VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

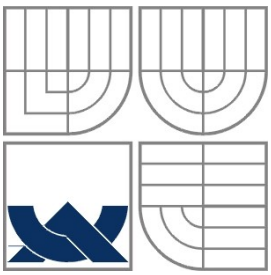
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

VLIV MAPOVÁNÍ KANDIDÁTNÍHO ŘEŠENÍ NA EFEKTIVITU EVOLUČNÍHO ALGORITMU

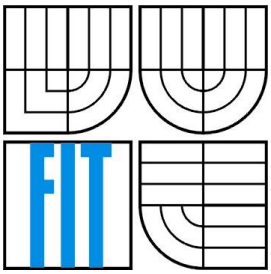
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE JIŘÍ HRBÁČEK
AUTHOR

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

VLIV MAPOVÁNÍ KANDIDÁTNÍHO ŘEŠENÍ NA EFEKTIVITU EVOLUČNÍHO ALGORITMU

THE IMPACT OF CANDIDATE SOLUTION MAPPINGS ON EVOLUTIONARY ALGORITHM EFFICIENCY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — Jiří Hrbáček
AUTHOR

VEDOUCÍ PRÁCE — Ing. Jan Křivánek
SUPERVISOR

BRNO 2010

Abstrakt

Náplní předložené práce je sumarizace znalostí v oblasti teorie mapování kandidátního řešení, jejich analýza a aplikace na evoluční algoritmy. Práce podává přehled evolučních algoritmů, jejich klasifikaci a použití. Cílem práce je propojení získaných znalostí z oblasti evolučních algoritmů a mapování kandidátního řešení a vytvořit systém, který bude úspěšně demonstrovat vliv mapování na efektivitu evolučních algoritmů.

Abstract

The Concern of the present study is summarizing knowledges in the theory of mapping candidate solutions , analysis and application of evolutionary algorithms. The study provides summary of the evolutionary algorithms, classification and application. The target of the study is links gained knowledge from sectionS of ; evolutionary algorithms, mapping candidate solutions and creations of a system that will demonstrate and influence mapping the efficiency of the evolutionary algorithms succesfully.

Klíčová slova

evoluční algoritmy, genetické algoritmy, genetické programování, evoluční strategie, mapování, genotyp, fenotyp, kódování

Keywords

evolutionary algorithms, genetic algorithms, genetic programming, evolutionary strategy, mapping, genotype, fenotype, representations

Citace

Jiří Hrbáček: Vliv mapování kandidátního řešení na efektivitu evolučního algoritmu, bakalářská práce, Brno, FIT VUT v Brně, 2010

Vliv mapování kandidátního řešení na efektivitu evolučního algoritmu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Jana Křivánka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jiří Hrbáček
19.5.2010

Poděkování

Rád bych poděkoval Ing. Janu Křivánkovi za laskavé vedení práce, věnovaný čas a podnětné náměty na zaměření a cíl práce.

Obsah

1 Úvod

Evoluční algoritmy jsou numerické algoritmy, které nalézají uplatnění především pro řešení složitých optimalizačních problémů. Vycházejí ze základních principů Darwinovy a Mendelovy teorie evoluce. Jsou nadtřídou matematických postupů, které jsou založeny na evolučních procesech v přírodě. Hlavní myšlenkou evoluce je předávání rodičovského genomu novým potomkům a následné obnovení populace. Pracují s množinou možných řešení, které se každou generací postupně vylepšuje tím, že eliminují horší řešení a preferují se řešení lepší. Správně zvolené mapování kandidátního řešení, by mělo vést k přesnějším řešením evolučního algoritmu.

Cílem práce je implementace demonstrativního evolučního algoritmu a navržení právě takových mapovacích funkcí, které nám umožní pro konkrétní problémy dosahovat přesnějších výsledků. Dosažené výsledky následně zpracovat a vyhodnotit úspěšnost návrhu mapování.

V druhé kapitole, *Evoluční algoritmy,* jsou základní informace o rozdělení evolučních algoritmů do čtyř skupin. Každá skupina je základně charakterizována, jsou zde uvedeny klady a zápory , vhodnosti použití jednotlivých evolučních algoritmů.

Ve třetí kapitole, *Mapovací systémy evolučních algoritmů,* se nachází stručný úvod do problematiky mapování. Podrobnější definování čtyř základních přístupů mapování, definování kódování a popis základních vlastností kódování.

Vlastní *Návrh systému mapování* následuje ve čtvrté kapitole, kde jsou definovány jednotlivé kroky návrhu systému, který používá optimalizační techniky evolučního algoritmu, k hledání co nejpřesnějších výsledků. V této kapitole je také uvedeno zhodnocení dosažených výsledků. Průběh evolučního algoritmu na testovacích funkcích bez mapování a s různým mapováním. Porovnání a vyhodnocení této práce a posouzení zkoumaného problému tj. vliv mapování kandidátního řešení na efektivitu evolučního algoritmu.

Shrnutí přínosu práce, její úspěšnosti a splnění požadavků a myšlenek, lze nalést v poslední kapitole *Závěr.*

2 Evoluční algoritmy

Evoluční algoritmy jsou výpočetní prohledávací modely, které jsou založeny na evoluci v živé přírodě. Patří do třídy algoritmů umělé inteligence (soft computing). Většinou nepracují s jednotlivým řešením, nýbrž celou množinou možných řešení. Používají principy biologické evoluce - reprodukce, křížení, mutace a selekce. Pomoci těchto principů postupně vylepšují řešení a eliminují řešení horší. Klíčovým se stává vhodné zakódování řešení do genotypu (řetězce znaků) nad kterým probíhá vlastní reprodukční proces. Evoluční algoritmy jsou typické svojí robustností a jsou daleko robustnější než ostatní prohledávací algoritmy. [1][2]

2.1 Uplatnění evolučních algoritmů

Evoluční algoritmy nalézají uplatnění především pro řešení složitých optimalizačních problémů zejména tam, kde klasické metody selhávají nebo jsou nedostačující. Řešení evolučními algoritmy nezávisí na gradientních informacích, proto jsou vhodné pro řešení problémů, kde nám nejsou takové informace dostupné. Mohou řešit i problémy, které nejsou přímo vyjádřeny přesnou účelovou funkcí. Jde o úlohy s omezujícími podmínkami a různými extrémy a kriterií. Uplatnění mají především v oblastech umělé inteligence a inženýrského návrhu[1][3]:

- Numerická, kombinatorická optimalizace
- Modelování a identifikace modelu
- Plánování a řízení
- Inženýrský návrh
- Dolování dat
- Strojové učení a umělá inteligence

Dále se používají pro numerickou a kombinatorickou optimalizaci, při návrhu obvodů, plánování výroby, strojové učení, při tvorbě ekologických, sociálních a ekonomických modelů atd. Jejich použití je efektivní zejména v úlohách, kde je prostor hledání příliš rozsáhlý, kde nelze provést matematickou analýzu problému a kde tradiční úlohy selhávají.
Mezi hlavní nevýhody evolučních algoritmů patří především nejistota správnosti řešení. Kvalita řešení lze hodnotit pouze intuitivně, nemáme možnost otestovat, zda jsme dosáhli globálního optima. Další nevýhodou je veliká časová náročnost a s rostoucí rozsáhlostí úlohy roste i vzdálenost od optima.

2.2 Dělení evolučních algoritmů

Evoluční algoritmy dělíme na několik následujících typů. Dělení provádíme především podle implementace a podle zaměření na konkrétní třídu problému[2]:

- **Genetické algoritmy –** nejrozšířenější typ evolučních algoritmů. Řešení je zakódováno do binárního řetězce. Dále prováděním rekombinace a mutace na jednotlivé generace kandidátních řešení se postupně dostáváme ke globálnímu optimu.
- **Genetické programování –** jedná se o vytváření počítačových programů za využití metod biologické evoluce. Dochází k postupnému zlepšování populace počítačových programů – strojové učení.[4]
- **Evoluční strategie –** kandidátní řešení se kóduje podobně jako u genetických algoritmů do řetězce čísel, ovšem do řetězce čísel reálných. Rodiče produkují potomky a dochází k soupeření mezi rodiči a potomky.
- **Evoluční programování –** podobné genetickému programování s tím rozdílem, že struktura programu je pevně daná, nedochází k zlepšování populace programu, vyvíjí se pouze číselné parametry. Typickým operátorem je mutace.

2.2.1 Genetické algoritmy

Genetické algoritmy jsou nejčastější a také nejznámější skupinou evolučních algoritmů. Jedná se o účinný prohledávací algoritmus pro adaptivní systémy umělé inteligence.[1] Definujeme operátor *křížení* (crossover operátor), který je hlavní rozlišovací znak genetických algoritmů a operátor *inverze*. Genetické algoritmy našly své hlavní uplatnění v optimalizačních úlohách hledáním globálního extrému vícerozměrných funkcí. Používá se v podobě *jednoduchého/kanonického genetického algoritmu* (simple genethic algorithm – SGA). Základní znaky jsou [1],[2],[3],[5]:

- *Reprezentace - chromozóm* (jedinec, kandidátní řešení) je reprezentován binárním vektorem délky *n*

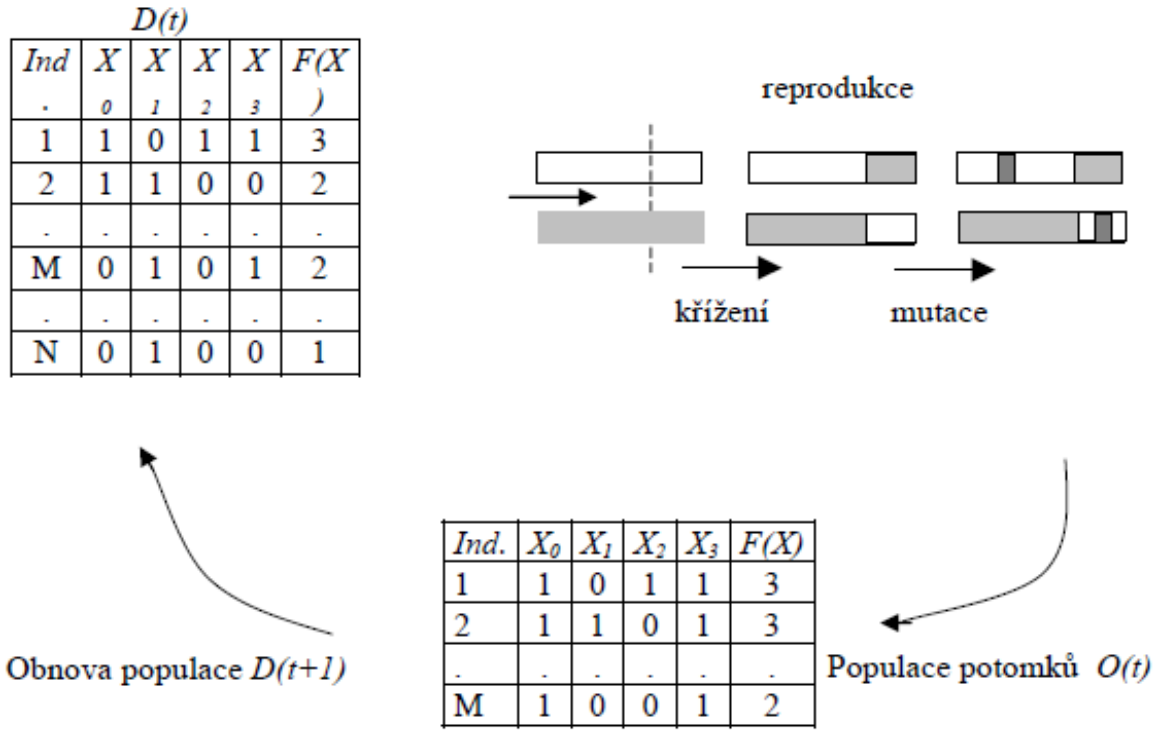
$$x \in \{0,1\}^n \tag{2.1}$$

- Na základě hodnoty *fitness* funkce f (2.2) dojde k výběru rodičů z populace jedinců. Pravděpodobnost uváznutí v lokálním minimu a také rychlost konvergence této funkce je pevně závislá na způsobu výběru rodičů.

$$f : \{0,1\}^n \to R \tag{2.2}$$

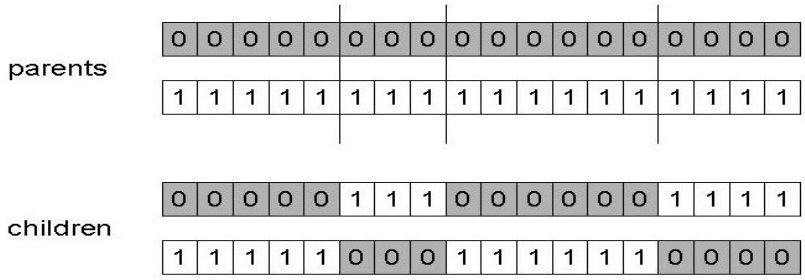
Vývojový diagram genetického algoritmu viz obr. 2.1 [1]*:*

Náhodně se vygeneruje počáteční populace. Určí se hodnota fitness funkce. V procesu reprodukce se náhodně vyberou dvojice rodičů pro křížení a mutaci. Část potomků se potom použije pro nahrazení svých rodičů v původní populaci *D(t)* a takto získáme novou populaci *D(t+1)*. Ukončení procesu je buď stanoveno nebo pokud dojde ke stagnaci vývoje.



