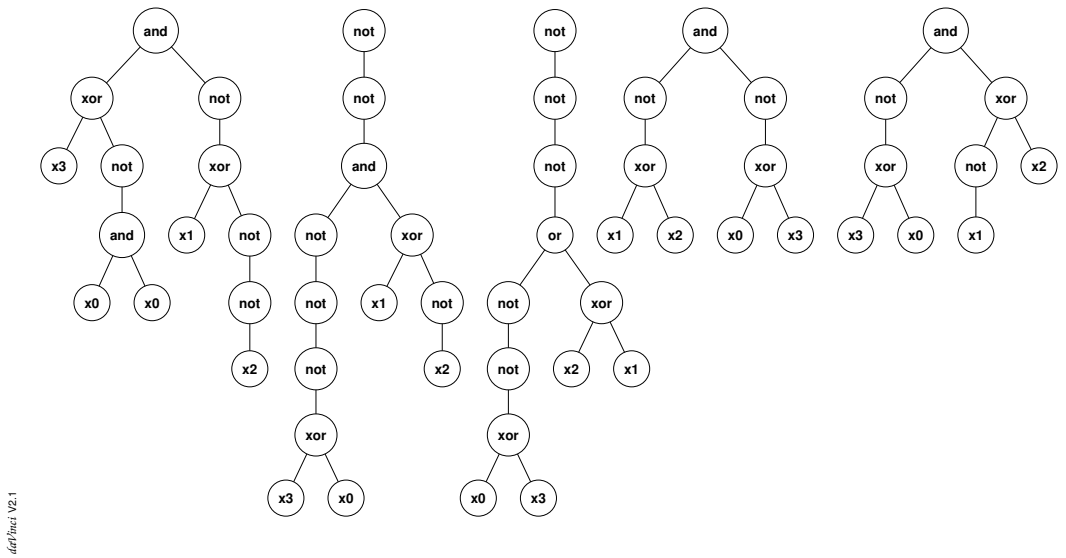


Figure 3: Syntaktické stromy pre symetriu binárnych vektorov dĺžky n=4. Riešenia boli získané postupne v epochách: 170,260,490,500 a 640

Na obrázkoch 4 a 5 sú grafy zobrazujúce "vývoj" najlepšej resp. priemernej hodnoty účelovej funkcie počas evolúcie. Pokutový člen $\alpha$ bol nastavený vzhľadom na veľkosť generovaných syntaktických stromov tak, aby ovplyvňoval hodnoty účelovej funkcie len za desatinnou čiarkou. Celočíselná hodnota vyjadruje počet chýb príslušnej aproximácie podľa trénovacej monožiny. V prvom prípade (obrázok 4) bola pravdepodobnosť kríženia malá: $P_{cross} = 0.2$, v druhom prípade (obrázok 5) veľká: $P_{cross} = 0.7$. Pravdepodobnosť mutácie bola v oboch prípadoch rovnaká: $P_{mut} = 0.9$. Na prvý pohľad je vidieť rozdiel v charaktere uvedených grafov. V prvom prípade sa najlepšia účelová hodnota "postupne znižuje", avšak v druhom prípade odrazu klesne na interval $0 - 1$ reprezentujúci správne riešenie. Priemerná hodnota účelovej funkcie v prvom prípade rovnomerne klesá, pričom v druhom prípade je jej priebeh chvíľu klesajúci, potom zase rastúci. Tento rozdiel by sa snáď dal interpretovať tak, že keď sa používa kríženie zriedkavo, postupne sa zlepšujú riešenia obsiahnuté v populácii. Avšak ak sa kríženie používa často, v populácii sa paralelne vytvára viacero riešení a nakoniec jedno z nich preváži.