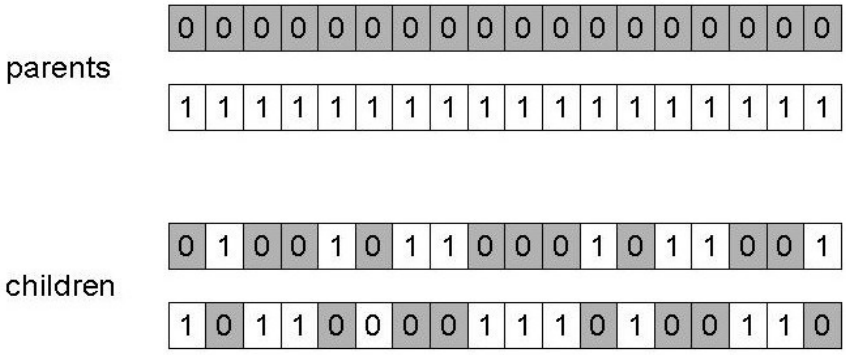
Obr. 2.1: Vývojový diagram genetického algoritmu.

- *Křížení* – (crossing-over) operátor, který je pro genetické algoritmy tím nejtypičtějším. Základem křížení je náhodný výběr dvojice jednotlivců a mezi touto dvojicí dochází k rekombinaci výměně genů.(výměně genové informace). Operátor křížení je prováděn s pravděpodobností p_c. Často se tato operace neprovádí se 100% pravděpodobností.
  - *Jedno a vícebodové křížení* – nejjednodušší způsob rekombinace. Dochází k výměně části genetických informací nebo výměně více úseků chromozómů obou rodičů viz obr.2.2. Někteří potomci mohou poté disponovat zvýšením fitness funkce.



Obr 2.2: Ukázka tříbodového křížení.[1]

- *Uniformní křížení* – při uniformním křížení dochází k výměně nulového a jedničkového chromozómu. Výměna genů se provádí s pravděpodobností p_u. Uniformní křížení přináší potřebnou různorodost do populace, použití je vhodné při řešení složitějších funkcí. Ale bylo také zavrhováno, pro přílišné rozvracení kódu viz obr 2.3.



Obr 2.3: Uniformní křížení. [1]

- *Selekce* – nebo-li výběr, vytváří ze staré populace populaci novou. Nejběžnější výběr je náhodný výběr pomoci tzv. rulety (roulette wheel selection), pravděpodobnost výběru jednotlivce je závislá na jeho fitness. Musí se dodržet určitá pravidla, především upřednostnit jedince s vyšší hodnotou fitness a také vybrat dostatečně různorodou novou populaci. Pokud algoritmus nesplňuje některý z požadavků, dochází k pomalé konvergenci algoritmu nebo naopak předčasné konvergenci. Rychlost konvergence algoritmu je tudíž závislá na *Selekční intenzitě/tlaku* (2.3). Čím vyšší je selekční intenzita, tím více jedinců s vyšší hodnotou fitness nová populace obsahuje. Nastává problém předčasné konvergence. Tuto situaci řeší *takeover time*, kdy je nová populace tvořená jedinci s nejlepším chromozómem za neúčasti rekombinačního a mutačního operátoru.

$$I = \frac{\overline{M^*} - \overline{M}}{\overline{\sigma}} \qquad (2.3)$$

kde $\overline{M}$ označuje průměrnou hodnotu fitness v populaci před selekcí, $\overline{M*}$ je průměrná hodnota po selekci a $\overline{\sigma}$ je odchylka (rozptyl) fitness hodnot před selekcí.

Nejčastěji používané selekční algoritmy:[1]

- proporcionální selekce (roulette wheel selection)
- selekce zbytková (truncation)
- lineární uspořádání (ranking)
- exponenciální uspořádání
- turnajová selekce

- *Mutace* – velmi významný operátor. Pro genetické algoritmy je mutace zdrojem nových informací. Příliš vysoká pravděpodobnost mutace p_m může mít fatální následky naopak nízká nepřinese dostatečný počet nových informací. Nejběžněji se používá bitová negace náhodně vygenerovaného bitu. Pravděpodobnost p_m vybrání bitu pro negaci bývá v intervalu *<1/bitů v generaci); 1/(bitů v chromozómovém řetězci)>*.

- *Obnova populace – generativní* obnova populace, kdy je celá původní populace nahrazena svými potomky a *částečná* obnova populace (*steady state*), pouze jeden potomek nahradí nejslabšího původního jedince. Nejčastěji se používá obou variant, kdy je až 50% rodičů nahrazeno svými potomky.

Hlavní nedostatky genetických algoritmů [5]:

- *Omezenost prostoru kandidátních řešení* – podle vzorce (2.1) vyplývá, že počet kandidátních řešení je 2^n, kde *n* je délka řetězce.
- *Proporcionální výběr jedinců* – dochází k výběru mezi jedinci s podobnou hodnotou fitness, tudíž méně různorodých jedinců.

2.2.2 Genetické programování

Cílem této metody je generovat celé programy v určitém programovacím jazyce, které řeší nějakou danou úlohu. Dochází zde k prohledávání v prostoru řešení, který se v průběhu evoluce mění, ne pouze k paralelnímu prohledávání statického stavového prostoru. Hlavní rozdíl mezi genetickým programováním a genetickým algoritmem je ten, že genetický algoritmus kóduje problém do řetězce pevné délky, na rozdíl genetické programování pracuje se *spustitelnými strukturami* proměnlivé délky. Nejčastější aplikací používanou v genetickém programování je tzv. s*ymbolická regrese.*[1] Základní znaky genetického programování jsou[1],[2],[5]:

- *Reprezentace* – genetické programování pracuje se stromovými strukturami, tyto struktury představují spustitelné kódy a mají typicky proměnnou délku. Stromové výrazy vycházejí z množiny terminálů *T* a množiny funkcí *F*. Definice množiny funkcí[1]:
  1. Každé $t \in T$ je korektní výraz.
  2. $f(e_1, e_2, \ldots, e_n)$ je korektní výraz právě tehdy když $f \in F \ \wedge$ $arity(f) = n \ \wedge \ e_i$ je korektní výraz $\forall i \in \{1,2 \ldots, n\}$ (2.4)
  3. Žádná jiná forma korektních výrazů neexistuje.

- *Genetické operátory* – kromě běžných operátorů (křížení, mutace atd.) existují i pokročilejší operátory, které generují samotné programy.

- K zjištění *fitness* hodnoty se provede část kandidátního programu s definovanou množinou vstupů a poté se vyhodnocují získané výsledky.

- Výběr jedinců probíhá na základě hodnoty *fitness* funkce, obor hodnot jsou všechna R.

Při genetickém programování musíme nejprve definovat [1]:

- Množinu terminálů
- Množinu funkcí
- Způsob výpočtu fitness
- Parametry genetického programování
- Určení výsledku a ukončení evoluce

- *Množina terminálů* – definuje vstupy do programu – tedy konstanty, funkce bez argumentů a proměnné. Při zpracovávání terminály vrací programu určité dílčí hodnoty a právě proměnné hrají důležitou roli v procesu učení, kdy dodávají do programu data z tzv. t*rénovací množiny.*

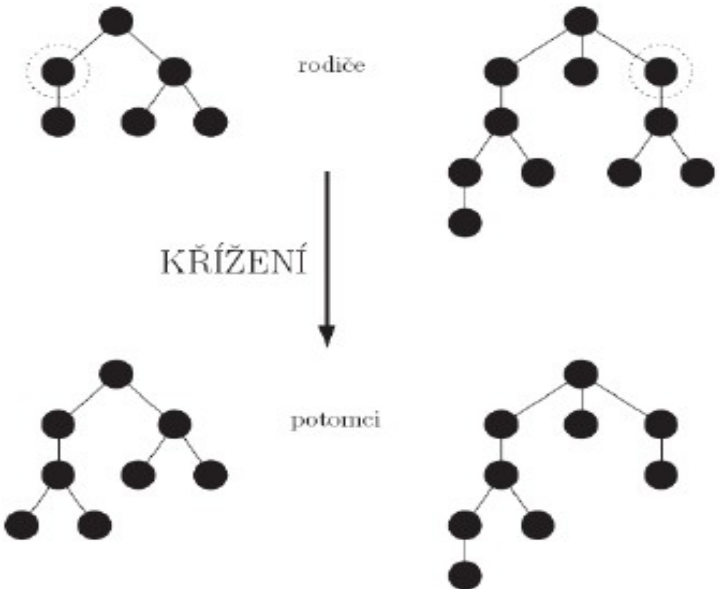
- *Množina funkcí* - velikost množiny je prakticky neomezená, ovšem bereme ohled na její složitost a časovou náročnost. Typicky obsahují aritmetické a logické funkce, dále typické konstrukce programovacích jazyků. Je nutné použití *chráněných* variant funkcí. Funkce a terminály bychom měli volit co nejlépe, aby vedli k co nejvhodnějším řešením našeho problému.

- *Fitness funkce* – k vyčíslení funkce se spustí program a ten je testován pomocí *trénovací množiny*, tu tvoří množina vstupních hodnot. Výstupy získané programem jsou porovnány s požadovanými hodnotami a následně je vyhodnocena odchylka. Tato odchylka je poté

chápána jako hodnota fitness. Výsledný program je nutné po dokončení evoluce ověřit *množinou testovacích vstupů.*[1]

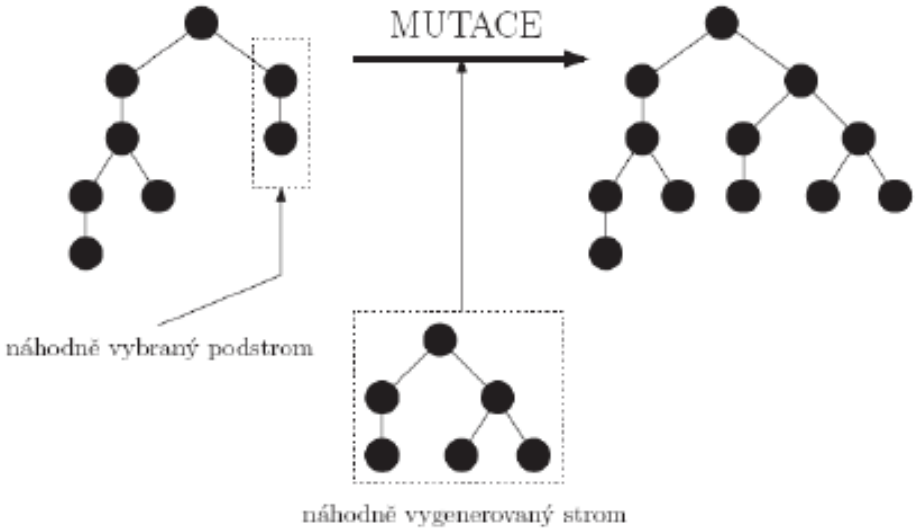
- *Hrubá fitness* – je odchylkou při řešení problému v jednotlivých případech.
- *Standardizovaná fitness* – převádí hrubou fitness (0 značí nejvyšší fitness).
- *Přizpůsobená fitness* – nejpoužívanější, nachází se v intervalu <0,1>.
- *Normalizovaná fitness* – podíl vhodnosti jedince ku vhodnosti všech jedinců v populaci.

- *Křížení* – kombinuje genetický materiál rodičovské dvojice prohozením podstromu jednoho rodiče s podstromem rodiče druhého. Výsledkem jsou dva nový potomci viz obr. 2.4.



Obr 2.4*:* Ukázka křížení. Zakroužkované uzly se vzájemně prohodí i se svými podstromy. [1]

- *Mutace* – pracuje pouze s jedním jedincem a většinou se aplikuje na potomky vzniklé křížením. Náhodně se vybere jeden uzel podstromu a tento uzel se nahradí náhodně vygenerovaným stromem viz obr. 2.5. Takto vzniklý jedinec se vloží zpět do populace.



Obr 2.5: Ukázka mutace. Vyznačený podstrom se odstraní a nahradí nově vygenerovaným podstromem. [1]

Pro všechny evoluční algoritmy je použití operátorů mutace a křížení podstatné a taky je důležitá volba jejich aplikace. U genetického programování se ve většině pramenů[5] doporučuje použití křížení dvou rodičovských jedinců nebo mutace jednoho rodičovského jedince (s nižší pravděpodobností, obvykle kolem 5%).