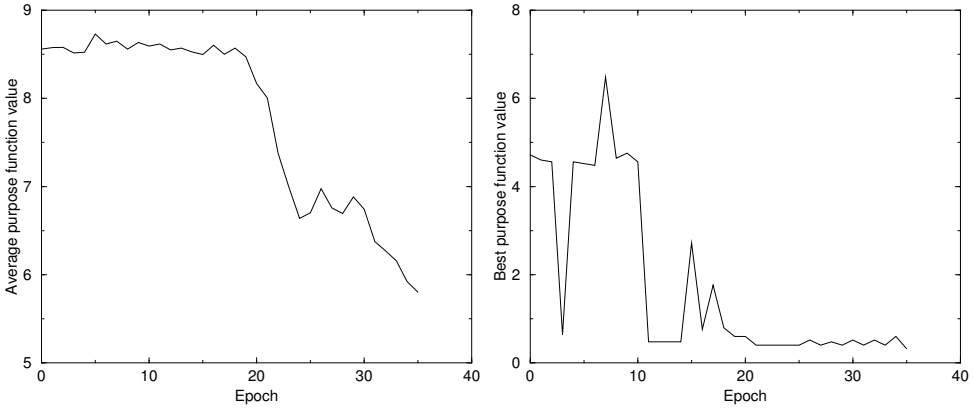


Figure 4: Priebeh účelovej funkcie počas evolúcie. $P_{cross} = 0.2, P_{mut} = 0.9$

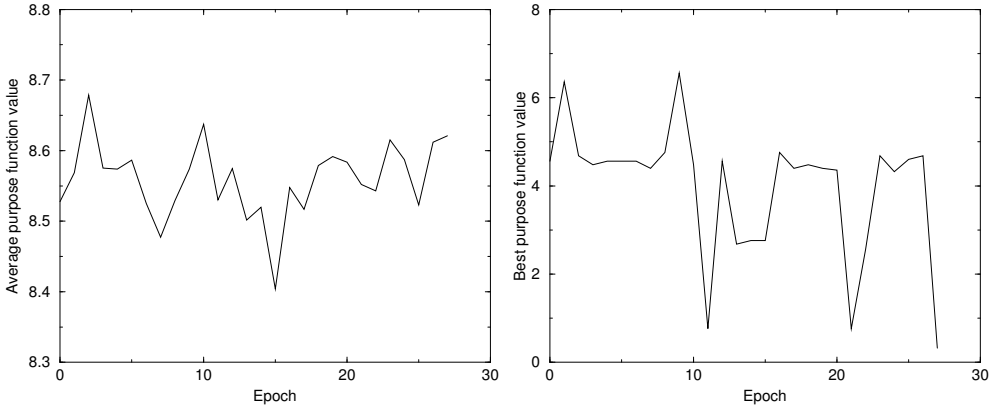


Figure 5: Priebeh účelovej funkcie počas evolúcie. $P_{cross} = 0.7, P_{mut} = 0.9$

V ďalších častiach kapitoly sa pokúsime podrobnejšie preskúmať vplyv jednotlivých parametrov algoritmu na jeho celkovú funkčnosť a efektívnosť. Používa sa funkcia parity pre $n = 4$. Najskôr sa pokúsime zistiť, čo spôsobí zmena veľkosti populácie. Grafy na obrázku 6 zobrazujú závislosť počtu iterácií algoritmu potrebných na dosiahnutie optimálneho riešenia vzhľadom na veľkosť populácie. Zobrazené hodnoty boli získané ako priemer zo 100 nezávislých pokusov, pričom algoritmus bol ukončený ak našiel optimálne riešenie (vzhľadom na veľkosť syntaktického stromu), alebo ak sa za posledných 500 iterácií nezlepšilo globálne riešenie. Uvedený spôsob "štatistiky" bol aplikovaný aj

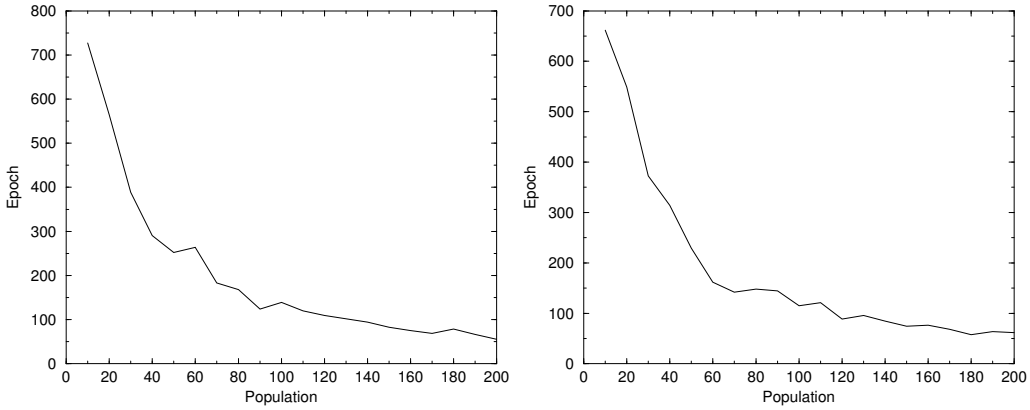


Figure 6: Počet epoch potrebných na dosiahnutie optimálneho riešenia v závislosti od veľkosti populácie pre obe verzie algoritmu. Funkcia: parita pre $n = 4$; Parametre algoritmu: $P_{cross} = 0.8, P_{mut} = 0.5, \alpha = 0.01$