Nedostatky genetického programování:

- *Rozrůstání jednotlivých jedinců* – vznikají části programu, které mohou být syntakticky správně, ale neovlivňují výsledek. Jsou to takzvané *introny*. V průběhu evoluce dochází k nárůstu těchto intronů, což má za následek zbytečné zvětšování programu a zpomalení. Tento jev se nazývá *bloat*. Lze řešit vhodně zvoleným penalizováním velikosti jedinců při vyčíslování jejich fitness funkce.
- *Nízká použitelnost genetického programování* – v současné době se nedají použít na rozsáhlejší programy. Technika je vhodná pro vývoj drobnějších a dobře definovaných programů.
- *Generalizace* – evolucí vytvoříme program, který pracuje správně pro trénovací data, ovšem pro ostatní data z daného prostoru už správně pracovat nemusí. Tento problém se dá odhalit pomocí *testovací množiny*. Výsledkem je nízká generalizace výsledného programu.
- *Konvergence* – další velkou nevýhodou genetického programování je extrémně pomalá konvergence. Ta je způsobena časovou náročností výpočtu fitness funkce a taky z důvodu

nutné existence velikého množství jedinců v každé generaci. Tyto problémy částečně řeší pokročilá témata genetického programování (viz [1]).

2.2.3 Evoluční strategie

Evoluční strategie je dalším typem evolučních algoritmů, který dosahuje dobrých výsledků především pro optimalizační úlohy, při hledání extrémů funkcí *n* proměnných. Používají se v úlohách, kde nejsme schopni určit analytické řešení a klasické techniky selhávají. Úspěšnost těchto algoritmů je opět nezaručená a závisí na konkrétní úloze a volbě parametrů. Principy metod jsou shodné jako u genetických algoritmů, pokud se jedná o optimalizaci jedinců v populaci, dědění vlastností jedinců, a výběru lepších jedinců na základě jejich fitness funkce. Pokud se však jedná o optimalizaci funkce reálné proměnné, pracují s hodnotami reálných proměnných na rozdíl od genetických algoritmů (bitové řetězce převádějí na reálné hodnoty jen v krajních případech). Nejzajímavější vlastností evolučních algoritmů je schopnost autonomní dynamické změny probíhající evoluce. Právě tímto postupným snižováním mutačních kroků se algoritmus přibližuje hledanému optimu. Algoritmus pravidelně mění oblast prohledávání po několika krocích, dokud nedojde k požadovanému optimu. [1],[5] Základní znaky evoluční strategie [1],[5]:

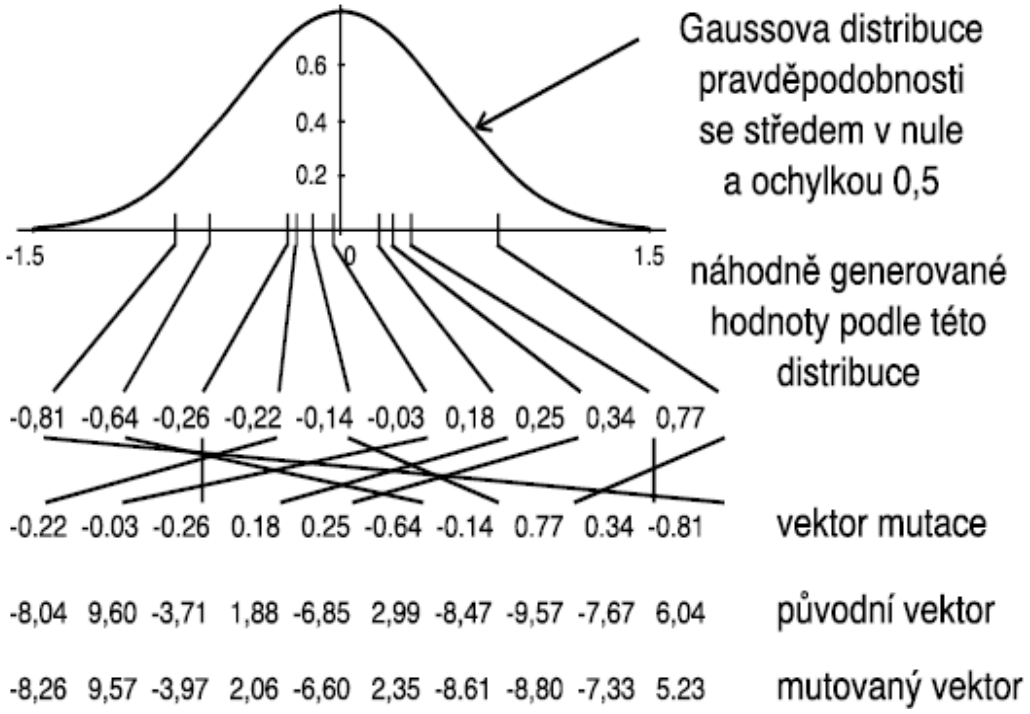
- *Reprezentace* – kandidátní řešení je kódováno v podobě vektoru reálných čísel, ale i jiných parametrů jako jsou řídící parametry evoluce [5]:

$$X = < x_1, x_2, ..., x_n, \sigma_1, \sigma_2, ..., \sigma_n, \alpha_1, \alpha_2, ..., \alpha_k > \qquad (2.5)$$

  kde x_i jsou hodnoty hledaných parametrů, σ_i hodnoty mutačních kroků $\forall\, i \in \{1, 2,..., n\}$ α_i jsou hodnoty rotačních úhlů $\forall\, i \in \{1, 2,..., k\}$ $k = n\,(n-1)/2$.

- *Mutace* – je klíčovým operátorem evoluční strategie. Mutace je uplatněna na každý parametr každého jedince přičtením náhodné hodnoty Gaussovy distribuce pravděpodobnosti(má nulovou střední hodnotu a odchylkou σ_i). Mutace probíhá podle následujícího vzorce (2.6), kdy se nejdříve vygenerují nové hodnoty rozptylů σ_i a s novými hodnotami rozptylů se poté provádí mutace parametrů x_i.

$$\sigma' = \sigma_i e^{\tau . N(0,1)}$$
$$x_i' = x_i + \sigma_i' . N(0,1) \qquad (2.6)$$

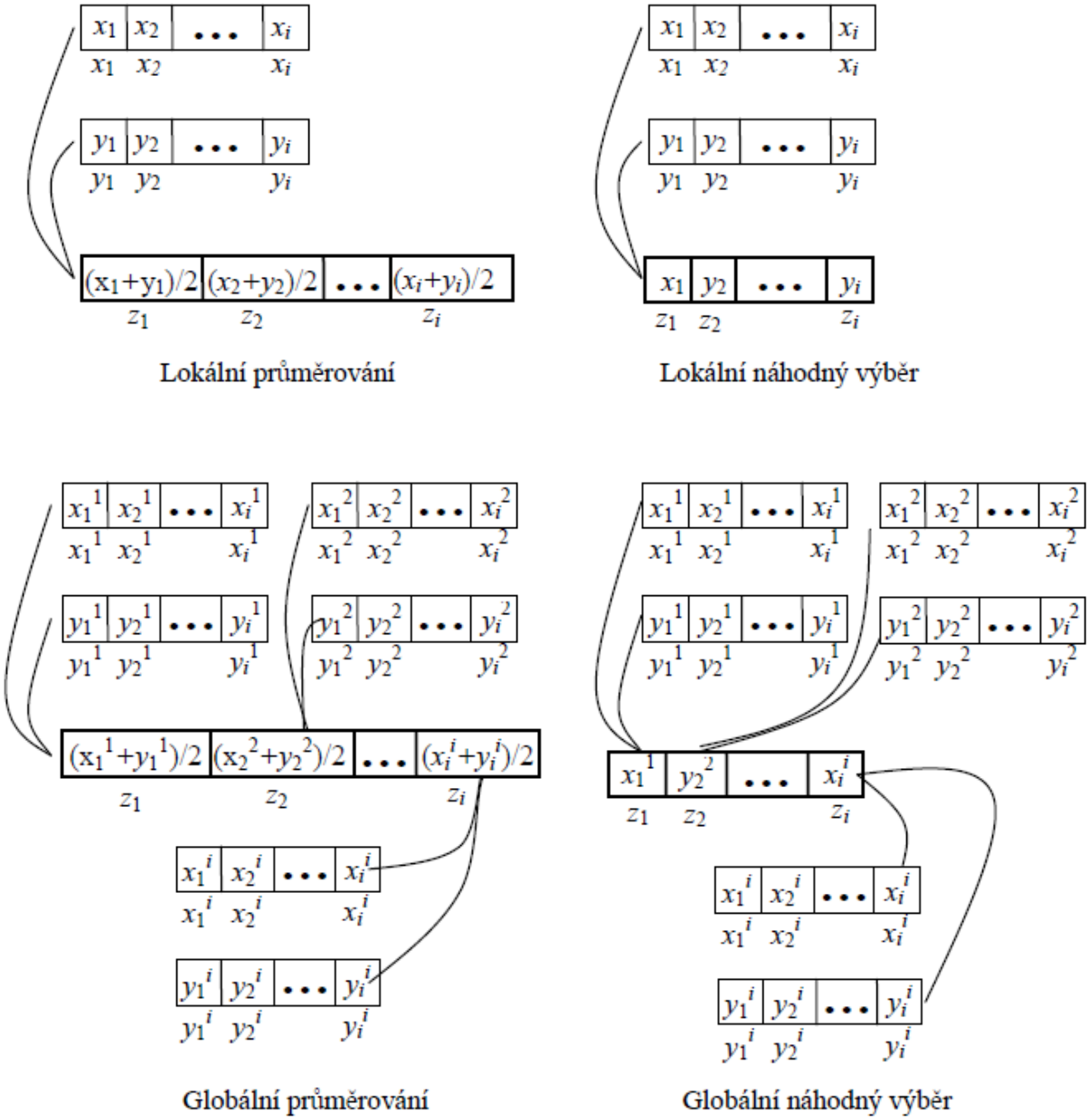


Obr 2.6: Generování mutace.[1]

kde $\tau \approx \frac{1}{\sqrt{n}}$, je tzv. *úroveň učení* a N(0,1) je náhodná množina s normálním rozložením, se střední hodnotou 0 a rozptylem 1.

Na obrázku je zobrazeno Gaussovo normální rozdělení se středem v nule a odchylkou 0,5, dále pak deset náhodně generovaných hodnot, podle této distribuce. Provede se náhodné prohození těchto hodnot, což tvoří tzv. vektor mutace. Hodnoty vektoru mutace jsou přičteny k původním (rodičovským) hodnotám. Výsledkem je mutovaný vektor tj. vektor potomka.

- *Křížení* – jedinci jsou zakódováni vektorem reálných čísel a řídícími parametry evoluce, proto je křížení specifické pro každý typ problému. Na rozdíl od genetických algoritmů, zde probíhá křížení *uniformní* tzn., že pravděpodobnost křížení je pro všechny jedince stejná, bez ohledu na hodnotu jejich fitness. Dva nejpoužívanější typy křížení u evoluční strategie jsou *diskrétní* křížení a křížení *průměrem.* Diskrétní křížení pracuje na principu náhodně zvolených hodnot z hodnot svých rodičů (z_i je náhodná hodnota z množiny $\{x_i , y_i\}$). Křížení průměrem funguje jednoduše, vezmou se hodnoty obou rodičovských jedinců a podle vzorce $z_i = (x_i + y_i)/2$ se vypočítá hodnota potomka. U obou těchto způsobů se mohou rodičovské dvojice volit pro každý parametr a nebo zůstává jedna dvojice pro všechny parametry nového jedince viz obr2.7.



Obr 2.7: Způsoby rekombinace jedinců u evolučních strategií (čerpáno z [1]).

- *Obnova populace (výběr jedinců)* – po provedení operátorů křížení a mutace vstupuje do populace nový jedinec. Tento jedinec vstupuje do populace na základě vyšší hodnoty *fitness*. Rozlišujeme tzv. *plus strategii* $(\mu+\lambda)$ – lepší potomek vytěsní z populace horšího rodiče, pokud je rodič lepší, tak zůstává v populaci. Další variantou je tzv. *čárková strategie* (μ,λ) – rodič je vždy nahrazen nejlepším z potomků, i když je potomek horší než rodič. Čím je násobnost λ (počet potomků) oproti μ *(počet rodičů)* vyšší, tím je vyšší i tzv. *selekční tlak* (používá se většinou varianta $\lambda \approx 7\mu$).

2.2.4 Evoluční programování

Evoluční programování je velice podobné evolučním algoritmům. Oba tyto postupy byly ovšem vymyšleny nezávisle na sobě. Byly používány v rámci konečných automatů, ve snaze navrhnout takový konečný automat, který s předpověďmi symbolických řetězců, které generuje z tzv. Markovovských procesů a měnících se časových sérií, dokáže předpovědět události, podle kterých pak mění své jednání. V dnešní době se dají použít i při řešení širšího okruhu optimalizačních úloh, tak jako evoluční algoritmy, ovšem používají se především k hledání optimálních parametrů existující struktury výpočetního modelu.[5] Základní znaky jsou [5]:

- *Reprezentace* – chromozómy jsou kódovány v podobě vektoru reálných čísel. V modernějším evolučním programování se používá pokročilejších schémat.
- *Křížení* – operátor křížení se standardně v evolučním programování nevyskytuje. Vychází se z živé přírody, kde také nedochází k mezidruhovému křížení a k vývoji určitého druhu dochází vzájemným soupeřením jedinců daného druhu.
- *Mutace* – operátor mutace má prakticky stejný význam jako u evolučních strategií, kdy dochází k součtu hodnot parametrů s náhodným gaussovským šumem.
- *Obnova populace* – výběr jedinců probíhá většinou turnajovým způsobem. Tzn. máme skupinu q, tato skupina se porovnává s náhodně vybranou skupinou rodičů (μ) a jejich potomků (λ) a do nové generace je vybrána skupina s největším počtem vítězství. Parametr q také ovlivňuje silu selekčního tlaku.

3 Mapovací systémy evolučních algoritmů

Tato kapitola byla čerpána z [7], [8].

Mnohobuněčné organismy jsou shluky buněk organizované ve strukturách, díky kterým ve výsledku vzniká možnost koordinace a funkcionality. Jeden mnohobuněčný organismus se skládá z několika set buněk, např. červ mikroskopické velikosti, případně se mohou skládat z astronomických čísel – např. lidské tělo, které obsahuje mnoho trilionů buněk. Organizovaný systém je založen na myšlence, že velké množství prvků ve struktuře je schopno vyřešit určitý problém objektivněji, než je schopen člověk ze svých každodenních zkušeností. Realizace struktury, která tvoří mnohobuněčný organismus, je mapováním procesu, které začíná od jedné buňky a staví postupně organizovanou strukturu formující kompletní organismus. Zatím nejsou přesně známy všechny detaily, které tvoří biologickou buněčnou strukturu, ale bylo objasněno již mnoho aspektů. Je známo, že tvorba finální struktury buněk ze vzorů organizované aktivity v prostoru a čase vzniká použitím různých mechanismů. Tyto mechanismy zahrnují výměnu signálů mezi buňkami, reakce buněk na okolní podmínky a schopnost růstu, dělení, zániku, migrace a diferenciace. Všechny tyto aktivity jsou ovlivněny v genomech a při počátečním stavu buněčného stadia, tzn. při soustředění a rozšíření chemikálií, které určují počáteční krok procesu mapování. Buněčné stadium nicméně neobsahuje kyanotyp (způsob fotochemického rozmnožování), ale především instrukce, které ve vhodném prostředí řídí proces organizované konstrukce organismu. Lze tedy říci, že buněčné stadium a jeho genom tvoří reprezentaci mapování organismu. Metoda mapování poskytuje možnost definice celkového popisu potenciálně velmi komplexních struktur. Proces mapování je organizovaný rozšířený proces, jehož aktivity jsou decentralizované a jsou charakterizovány paralelní operací svých prvků.

Základní problém s aplikací mapovacího přístupu k popisu návrhu umělého systému je definice mapovacího systému. V některých případech jsou příklady mapovacích pravidel i v přírodě, protože tato pravidla jsou aktuálně pozorována u existujících biologických organismů. Nebiologické instance jsou matematicky definovány podmínkami rekurzivních pravidel a tedy je potřeba přeložit tato pravidla do daného programovacího jazyka. Problém se stává mnohem složitější, pokud jsou spíše než detaily mapovacího procesu specifikovány požadované závěry mapování. Realizace mapovacího systému je dáno konstitucí výsledku mapování, které se nazývá problém *usuzování* mapovacího systému. Složitost s problémem usuzování pochází z komplexního vztahu, který obecně existuje mezi popisem mapování a výsledkem mapovacího procesu. Vyřešit problém usuzování mapovacích systémů můžeme metodou *pokus omyl*. Je potřeba mít počáteční odhad vlastností

systému mapování, pak můžeme vyvíjet jejich důsledky a porovnávat je s požadovanými účinky. Pokud je aktuální výsledek neuspokojivý, vracíme se zpět k prvnímu kroku a modifikujeme definici mapovacího systému.

3.1 Definice evolučního mapovacího systému

Umělé vývojové systémy je vhodné rozdělit do čtyř kategorií.

1. Jsou to systémy, kde mapovací proces je zcela parametrizován a evoluce tak může pracovat pouze s parametry, které jsou zakódovány v genetické informaci (genomu). Tento přístup klade velké nároky na návrh, právě na něm pak záleží vytvoření dobrého mapovacího procesu. Výsledek evoluce je zprostředkován netriviálním ručním návrhem mapování genotypu na fenotyp.

2. Dále systémy, kde základní mechanismy mapovacího procesu jsou zakódovány ve vyvinutém genotypu, ale jejich pořadí a počet užití je pevně daný a nemůže se vyvíjet. Obecně platí, že cílem tohoto přístupu není dosažení evolvability (snaha změřit schopnost organismu se vyvíjet), ale používá se hlavně k hledání mapovacího systému, který může přinést určité výsledky v rámci dané třídy mapovacích systémů. Tento přístup může být použitý např. k vývoji přepisovacích systémů, jako jsou *L-systémy*.

3. Systémy, kde základní mechanismy mapovacího procesu jsou ručně navrženy, ale jejich pořadí a počet užití je zakódován do genetické informace(genomu) a jsou předmětem evoluce (stále se vyvíjejí). Úkol navrhujícího, správně určit základní mechanismy, není samozřejmě triviální. Svoboda evoluce v kombinaci s těmito mechanismy umožňuje potenciální přístup k širokému spektru mapovacích procesů. Typickým příkladem takového přístupu, jsou mapovací systémy založené na genetickém programování.

4. Systémy, kde jsou některé aspekty obou základních mechanismů mapovacích systémů a jejich pořadí a počet užití jsou zakódovány v genomu a mohou se vyvíjet. Tyto systémy mohou prozkoumat větší prostor mapování a mají největší svobodu při hledání systémů, které mají nejlepší evoluční potenciál. Ovšem návrhář takového systému čelí volbě z mnoha nízkoúrovňových podrobností o evolučním prostředí. Pozorování biologických systémů, nám

napovědělo, že rozvíjet soubory vhodných mapovacích mechanismů úplně od základů je velice obtížné a problematické.

Tato klasifikace je užitečná pro výběr nejlepšího druhu umělého mapovacího systému pro danou aplikaci. Jakmile je vybrán druh evolučního mapovacího systému, je nutné pro něj vybrat genetické kódování. Obecně platí, že pro daný druh mapovacího systému, existuje mnoho možností genetického kódování. Pro dané genetické kódování musíme také zvolit vhodné genetické operátory, které ovlivňují vyhledávací parametry evolučního procesu. Obecně platí, že čím více je mapovací model abstraktní a formálně strukturovaný, tím více musíme dbát na vhodné definování genetických operátorů, tak aby zmutovaná genetická informace (genom) i nadále popisoval přesně stanovený mapovací systém. Možných kombinací prvků tvořících evoluční mapovací systémy je nespočetně mnoho.

3.2 Definice kódování

Důležitým rysem evolučních algoritmů je prohledávání, které je založeno na populacích. Populace *genotypů* patří do mnohodimenzionálního prostoru. Genotyp obsahuje geny a obvykle kóduje jeden *fenotyp.* Ten je kandidátem na řešení úlohy z příslušné domény řešení. Během kódování nabývají geny numerických hodnot tak, aby *genotyp* mohl být transformován na příslušný *fenotyp.* Celý dekódovací proces by měl být co nejjednodušší. *Fenotyp* je ohodnocen svou účelovou funkcí (*fitness*). Jsou preferovány genotypy s nejvyšší hodnotou *fitness,* které jsou tvořeny evolučními operátory (křížení, mutace) v průběhu generací. Vzájemné soupeření mezi jedinci se populace vyvíjí směrem ke genotypu, který koresponduje největší *fitness* fenotypu. To vede ke správnému řešení úlohy. Výkon vyhledávacích algoritmů je tudíž závislý na vlastnostech kódování a taky na použití genetických operátorů. Standardní kódování jsou binární, reálné vektory, stromové struktury atd. Optimalizační úlohy *f(x)* je možné rozdělit na mapování genotyp-fenotyp f_g a mapování *fitness* fenotypu f_p :[7], [8]

$$f_g(x_g): \Phi_g \rightarrow \Phi_p,$$
$$f_p(x_p): \Phi_p \rightarrow R,$$
$$kde\ f = f_p \circ f_g = f_p(f_g(x_g)).$$

Změna f_g také mění vlastnosti funkce *f.* Genetické operátory mutace a křížení jsou aplikovány na x_g, kdežto výběr řízení je založen na *fitness* x_p , $f_p(x_p)$ určuje obtížnost kódování a $f_g(x_g)$ určuje složitost problému. Existuje $\|\Phi_g\|!$ různých kódování. Kódování je právě mapování Φ_g na Φ_p. Určuje vhodné $x_g \in \Phi_g$ na $x_p \in \Phi_p$. Metrika musí být definována v obou vyhledávacích prostorech Φ_g a Φ_p. Určuje vzdálenost mezi jednotlivci a měří podobnost mezi nimi. Obecně platí, že metrika používána pro Φ_p je definována posuzovaným problémem a metrika používána pro Φ_g je určena použitím

operátorů. Genotypové operátory jako jsou *mutace* a *křížení* jsou definovány na základě používaných metrik. Metriky a používané operátory na sobě úzce závisí, jedno určuje druhé.

- *Přímé kódování* – pokud jsou genetické operátory aplikovány přímo do fenotypů, není nutné specifikovat kódování a fenotypy jsou identické s genotypy:

$$f_g(x_g): \Phi_g \to \Phi_g$$
$$f_p(x_p): \Phi_g \to R.$$

To znamená, že f_g je identická funkce s funkcí $f_g(x_g) = x_g$. Použití přímého kódování nemusí nutně znamenat, že si problém usnadníme, protože návrh správných genetických operátorů je obtížný.

- *Nepřímé kódování* – použitím vhodného mapování získáváme určité výhody. Může se považovat za specifické omezení. Mohou být použity normalizované genetické operátory jako binární (prosté GA) nebo kontinuální (ES) se známým chováním a vlastnostmi, které jsou dobře srozumitelné. Např. při mapování mnoha různých fenotypů na pouze několik genotypů. Nepřímé kódování je nezbytné, pokud je obtížné navrhnout konkrétní operátory, které řeší daný problém nebo pokud nejsou takové operátory k dispozici. Návrh takových operátorů ovšem může být velmi obtížný a k zajištění platného řešení, musí být často použity další opravné mechanismy. Kódování může problém usnadnit začleněním specifických znalostí pro daný problém.

3.3 Vlastnosti kódování

3.3.1 Vysoká lokalita (High-locality) kódování

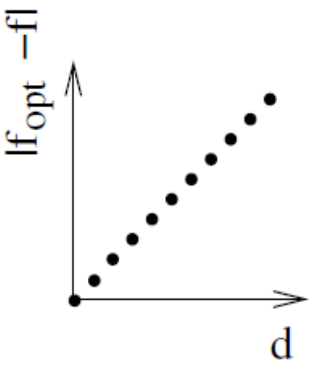
Mapování genotypu na fenotyp může změnit strukturu okolí a strukturu návrhu krajiny (strukturu fitness). Každý soused může být dosažen přímým pohybem (mutací, křížením), proto je okolní struktura závislá na použití operátorů a metrik. Soubor okolí může být odlišný pro genotypy a fenotypy. Lokalita kódování popisuje, jak dobře okolní genotypy odpovídají okolním fenotypům. Jestliže jim odpovídají, tzn. mají řešení blízko sebe, pak mluvíme o vysoké lokalitě kódování.

Rozdílné typy fenotypového návrhu mapování

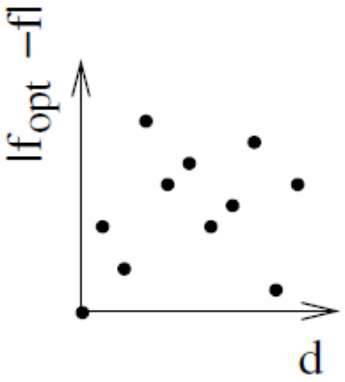
Kategorie 1: Nejlepší řešení jsou blízko sebe a pozitivně korelují se vzdáleností k optimálnímu řešení (přibližují se k optimálnímu řešení viz obr 3.3). Snadná mutace, základní vyhledávání.

Kategorie 2: Nejlepší řešení jsou volně rozházeny v prostoru a nekorelují se vzdáleností k optimálnímu minimu (obr.3.2). Struktura prostorů hledání nestanoví informace pro řízený způsob hledání – obtížné pro řízené způsoby hledání.