pri pokusoch v ďalších častiach tejto práce. Použité boli dve verzie algoritmu. Prvá implementácia (pseudokód na obrázku 1) aplikuje operátor mutácie len na potomkov získaných krížením a títo sú následne zaradení do novej populácie. Druhá verzia je rozšírením pôvodnej o mutáciu všetkých nových členov populácie, teda aj tých ktorí boli pôvodne len kopírovaní. Tabuľka na obrázku 7 a tiež charakter uvedených grafov ukazujú, že takéto rozšírenie algoritmu nie je na škodu, dokonca mu môže pomôcť. Je zrejmé, že pri pôvodnej verzii algoritmu sa so znižovaním pravdepodobnosti kríženia zníži aj efektívnosť, pretože sa nepriamo znížilo aj percento jedincov, ktoré sa mutujú. Pri rozšírenej verzii už tento rozdiel nie je a ako vidieť z nameraných hodnôt, dokonca algoritmus bez kríženia dosiahol slušné výsledky.

Vráťme sa však k pôvodnému zámeru, a to zistiť vplyv veľkosti populácie na efektívnosť symbolickej regresie. Z grafov je zrejmé, že zvyšovaním veľkosti populácie sa efektivita algoritmu zvyšuje (čo samozrejme nie je prekvapivé zistenie), pre danú funkciu (parita s $n = 4$) od 100-člennej populácie potrebuje algoritmus v priemere 100 iterácií. Pravdepodobnosť nájdenia optimálneho riešenia bola v oboch prípadoch 100%-ná pre populáciu väčšiu ako 40 členov a pravdepodobnosť nájdenia ľubovoľného riešenia bola 100%-ná už od 20 člennej populácie.

| | pôvodná verzia | | | rozšírená verzia | | |
|---|---|---|---|---|---|---|
| veľkosť populácie | 200 | 100 | 50 | 200 | 100 | 50 |
| $P_{cross} = 1, P_{mut} = 0.5$ | 61 | 124 | 250 | 54 | 121 | 259 |
| $P_{cross} = 0.7, P_{mut} = 0.5$ | 86 | 171 | 350 | 53 | 128 | 258 |
| $P_{cross} = 0.5, P_{mut} = 0.5$ | 119 | 287 | 492 | 75 | 135 | 243 |
| $P_{cross} = 0, P_{mut} = 1$ | | | | 64 | 185 | 237 |

Figure 7: Porovnanie efektívnosti dvoch verzií algoritmu. Prvá používa mutáciu len na potomkov získaných krížením, pričom druhá verzia mutuje všetkých jedincov, ktorí budú zaradení do novej populácie. Hodnoty vyjadrujú priemerný počet itrerácií (zo 100 pokusov) potrebných na dosiahnutie optimálneho riešenia.

Ďalej sa zameriame na význam kríženia pri symbolickej regresii. Pokúsime sa zistiť, či je kríženie naozaj dôležité alebo vystačíme len s mutáciou. Nastavením pravdepodobnosti kríženia na nulu ($P_{cross} = 0$) v rozšírenej verzii algoritmu, dostávame algoritmus používajúci len mutáciu. Prvé výsledky boli prezentované už v tabuľke 7, kde je vidieť, že pri použití rozšírenej verzie algo-

ritmu dostávame aj napriek znižujúcej sa pravdepodobnosti kríženia porovnateľné výsledky. Na obrázku 8 je graf zobrazujúci závislosť počtu potrebných iterácií na dosiahnutie optimálneho riešenia vzhľadom na pravdepodobnosť kríženia a graf zobrazujúci pravdepodobnosť nájdenia optimálneho riešenia vzhľadom na pravdepodobnosť kríženia. Bola použitá populácia veľkosti 100 a pravdepodobnosť mutácie 50%. Z grafov vidieť, že kríženie v našom prípade nielenže zvyšuje efektivitu pri hľadaní optimálneho riešenia, ale tiež výrazne ovplyvňuje spoľahlivosť algoritmu z pohľadu hľadania optimálneho riešenia. Užitočnou informáciou, ktorá sa dá z grafov vyčítať je hodnota "optimálnej" pravdepodobnosti kríženia (asi $60 - 80\%$).