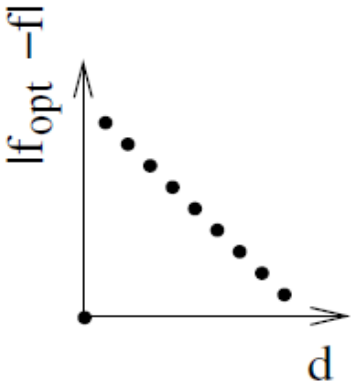
Kategorie 3: Nejlepší řešení negativně korelují se vzdáleností k optimálnímu řešení (vzdalují se od optimálního řešení obr. 3.1). Struktura prostoru hledání klame – klamný problém.



Obr 3.3: Kategorie 1: Positivní korelace [7].



Obr 3.2: Kategorie 2: Nesouvztažnost [7].



Obr 3.1: Kategorie 3: Negativní korelace [7].

Nízká versus vysoká lokalita

Kategorie 1:

Vysoká lokalita kódování zachovává obtížnost problému. Jednoduché problémy zůstávají pro řízené hledání snadné.

Nízká lokalita kódování dělá problémy více obtížné. Vyplývající problém se stává druhou kategorií.

Kategorie 2:

Vysoká lokalita kódování zachovává obtížnost problému. Problémy zůstávají obtížné pro řízené hledání.

Nízká lokalita kódování v průměru nemění složitost problému. Problémy zůstávají obtížné.

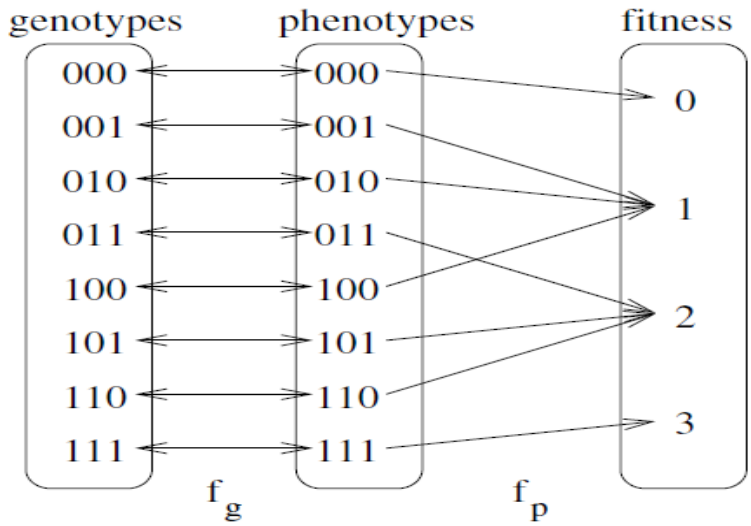
Kategorie 3:

Vysoká lokalita kódování zachovává ošidnost problému. „Pasti zůstávají pastmi.“

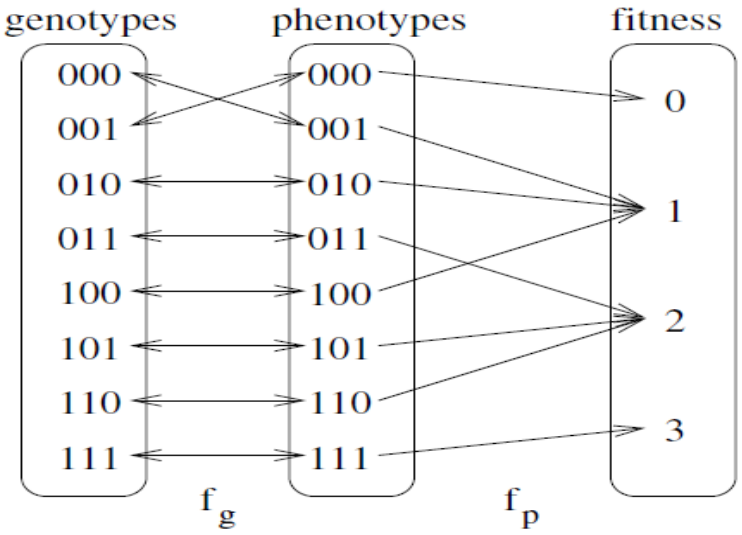
Nízká lokalita kódování přemění problém do druhé kategorie problému. Problémy zůstávají obtížné.

Příklady lokality

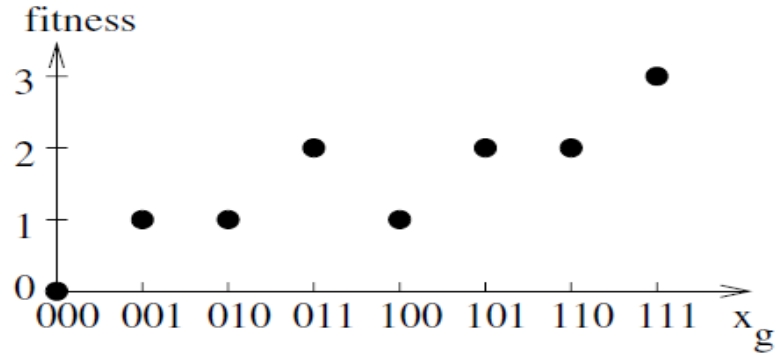
Genotypy i fenotypy jsou binární. Jako krok použijeme *bit-flipping*. Hledáme maximální počet 1 v bitovém řetězci. Všechny stavební bloky jsou velikostí $k = 1$. Problémy jsou jednoduché pro selektorekombinační GA (algoritmy, používající jako operátory jen selekci a křížení, nikoliv mutaci) viz obr. 3.4. Na obr. 3.5 je kódování s nižší lokalitou, kde dochází ke změnám okolní struktury. Ne všechny genotypové stavební bloky jsou velikosti 1. Ačkoliv funkce f_p zůstává nezměněná, tak funkce f se stává náročnější. Na obr. 3.6 je názorně zobrazena *vysoká lokalita* kódování, kde je problém snadno řešitelný pomocí operátorů křížení a selekce. Na obr. 3.7 je zobrazen rozdílný návrh genotypů 000 a 001, tento problém se stává obtížnější pro selektorekombinační GA. Okolí není uchované podle kódování.



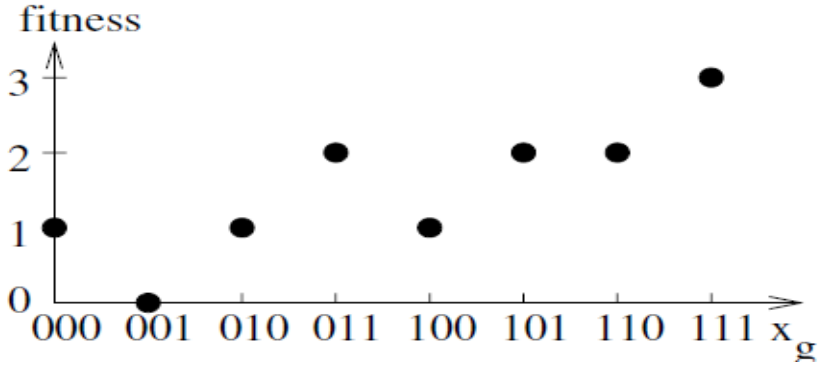
Obr 3.4: Mapování genotypu na fenotyp [7].



Obr 3.5: Mapování genotypu na fenotyp [7].



Obr 3.6: High-locality kódování [7].



Obr 3.7: High-locality kódování [7].

Při používání *vysoké (high) locality* kódování okolí genotypu odpovídá okolí fenotypu. Nemění strukturu obtížnosti problému, jednoduché problémy zůstávají jednoduchými a obtížné obtížnými. Záleží na použité vzdálenosti metrik, která je závislá na použitých operátorech.

3.3.2 Redundantní kódování a neutrální sítě

Redundantní kódování znamená, že počet genotypů je větší než počet fenotypů. Použití redundantního kódování f_g znamená proměnlivé $f = f_p (f_g)$. Redundantní kódování je méně užitečné kódování, které používá vyšší počet alel (projev genu), ale nezvyšuje množství zakódovaných informací. Redundantní kódování není vynálezem AI výzkumů, ale je vypozorováno z přírody. Zde existují rozdílné názory vlivu na výkon EA, někteří tvrdí, že dochází ke snižování výkonu kvůli ztrátě rozmanitosti (Davis, 1989; Eshelman and Schaffer, 1991), jiní zase zastávají názor, že redundantní kódování zvyšuje výkon EA (Gerrits and Hogeweg, 1991; Cohoon et al., 1988; Julstrom, 1999). Velké množství názorů se uchyluje k *neutrální teorii.*

Tato teorie předpokládá, že nepřirozená selekce stanoví výhodné mutace, ale náhodná zachycení neutrálních mutací je hybnou silou molekulární evoluce. Tyto nápady redundantního kódování (neutrální sítě) byly použity s velkým nadšením v EA v naději, že zvyšování vývinu systému také zvyšuje výkonnost systému. Knowles a Watson v roce 2002 názorně dokázali, že to není zcela pravdivé tvrzení. *Neutrální síť* je soubor genotypů spojených pomocí jednobodových mutací, které mapují na stejný fenotyp. Výhodou je, že populace může proudit po těchto sítích, což snižuje šanci uváznutí v neoptimálních řešeních. Populace je schopná se rychle obnovit po každé změně, která nastala. Vzrůstá evolvabilita systému. Při použití redundantního kódování rozlišujeme:

Synonymní redundantní kódování: všechny genotypy, které kódují stejný fenotyp jsou jeden druhému podobné. Lze ho popsat pomocí pravidla $k_r = \frac{\log(|\Phi_p|)}{\log(|\Phi_g|)}$ podle kódování r optimálního řešení vzhledem k problému kódování f_g.

Při použití pojmu BB (Begin Block) a binárního kódování: $k_r = \frac{k_g}{k_p}$

r: počet genotypových BB uspořádání z k_g, které představuje optimální fenotypové BB uspořádání k_p.

Příklad:[7]

genotypes x_g	x_p
00 00, 00 01, 01 00, 01 01	0 0
10 00, 10 01, 11 00, 11 01	1 0
00 10, 01 11, 00 11, 01 11	0 1
10 10, 10 11, 11 10, 11 11	1 1

k = 2 (uspořádaní fenotypových BB)

k_r =2 (jedna alela fenotypu je zastoupena použitím dvou alel genotypu). Jednotná redundance: r = 4 (nejlepší BB je zastoupen čtyřmi genotypovými BB)

Nesynonymní redundantní kódování: genotypy, které kódují stejný fenotyp, nejsou jeden druhému podobné. Toto kódování nepovoluje řízené hledání. Vyhledávání EA se stává náhodné. Choi a Moon definovali jednotně redundantní kódování, která jsou maximálně nesynonymní, a ukázali, že takové kódování vyvolává nekorelující vyhledávání prostorů. Pro takové kódování je očekávaná vzdálenost mezi jakýmikoliv genotypy, které odpovídají stejnému fenotypu, neměnná a téměř rovná velikosti *n.* Normalizace (transformace jednoho zdroje je konzistentní s ostatními) může přeměnit nekorelující hledací prostory do korelujících hledacích prostorů s vyšší lokalitou. Některé vybrané příklady:

- *Dělení problémů v grafech*: *k* podmnožiny jsou zastoupeny celými čísly od 0 do *k*-1, kde uzly jsou obsaženy ve stejné skupině, jestliže jsou zastoupeny stejným číslem. Každý fenotyp je zastoupen *k* odlišných genotypů.

• *HIFF problémy* – binární zakódování, kde je každý fenotyp zastoupen podle páru bitových doplňkových genotypů.

• *TSP* – založen na křížení, ve kterém jsou vrcholy indexovány od 1 do n a každá cesta je založena podle změny vrcholových indexů a každý fenotyp je založen na $2n$ genotypu.

Triviální volba mapování: přiřazuje binární fenotypy k binárním genotypům, jeden bit fenotypu je zastoupen k_r genotypovými bity. Obecně platí, že fenotypový bit je roven 0, jestliže méně než u genotypových bitů je rovno 0. Pokud více než u genotypových bitů je rovna 1, pak fenotypový bit je také roven 1. Pro $u = k_r / 2$ je hodnota fenotypového bitu určena většinou genotypovými bity.

Obecně platí,že :

$$x_i^p = \begin{cases} 0\,if & \sum_{j=0}^{k_r-1} x_{k_r i+j}^g < u \\ 1\,if & \sum_{j=0}^{k_r-1} x_{k_r i+j}^g \geq u, \end{cases}$$

kde $u \in \{1, \ldots, k_r\}$.

Příklady [7]:

genotypes x_g	x_p
000, 001, 010, 100	0
110, 101, 011, 111	1

$k = 1$

$k_r = 3$

$u = 2$

genotypes x_g	x_p
000	0
001, 010, 100,110, 101, 011, 111	1

$k = 1$

$k_r = 3$

$u = 1$

Existují teoretické modely, které nám povolují předpovědět očekávanou GA výkonnost při použití redundantního kódování $N=O(2^{kr}/r)$. Existují některé doporučení pro návrh redundantního kódování, a to nepoužívat nesynonymní redundantní kódování. Musíme zkoumat, jak kódování změní velikost vyhledávacího prostoru a zda jsou nadměrně zastoupeny podobná řešení k optimálnímu řešení. Pokud nejsou k dispozici poznatky o optimálním řešení, je doporučeno použít jednotně redundantní kódování.

4 Návrh systému mapování

4.1 Evoluční algoritmus

Při tvorbě evolučního algoritmu, jsem uplatnil znalosti získané z evoluční strategie. Tento algoritmus dosahuje dobrých výsledků především pro optimalizační úlohy, při hledání extrémů funkcí *n* proměnných. Používají se v úlohách, kde nejsme schopni určit analytické řešení a klasické techniky selhávají. Úspěšnost těchto algoritmů je nezaručená a závisí na konkrétní úloze a volbě parametrů.

Evoluční algoritmus má volitelnou velikost populace a obsahuje také proměnnou, která říká, jaká je pravděpodobnost, že se mutace vykoná, zda se využije mapovaní či ne nebo jaká funkce bude zvolena. K selekci využívá kolo štěstí, kde selekci rodiče ovlivňuje hodnota fitness, nebo turnaj. Dále je v selekci zohledňován nejlepší rodič, a proto se vždy vybere nejlepší jedinec a k němu selekční metodou další do doplnění populace. Nepoužitím selekčních metod, které vyžadují setřídění populace, urychluje výpočet evoluce. Každý potomek má jistou pravděpodobnost *pMutace*, že zmutuje. Pokud ano, zmutuje parametr *x* nebo *y* nebo zmutují oba zároveň. Mutace se provede pouhým přičtením či odečtením změny. Hodnota změny ovlivňuje jak náhoda tak i rozměr definovaného prostoru (pro menší prostor jsou změny menší). Zda se bude odčítat nebo přičítat, ovlivňuje pouze náhoda.

Obnovení populace se provede podobně jako výměna rodičů za potomky, pouze s tím rozdílem, že se dva náhodně vybraní potomci z intervalu <2,pocet_populace-1> smažou, na začátek se přidá nejlepší rodič a nakonec se náhodně vygeneruje nový jedinec. Protože se nemažou první dva potomci v populaci, zvýhodňuje se mutace nejlepšího rodiče. Tímto je dosaženo, že první je buď nejlepší nebo jeden z nejlepších (pokud je jeden z potomků lepší).

Evaluace fitness důležitou částí evolučního algoritmu, protože dle ní bude evoluce konvergovat či divergovat. Fitness funkce vypočte funkční hodnotu na souřadnicích (x,y) pro zvolenou funkci a mapování. První mapovací funkce je mapování pomocí logaritmu. Pro toto mapování je potřeba souřadnice (x,y) vypočítat postupným logaritmováním, z důvodu velkého čísla, které by mohlo přetéct. Princip je dle vět o logaritmování která říká že log(a*b) = log(a) + log(b). Jako druhá mapovací funkce je mapování pomocí lineární funkce.

4.1.1 Aplikace

Program je naimplementován v jazyce C/C++ a pro vykreslování grafiky jsem použil knihovnu GLUT, což také dovoluje spustitelnost tohoto programu i pod Linuxem. Umožňuje nastavit celkem 7 parametrů z příkazové řádky, syntaxe je následující:

mapovani velikost_populace xmin xmax ymin ymax mapovani a b pmut < log.txt

int velikost_populace	//kolik generací se má spuštění maximálně provést (předdefinováno 0)
int xmin	//dolni mez x-ove souradnice(předdefinováno -60)
int xmax	//horni mez x-ove souradnice(předdefinováno 60)
int ymin	//dolni mez y-ove souradnice(předdefinováno -60)
int ymax	//horni mez y-ove souradnice(předdefinováno 60)
int mapovani	//typ mapovani (0): 0=neni,1=logaritmus,2=linearni
float a	//parametr pro linearni mapovani a*x+b (nutno zadavat 0 < a < 1) neošetřeno (a=0.5)
float b	//parametr pro linearni mapovani a*x+b (b = 0.0)
double pmut	// 0.0 – 1.0 pravděpodobnost mutace (předdefinováno 0.5)
< log.txt	//přesměrování výpisů ze stdout do souboru log.txt

Parametry jsou předdefinované, a proto se nemusí uvádět, slouží jen pro možnost změny parametrů. Nemusí se uvádět všechny, vždy ale musí být v tomto pořadí, protože program neošetřuje chybné zadání parametrů. Například pokud chci nastavit meze pro *y*, musím nastavit i předchozí parametry. Další parametry se doplní předdefinovanými hodnotami.

Aplikace umožňuje zkoušet hledání minima pro definovaný prostor šesti funkcí s mapováním nebo bez mapování. Meze se nastavují pouze při spuštění, funkce a mapování za běhu. Vývoj je po jednotlivých generacích, který uživatel ručně simuluje (klávesou N) a tím vidí i průběh , který může zastavit.

4.1.2 Ovládání

Pro manipulací s evolučním algoritmem a také celým programem jsou přednastavené klávesy:

šipky	...	vertikální a horizontální posun zobrazení
+, -	...	zoom zobrazení
1-6	...	výběr funkce
I	...	inicializace nové populace (nastaví se generace opět na 0)
N	...	spouští celou jednu generaci dle vstupního nastavení
O	...	mapování (přepínání mezi 0 – bez mapování, 1 – s logarit. mapováním)
T	...	nastavení selekce na turnaj
W	...	nastaveni selekce na kolo štěstí
R	...	reset zobrazení (posunutí a zoom)

4.1.3 Zobrazení:

Zobrazují se tři grafy. Vlevo dole je graf fitness = f(x) napravo od něj je graf fitness = f(y) a nad nimi je půdorys a to y = f(x).

Pro zobrazení fitness se vypočte maximum fitness a zobrazuje se v intervalu (0, fitness_max), což má za následek případně podrobnější náhled, který na první pohled vypadá, že fitness je vysoká. Bohužel popisky grafů jsou pouze bez číselných hodnot a záporná fitness se zobrazuje pod graf.

4.2 Testovací funkce

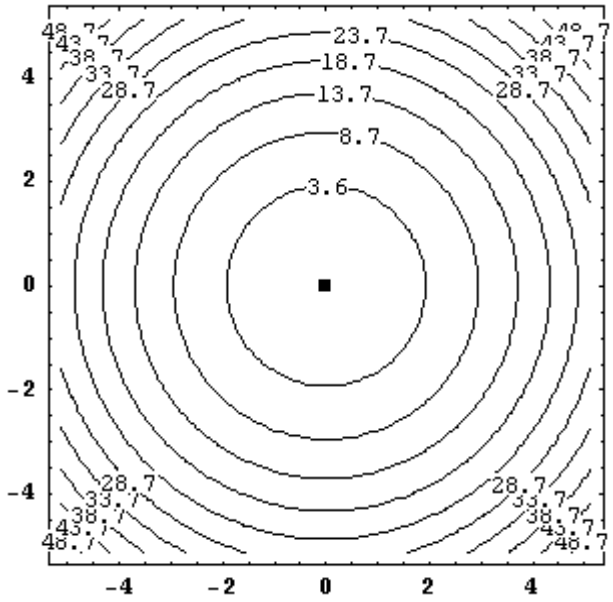
Tato kapitola byla čerpána z [9].

4.2.1 První de Jongova funkce

$$\sum_{i=1}^{D} x_i^2$$

Globální minimum:

v E_n : na pozici $(x_1,x_2,...x_n) = (0,0,...,0)$ o hodnotě y = 0 * n = 0



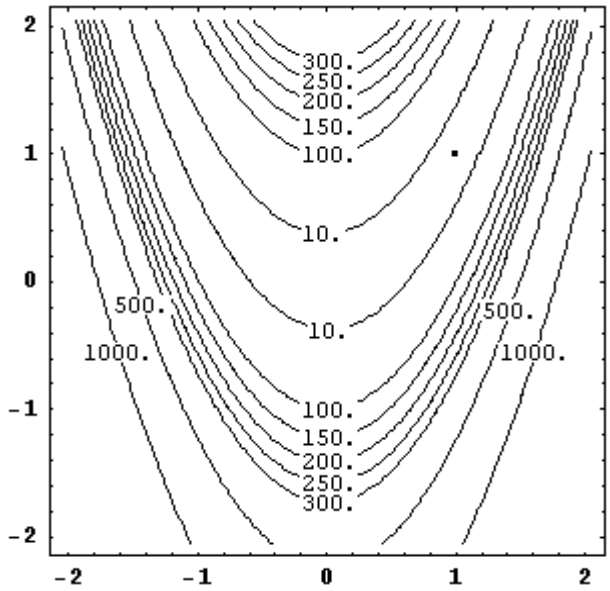
Obr 4.1: První de Jongova funkce. Tmavý bod reprezentuje pozici globálního extrému [9]..

4.2.2 Druhá de Jongova funkce

$$\sum_{i=1}^{D-1} 100\cdot(x_i^2 - x_{i+1})^2 + (1 - x_i)^2$$

Globální minimum:

v En : na pozici (x1,x2,...xn) = (1,1,...,1) o hodnotě y = 0 * n = 0



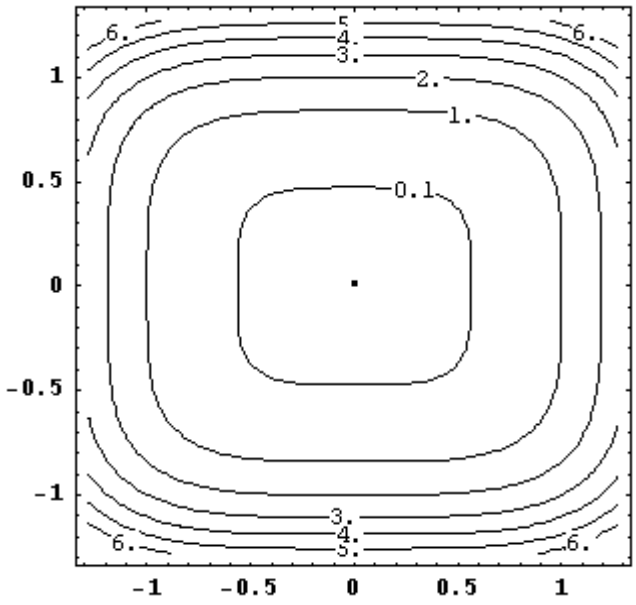
Obr 4.2: Druhá de Jongova funkce. Tmavý bod reprezentuje pozici globálního extrému [9].

4.2.3 Čtvrtá de Jongova funkce

$$\sum_{i=1}^{D} i \cdot x_i^4$$

Globální minimum:

v E_n : na pozici $(x_1,x_2,...x_n) = (0,0,...,0)$ o hodnotě y = 0 * n = 0



Obr 4.3: Čtvrtá de Jongova funkce. Tmavý bod reprezentuje pozici globálního extrému [9].

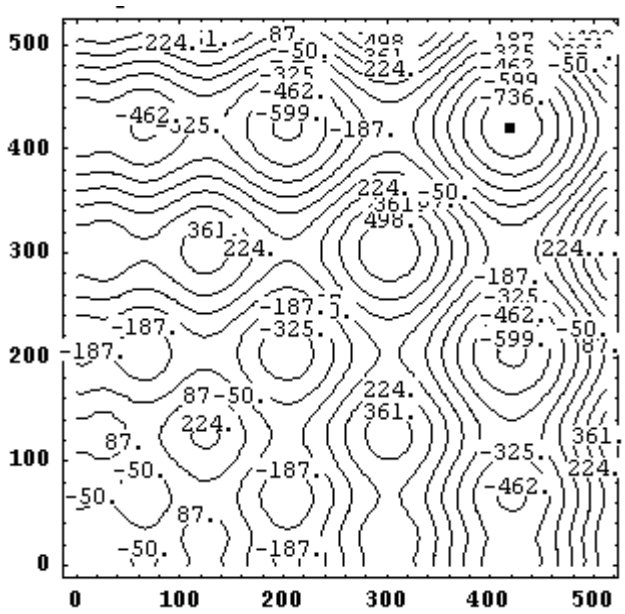
Tyto funkce jsem vybral záměrně, protože mají pouze jedno minimum, které je zároveň i globálním minimem a to dokonce v (0, 0). Pro tyto funkce by mělo být mapování výhodné, protože se namapují body blízko kolem globálního minima.