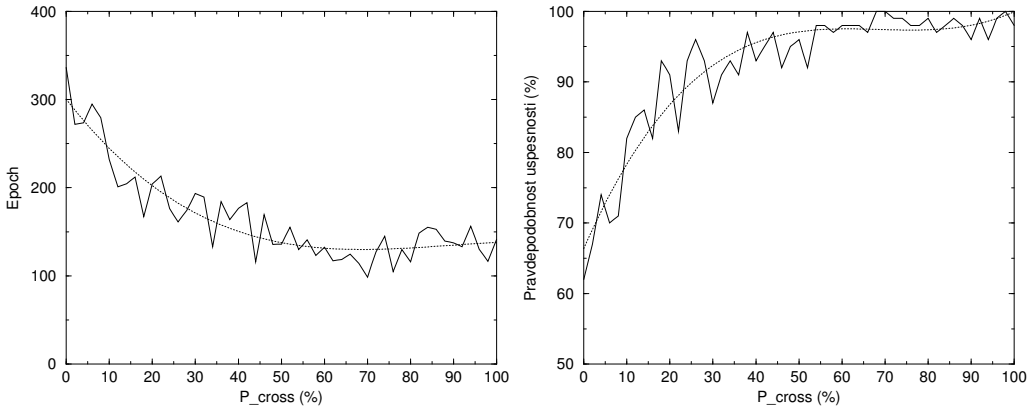


Figure 8: Závislosť počtu potrebných iterácií algoritmu na dosiahnutie optimálneho riešenia vzhľadom na pravdepodobnosť kríženia. Funkcia: parita pre $n = 4$; Parametre algoritmu: $Pop = 100, P_{mut} = 0.5, \alpha = 0.05$

Ďalej som sa snažil zistiť, ako ovplyvňuje funkčnosť a efektivitu algoritmu pravdepodobnosť mutácie. Graf na obrázku 9 zobrazuje počet iterácií potrebných na dosiahnutie optimálneho riešenia pri zmene pravdepodobnosti mutácie od $0 - 100\%$. Testovacia funkcia bola parita pre štyri vstupy a parametre algoritmu nasledovné: $Pop = 50, P_{cross} = 0.8$. Pravdepodobnosť úspešnosti nájdenia optimálneho riešenia bola pri všetkých hodnotách pravdepodobnosti mutácie nad 95%, takže algoritmus fungoval "dosť dobre". Napriek môjmu očakávaniu, graf neobsahuje žiaden výrazný extrém, takže sa nedá jednoznačne povedať, aká pravdepodobnosť mutácie je najvhodnejšia, preto som uskutočnil ďalší pokus. Tentoraz som použil funkcu parity pre 5 vstupov s nasledovnými parametrami algoritmu: $Pop = 100, P_{cross} = 0.7$. Výsledky sú v grafoch na obrázku 10. Ako vidieť z grafov, pri týchto parametroch mal algoritmus väčšie problémy hľadať optimálne riešenie, ale zase je tu lepšie vidieť, čo spôsobí zmena pravdepodobnosti mutácie. Najrozumnejšie zrejme budú hodnoty okolo 50%.

Pre boolovské funkcie s viac ako štvorma vstupmi sú potrebné 300 a viac členné populácie, takže testovaný algoritmus je časovo príliš náročný. Preto som sa pokúsil implementovať do genetického programovania jednoduchý elitizmus (do ďalšej generácie sa dostane $x$ kópií najlepšieho jedinca z aktuálnej populácie) s nádejou, že takto upravený algoritmus dokáže v rozumnejšom čase hľadať uspokojivé riešenia. Získané výsledky sú uvedené v tabuľkách 11, 12 a 13.