4.2.4 Schwefelova funkce

$$\sum_{i=1}^{D} \left(-x_i \cdot \sin\left(\left|\sqrt{x_i}\right|\right)\right)$$

Globální minimum:

v E_n : na pozici $(x_1,x_2,...x_n) = (420{,}969;...;420{,}969)$ o hodnotě y = -418,983 * n



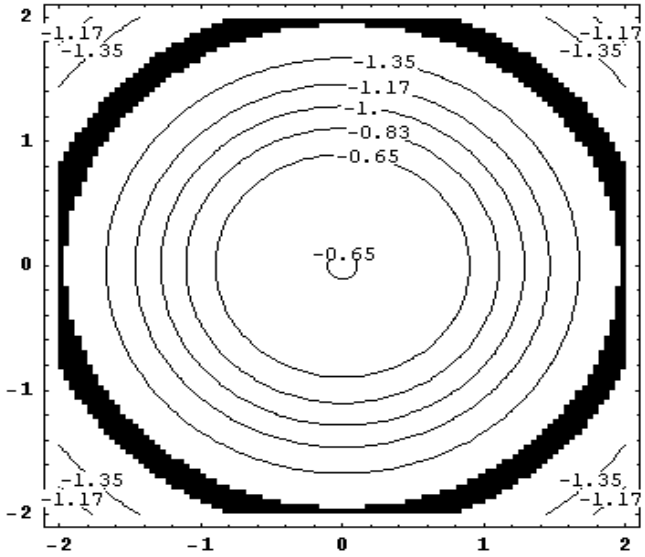
Obr 4.4: Schwefelova funkce. Tmavý bod reprezentuje pozici globálního extrému [9].

4.2.5 Sinová obálka sinusoidální funkce

$$\sum_{i=1}^{D-1}\left(0{,}5+\frac{\sin\left(x_i^2+x_{i+1}^2-0{,}5\right)^2}{\left(1+0{,}001\cdot\left(x_i^2+x_{i+1}^2\right)\right)^2}\right)$$

Globální minimum:

v E_n : na pozici kružnice v E_n o hodnotě y = -1,4915 * (n-1)



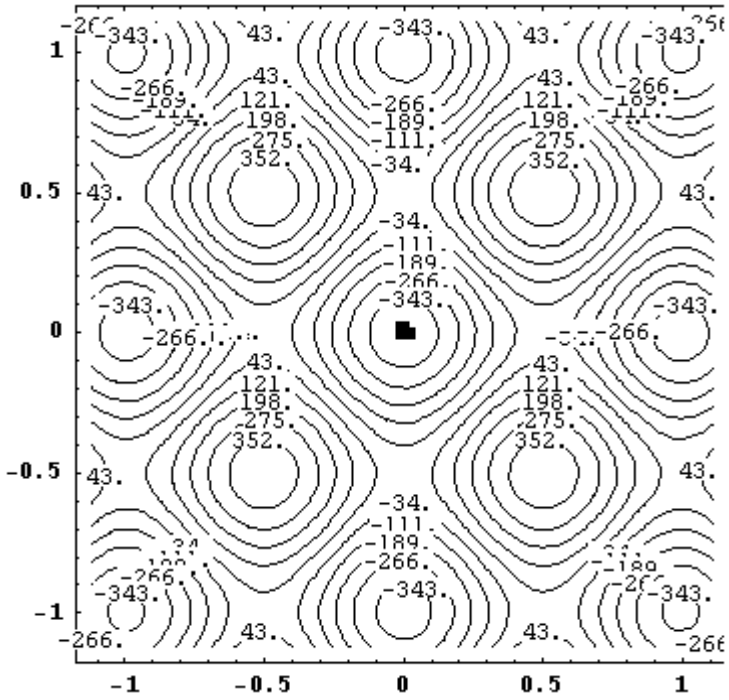
Obr 4.5: Sinová obálka sinusoidální funkce. Tmavý bod reprezentuje pozici globálního extrému [9].

Tyto funkce mají globální minimum jinde než v (0,0) nebo jich mají dokonce více. Např. u sinové funkce je to celá kružnice. U těchto typů funkcí by mapování nemělo být výhodnější, spíše naopak, rovnoměrné rozložení souřadnic lépe najde globální minimum mimo (0,0).

4.2.6 Rastriginova funkce

$$2 \cdot D \cdot \sum_{i=1}^{D} x_i^2 - 10 \cdot \cos(2 \cdot pi \cdot x_1)$$

v E_n : na pozici $(x_1,x_2,...x_n) = (0,0,...,0)$ o hodnotě y = -200 * n



Obr 4.6: Rastriginova funkce. Tmavý bod reprezentuje pozici globálního extrému [9].

Tato funkce má spoustu lokálních minim, avšak globální má v (0,0). Zde by mělo být mapování hodně výhodné hlavně kolem bodu (0,0) a tedy jak logaritmus, tak i lineární mapování. Bez mapování snadno evoluční algoritmus uvázne v lokálním minimu.

4.3 Mapovací funkce

4.3.1 Mapování

Náš mapovací proces je zcela parametrizován a evoluce tak může pracovat pouze s parametry, které jsou zakódovány v genomu. Jedná se o zjednodušený první přístup mapování. Tento přístup klade velké nároky na návrh, právě na něm pak záleží vytvoření dobrého mapovacího procesu. Výsledek evoluce je zprostředkován netriviálním ručním návrhem mapování genotypu na fenotyp.
Mapování se provádí pouze při generování jedinců. Vygeneruje se náhodné číslo a hned se namapuje na předdefinovaný prostor a s touto hodnotou se pak počítá jako s hodnotou bez mapování (sekce mutace, výpočet fitness i vykreslení). Rozhodl jsem se tak, protože mutace tak velikého čísla, které by se logaritmem namapovalo do definovaného prostoru, neudělala takřka žádnou změnu. I přesto by číslo nikdy k souřadnici (0,0) nedošlo.

4.3.2 Pomocí logaritmu

Kde mapovací funkce je dekadický logaritmus. Teoreticky takto:
f(x,y) → f(log(x), log(y)) pro x>0, y>0 .. I. kvadrant
f(x,y) → f(log(x), log(-y)) pro x>0, y<0 .. II. kvadrant
f(x,y) → f(log(-x), log(-y)) pro x<0, y<0 .. III. kvadrant
f(x,y) → f(log(-x), log(y)) pro x<0, y>0 .. VI. kvadrant
Ovšem při generování (*x,y*) aby se i po logaritmu namapovalo na daný prostor MxN, se hodnota *x* generuje z intervalu 1..10^N a hodnota *y* se generuje z intervalu 1..10^M. Což pro velká M a N jsou veliká čísla.
Prakticky se generování a následné počítání logaritmu dělá po částech, tedy využívají se věty:
$(ab)^n = a^n b^n$ ...(toto se využívá při generování náhodného čísla *x* i čísla *y*)
log(ab) = log(a) + log(b) ...(toto se využívá na zpětné namapování x a y na definovaný prostor)
Díky nímž lze rozdělit číslo na součin menších čísel, nad kterými se provede náhoda a nad těmito se pak provede součet logaritmů. Je zvoleno číslo exp = log(32768) = 4.51, protože rand() generuje čísla pouze 0 .. 32467 (tedy modulo 32768) a počítá se $rand()\%10^{exp}$ Takto náhodně vygenerované číslo se bez problémů vejde do základních typů např. int. Tato čísla se ukládají do vektoru.
Při zpětném logaritmování se hodnoty logaritmů přičítají k dolní mezi prostoru, tedy:
$x = xmin + \sum \log(x_i)$ pro každé x_i z pomocného vektoru násobků *x*. Podobně je to i pro souřadnici *y*.
Takto vygenerované a namapované hodnoty se pak mají teoreticky shlukovat kolem souřadnice (0,0). Prakticky se shlukují kolem souřadnic (1,1), (1,-1), (-1,-1) a (-1,1). V aplikaci dokonce (10,10) pro

každý kvadrant, což pravděpodobně způsobuje postupné generování náhodného čísla, protože například předdefinovaná mez 60 se rozdělí na tři čísla, kde se hledá náhoda v intervalu (0,32768)Po dekadickém zlogaritmováním je vysoká pravděpodobnost, že logaritmus náhodného čísla bude 3-4. Když sečteme 3 taková čísla, shlukují se pak všichni vygenerovaní jedinci právě kolem bodu 10 v souřadnici *x* i *y*.

4.3.3 Pomocí lineární funkce

Zde není třeba složené funkce a platí jednoduše f(x,y) → f(ax+b, ay+b).
Je vhodné zadávat číslo *a* v intervalech $0 < a < 1$. Číslo má být vyšší než nula, aby se nepřevracely hodnoty a nižší než jedna, aby se negenerovala čísla mimo definovaný prostor (což nevadí, ale je to zbytečné). Pro *b* omezení není, ale je nutné volit vhodné číslo *b*, aby se daný zúžený prostor nedostal mimo definovanou oblast. Pokud se tak stane, hodnoty se sice počítají, ale nevykreslují.

4.4 Testování a vyhodnocení výsledků

První testování proběhlo na funkcích *První de Jongova funkce, Druhá a Čtvrtá de Jongova funkce.* Provedl jsem 15 testů pro konkrétní případy u jednotlivých funkcí a porovnal jaký vliv mají jednotlivé mapovací funkce na vyhledání globálního minima. Podle zpracovaných výsledků viz. tab. 4.2 a tab. 4.3 je patrné, že algoritmus bez mapování má jisté výhody i nevýhody. Výhodou je, že náhodné generování, umožňuje po celém definovaném prostoru rychle najít řešení tab. 4.2. Nevýhodou může být, pokud máme definovaný prostor velkých rozměrů a malý počet jedinců populace. Je zde malá šance, že se podaří „trefit“ blízko středu a tudíž výpočet se stává pomalejší (minimum se najde pomaleji) tab. 4.3.

Mapování logaritmem se v našem případě prokázalo jako méně efektivní při prohledávání menšího prostoru s velkým počtem jedinců viz. tab. 4.2. Námi naimplementované mapování negeneruje tak velká čísla a proto vždy (nikoliv náhodou), budou generována čísla blízko hledaného minima, tedy středu (0,0). Při velkých rozměrech prostoru a s malým počtem jedinců populace, může být výhodnější než algoritmus bez mapování nebo s lineárním mapováním, protože i málo bodů namapuje přibližně stejně blízko hledaného středu viz tab.4.3. V našem případě nastává problém, protože se vždy vygeneruje kolem daného čísla (10,10) pro předdefinovaný prostor. Jelikož se mutace počítá tak, že je pevně stanovená maximální hodnota změny MAX a je vypočítána změna=rand()MAX a náhodně se vypočítá znaménko. Logaritmus vždy vygeneruje čísla kolem (10,10) i kdyby náhoda dávala pořád změnu o hodnotě MAX, tak to bude trvat jistý počet generací, protože se hodnota nemění, proto i řešení pomocí logaritmu udává přibližně stejný počet generací.

Problém proč se čísla generují kolem hodnoty (10,10) je z důvodu velkých čísel a funkce rand(), kterou jsme použili na náhodné generování, je maximálně pro modulo 32768, tudíž je potřeba velké číslo vyjádřit jako součin menších čísel viz. tab. 4.1.

x=log(x_velke)	x_velke= rand()%32768	pravděpodobnost vygenerování %
0 - 1	1 - 10	10/32768 = 0,03%
1 - 2	11 - 100	90/32768 = 0,27%
2 - 3	101 - 1000	900/32768 = 2,7%
3 - 4	1001 - 10000	9000/32768 = 27%
4 a větší	10001 - 32768	22768/32768 = 70%

Tab. 4.1 Procentuální pravděpodobnost náhodného generování

Stejný princip platí i pro souřadnici *y*, protože *fitness = f(x,y).* Tudíž máme 97% pravděpodobnost, že při generování například 3 různých čísel, se nám vygenerují čísla v rozsahu 3 až 5 například 4+3+4=11. Z toho vyplývá proč nám logaritmické mapování velkých čísel mapuje kolem souřadnice (10,10) pro každý kvadrant.

Výhoda lineárního mapování na těchto funkcích je opět ta, že náhodné generování umožňuje po celém definovaném prostoru rychle najít řešení. Navíc je zúžen prostor, kam se náhodně generují čísla. Prakticky můžeme toto mapování chápat tak, že hledáme podrobněji (tzn. vygenerujeme větší počet čísel na menším prostoru). Kde není třeba hledat, negenerují se hodnoty vůbec. Protože se jedná o náhodné rovnoměrné rozložení čísel, může nastat případ, kdy se hodnoty vygenerují daleko od hledaného bodu. Ovšem pravděpodobnost, že tato situace nastane je velice malá. Pokud špatně zvolíme vstupní parametry, nastane situace, kdy globální minimum nelze nalést, protože oblast kam generujeme čísla, globální minimum neobsahuje. Pro tento typ funkcí, kde je jediné globální minimum a je ve středu (0,0) nemusí být lineární mapování o moc efektivnější než obyčejný algoritmus bez mapování viz. tab. 4.2. Je to způsobeno tím, že náhodně vygenerovaná čísla daleko od globálního minima mají příliš velkou fitness na to, aby je evoluční algoritmus uchoval do další generace. Ovšem toto mapování může řešit nevýhodu evolučního algoritmu bez mapování, kdy máme velký a rozměrný prostor a malý počet populace a je malá šance „trefit“ se blízko středu (0,0). Řešení je právě v lineárním mapování, zúžením definovaného (příliš velkého) prostoru viz. tab. 4.3.