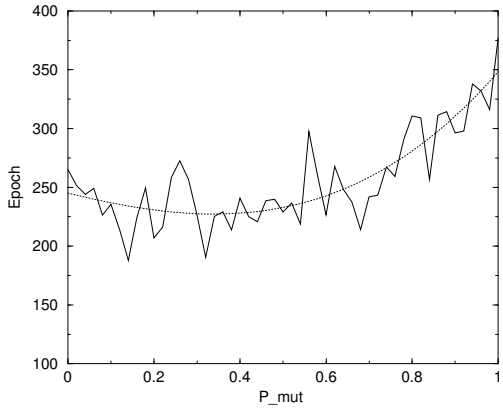


Figure 9: Počet iterácií potrebných na dosiahnutie optimálneho riešenia v závislosti na pravdepodobnosti mutácie. Funkcia: parita pre $n = 4$; Parametre algoritmu: $Pop = 50, P_{cross} = 0.8, \alpha = 0.01$

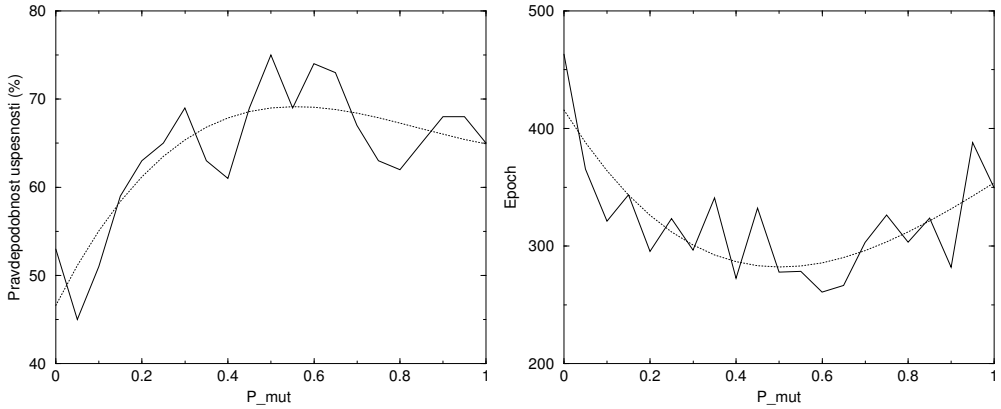


Figure 10: Pravdepodobnosť nájdenia optimálneho riešenia a počet iterácií potrebných na dosiahnutie optimálneho riešenia v závislosti na pravdepodobnosti mutácie. Funkcia: parita pre $n = 5$; Parametre algoritmu: $Pop = 100, P_{cross} = 0.7, \alpha = 0.05$

# Literatúra

1. Vladimír Kvasnička, Jiří Pospíchal, Martin Pelikán, *Read's Linear Codes And Evolutionary Computation Over Populations Of Rooted Trees*

2. Vladimír Kvasnička, Jiří Pospíchal, *Prednáška "Evolučné algoritmy"*

| počet vstupov | veľkosť riešenia | populácia | elitizmus | | počet epoch | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 5 | $\leq 12$ | 200 | 20 | | 105 | 220 | 110 | 70 | 85 |
| 6 | $\leq 20$ | 300 | 20 | | 176 | 113 | 15 | 502 | 422 |
| 7 | $\leq 25$ | 400 | 20 | | 67 | 214 | 82 | 141 | 77 |

Figure 11: Počet epoch potrebných na dosiahnutie "rozumného" riešenia pre fukncIu parity s rôznym počtom vstupov. Parametre: $P_{cross} = 0.7, P_{mut} = 0.8$. Uvedené hodnoty sú výsledkom samostatných pokusov (nie sú to priemerné hodnoty)

| elitizmus | 0 | 2 | 4 | 6 | 8 | 10 |
|---|---|---|---|---|---|---|
| počet epoch | 459 | 402 | 309 | 422 | 401 | 392 |
| pravdepodobnosť úspešnosti | 52 | 79 | 80 | 81 | 82 | 84 |

Figure 12: Parita pre $n = 5$; $Pop = 100, P_{cross} = 0.7, P_{mut} = 0.8$ Počet epoch a pravdepodobnosť dosiahnutia riešenia veľkosti najviac 12 vrcholov. Uvedené hodnoty sú priemerom zo 100 nezávislých pokusov

| elitizmus | 0 | 2 | 4 | 6 | 8 | 10 |
|---|---|---|---|---|---|---|
| počet epoch | 98 | 58 | 54 | 49 | 48 | 59 |

Figure 13: Parita pre $n = 5$; $Pop = 200, P_{cross} = 0.7, P_{mut} = 0.8$ Počet epoch na dosiahnutie ľubovoľného riešenia. Uvedené hodnoty sú priemerom zo 100 nezávislých pokusov