V tab. 4.2 máme znázorněny výsledky jednotlivých mapování. Testování proběhlo na všech třech *de Jongových testovacích funkcích,* v tabulce jsou uvedeny pouze výsledky *První de Jongovy funkce*, protože tyto funkce jsou založeny na podobném principu, proto i výsledné hodnoty testů byly velice podobné. Test probíhal v 15. pokusech a výsledky jsou brány s přesností 0,01.

Vstupní parametry:

- populace 200
- krajní meze souřadnice *x:* -60; 60

- krajní meze souřadnice *y:* -60; 60
- pravděpodobnost mutace 0,5

První de Jongova funkce			
			lineární mapování
test č.	**bez mapování**	**log(x)**	**a=0,5 b=0,0**
1.	26	30	29
2.	15	26	23
3.	11	29	33
4.	16	31	20
5.	19	23	25
6.	25	30	20
7.	18	28	17
8.	16	33	25
9.	19	34	19
10.	17	30	21
11.	13	27	22
12.	15	31	28
13.	17	24	30
14.	18	30	25
15.	20	34	21
průměr	**17,67**	**29,33**	**23,87**

Tab. 4.2 Testování jednotlivých mapování.

Jako nejefektivnější se nám jeví evoluční algoritmus bez mapování. Kde se nalezne globální minimum už v sedmnácté generaci. Dále následuje méně efektivní lineární mapování, které nalézá globální minimum v průměru o šest generací později. Jako nejméně efektivní mapování v tomto případě vyšlo mapování pomocí logaritmu, které nalézá globální minimum v průměru až v dvacáté deváté generaci. Výsledky jsou ovlivněny vstupními parametry a důvody proč se v tomto případě mapování nezdá být efektivní byly zmíněny v předchozím odstavci.

V následující tab. 4.3 máme opět znázorněny výsledky jednotlivých mapování. Testování proběhlo znovu na všech třech *de Jongových funkcích.* Do tabulky byly zaznamenány výsledné hodnoty tentokrát *Druhé de Jongovy funkce.* Výsledky všech tří funkcí byly opět podobné. Test probíhal v 15. pokusech a výsledky jsou brány s přesností na 0,05.

Vstupní parametry:

- populace 50
- krajní meze souřadnice *x:* -100; 100
- krajní meze souřadnice y*:* -100; 100
- pravděpodobnost mutace 0,5

Druhá de Jongova funkce			
			lineární mapování
test č.	bez mapování	log(x)	a=0,25 b=0,0
1.	460	149	235
2.	163	163	104
3.	315	260	20
4.	139	121	107
5.	185	202	30
6.	102	148	20
7.	320	124	89
8.	82	205	450
9.	385	135	19
10.	460	168	31
11.	206	155	163
12.	332	190	213
13.	607	176	34
14.	77	208	17
15.	453	302	185
průměr	**285,73**	**180,4**	**114,47**

Tab. 4.3 Testování jednotlivých mapování.

Zde můžeme pozorovat výrazný nárůst efektivity evolučního algoritmu. Zatímco evoluční algoritmus bez mapování našel globální minimum v průměru až po 285. generaci, mapování logaritmem je v průměru skoro 1,5 krát rychlejší. Nejefektivnější mapování se v tomto případě jeví lineární mapování, které našlo své globální minimum už po 114. generaci.

Dalším testování proběhlo na zbývající trojici testovacích funkcí. Většinu vlastností mají tyto tři funkce podobné. Zaměřil jsem se konkrétně na *Schwefelově testovací funkci* viz. tab. 4.4. Pokud hledáme globální minimum evolučním algoritmem bez mapování, je zde šance, že se algoritmus trefí zrovna do lokálního minima, které je zároveň globálním minimem. Protože algoritmus rovnoměrně rozmísťuje body k hledání. Pro velké rozměry prostoru je také vhodné i velkou populaci (tedy velký počet bodů k hledání, tím se nám zvyšuje šance nalezení oblasti kde se nachází globální minimum).

Mapování logaritmem se ukázalo pro tento typ funkce zcela nevhodné, nenajde globální minimum právě pro svoji charakteristiku, tj. mapování na podobná čísla. Mapuje pouze na úzký rozsah čísel, protože mutací se dostane maximálně do lokálního minima.

Lineární mapování je daleko vhodnějším typem mapování pro tuto funkci, opět je zde šance, že se trefíme do lokálního minima, které bude zároveň globálním, navíc při vhodně zvolených parametrech *a* a *b* můžeme globální minimum naleznout dříve. Obecně však nemusí toto mapování

globální minimum najít vůbec. Při špatně zvolených parametrech, nenajde globální minimum v zúženém oblasti definovaného prostoru.

V tab. 4.4 jsou demonstrovány výsledné hodnoty jednotlivých mapování na *Schwefelově funkci*. Test probíhal v 15. pokusech a výsledky jsou brány s přesností na 0,05. Jak už bylo zmíněno v předchozím odstavci, ukázalo se, že naimplementované mapování logaritmem je pro tuto funkci zcela nevhodné, jelikož globální minimum nebylo nalezeno. Dále se můžeme přesvědčit, že na vhodně zvolených parametrech opravdu záleží, především u lineárního mapování. Pokud zvolíme špatný parametr *a* nebo *b*, uvázneme v lokálním minimu a globální minimum nenajdeme. Naopak pokud zvolíme parametry správné efektivita algoritmu se výrazně zvyšuje. Efektivita algoritmu je také závislá na zvoleném počtu jedinců populace.

Vstupní parametry:

- populace 200
- krajní meze souřadnice *x:* 0; 500
- krajní meze souřadnice y*:* 0; 0
- pravděpodobnost mutace 0,5

Schwefelova funkce				
			lineární mapování	
test č.	**bez mapování**	**log(x)**	**a=0,5 b=100**	**a=0,5 b=350**
1.	30	nenašel	nenašel	26
2.	18	nenašel	nenašel	11
3.	37	nenašel	nenašel	30
4.	19	nenašel	nenašel	13
5.	34	nenašel	nenašel	15
6.	45	nenašel	nenašel	15
7.	17	nenašel	nenašel	22
8.	28	nenašel	nenašel	23
9.	22	nenašel	nenašel	16
10.	24	nenašel	nenašel	16
11.	23	nenašel	nenašel	18
12.	19	nenašel	nenašel	19
13.	16	nenašel	nenašel	16
14.	36	nenašel	nenašel	15
15.	18	nenašel	nenašel	18
průměr	**25,73**	**0**	**0**	**18,2**

Tab. 4.4 Testování jednotlivých mapování.

Poslední testovanou funkcí, na kterou se zaměříme, je *Rastriginova funkce* viz. tab. 4.5. Tato funkce má jedno globální minimum a to ve středu (0,0), dále má spoustu dalších lokálních minim. Proto by zde měly platit podobné principy jako pro *de Jongovy funkce.* Mapování by mělo být hodně výhodné hlavně kolem bodu (0,0) a tedy jak logaritmus, tak i lineární mapování. Bez mapování snadno evoluční algoritmus uvázne v lokálním minimu.

Vstupní parametry:

- populace 200
- krajní meze souřadnice *x:* -60; 60
- krajní meze souřadnice *y:* -100; 100
- pravděpodobnost mutace 0,5

Rastriginova funkce			
			lineární mapování
test č.	**bez mapování**	**log(x)**	**a=0,5 b=0**
1.	30	34	13
2.	16	35	9
3.	26	38	11
4.	23	33	9
5.	22	32	15
6.	19	29	10
7.	27	40	14
8.	31	31	12
9.	32	30	15
10.	25	36	11
11.	29	29	13
12.	32	26	10
13.	18	34	7
14.	29	39	13
15.	16	27	15
průměr	**25**	**32,87**	**11,8**

Tab. 4.5 Testování jednotlivých mapování.

Jak můžeme vidět, správně zvolené lineární mapování je u této funkce velice výhodné. Globální minimum v daném případě, nachází v průměru už po 11. generaci. Což je 2 krát výkonnější než evoluční algoritmus bez mapování. Mapování logaritmem v našem případě už nevypadá tolik efektivní, ovšem nehrozí nám uváznutí v lokálním minimu.

5 Závěr

Výsledkem předložené práce je sumarizace teoretických znalostí v oblasti mapování kandidátního řešení, jejich analýza a aplikování na evoluční algoritmy. Následně pak praktická aplikace těchto znalostí v podobě demonstrativního mapovacího systému. Největším přínosem této práce je zkutečnost, že se jedná o praktickou realizaci myšlenky, která lze jen těžko dohledat v odborných pramenech.

Závěrem je nutné říci, že je důležité, jak se nastaví definovaný prostor. Zda v něm globální minimum dané funkce leží či nikoliv a nebo zda není definovaný prostor příliš veliký. Například pro Rastriginovu funkci (a některé další), které je třeba prohledávat při malém definovaném prostoru, při velkém definovaném prostoru je při mutaci velká změna a nemusí tedy najít přesné globální minimum.

Naimplementované mapování pomocí logaritmu není příliš vhodné. Domnívám se, že problém způsobuje částečné počítání náhody pro tak veliká čísla, a tedy že mapuje čísla jinam, než by teoreticky mělo. Proto je mapování pomocí logaritmu z našeho testování neefektivní. Nejen, že minimum najde ve větším počtu generací než algoritmus sám bez mapování, ale někdy minimum ani nenajde. Je totiž úzce specializované na určité místo. I kdyby se doimplementoval posunutí středu, efektivita by se dle mého názoru příliš nezvýšila. Pouze pro speciální funkce, u kterých víme nebo tušíme, kde globální minimum je.

Mapování pomocí lineární funkce je něco mezi algoritmem bez mapování a algoritmem s logaritmickým mapováním, protože pro námi zvolenou podoblast rovnoměrně hledáme. Prakticky to znamená, že hledáme podrobněji tam, kde je třeba. Tam, kde není potřeba vyhledávat, nehledáme vůbec. Proto mapování pomocí lineárních funkcí dle testů vyšlo, že zlepšuje efektivitu výpočtu. Například pro Rastriginovu funkci, globální minimum najde až dvakrát rychleji. Algoritmus bez mapování snadno uvázne v lokálním minimu a už se z něj nedostane.

Pokud máme dobře naprogramovaný evoluční algoritmus, mapování nemá vliv ani tak na efektivitu, jako spíše na schopnost řešení s větší šancí najít. Pokud je evoluční algoritmus špatně navržen, samotné mapování nepomůže, protože už je potřeba podrobnějších znalostí o daném problému a jeho řešení. Což ovšem degraduje princip evoluce, kdy hledáme neznáme řešení. Pro speciální případy může dobře zvolená mapovací funkce zvýšit efektivitu výpočtu algoritmu.

Hlavním přínosem této práce by měla být jak teoretická část, tak především praktická část, která demonstruje použití evolučních algoritmů na netriviálním reálném problému s možností systém modifikovat. Hlavní možnosti vylepšení a navázání na praktickou část práce je dle autorova názoru pokračování ve vývoji mapovacích funkcí za účelem zvýšení efektivity vyvíjeného systému.

Literatura

[1] Schwarz, J., Sekanina, L.: Aplikované evoluční algoritmy, studijní opora. FIT VUT v Brně 2006.

[2] Wikipedia: Evolutionary Algorithm. článek dostupný na elektronické adrese http://en.wikipedia.org/wiki/Evolutionary_algorithm (prosinec 2008).

[3] Kvasnička, V.,Pospíchal, J., Tiňo, P.: Evoluční algoritmy. Vydavatelství STU Bratislava, 2000, 215 s., ISBN 90-227-1377-5

[4] Wikipedia: Genetické programování. článek dostupný na elektronické adrese http://cs.wikipedia.org/wiki/Genetick%C3%A9_programov%C3%A1n%C3%AD

[5] Eiben, A. E., Smith, E.: Introduction to Evolutionary Computing. Springer Verlag, November, 2003, ISBN 3540401849. Kniha je omezeně dostupná na elektronické adrese http://www.cs.vu.nl/~gusz/ecbook/ (prosinec 2008).

[6] Sipper, M. a kol.: A Phylogenetic, Ontogenetic, and Epigenetic View of Bio-Inspired Hardware Systems. IEEE Transactions on Evolutionary Computation, 1997.

[7] Rothlauf, F.: Representations for Evolutionary Algorithms. Copyright is held by the author/owner(s).GECCO'09, July 8-12, 2009, Montreal Quebec, Canada.

[8]Floreano. D., Mattiussi. C.: Bio-Inspired Artificial Intelligence.2008 Massachusetts Institute od Technology

[9] Zelinka. I., Oplatková. Z., Šeda. M., Ošmera. P., Včelař. F.: Evoluční výpočetní techniky – Principy a aplikace. Nakladatelství BEN – technická literatura, Praha 2009, ISBN 978-80-7300-218-3

Seznam příloh

Příloha 1: CD se složkami
- doc: elektronická forma textu bakalářské práce
- src: všechny vytvořené zdrojové kódy, s programovou dokumentací
- bin: příklad přeloženého mapovacího systému