

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV AUTOMATIZACE A INFORMATIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

GRAMATICKÁ EVOLUCE - JAVA/MATLAB IMPLEMENTACE

GRAMMATICAL EVOLUTION - JAVA/MATLAB IMPLEMENTATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. KAMIL MIŠKAŘÍK
AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. RADOMIL MATOUŠEK, Ph.D.
SUPERVISOR

BRNO 2013

ZADÁNÍ ZÁVĚREČNÉ PRÁCE

(na místo tohoto listu všijte originál a nebo kopii zadání Vaš práce)

ABSTRAKT

Univerzální třída implementující gramatickou evoluci. Testováno na aproximaci funkcí a nastavení PSD regulátoru pro řízení chaotického systému Henonovy mapy.

ABSTRACT

Universal class implements grammatical evolution. Tested on approximate functions and settings PSD controller for the chaotic system Henon maps.

KLÍČOVÁ SLOVA

Gramatická evoluce, Henonova mapa, diskrétní regulátor, aproximace funkcí

KEYWORDS

Grammatical evolution. Henon maps, Approximate function, PSD controller

BIBLIOGRAFICKÁ CITACE

MIŠKAŘÍK, K. *Gramatická evoluce - Java/Matlab implementace.* Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2013. 74s.

Vedoucí diplomové práce doc. Ing. Radomil Matoušek, Ph.D.

PODĚKOVÁNÍ

Děkuji vedoucímu diplomové práce doc. Ing. Radomilu Matouškovi Ph.D. za pomoc a odborné vedení při vypracování této práce.

Rovněž děkuji rodině za trpělivost a veškerou podporu v průběhu celého studia.

ZADÁNÍ ZÁVĚREČNÉ PRÁCE **3**

ABSTRAKT **5**

BIBLIOGRAFICKÁ CITACE **7**

PODĚKOVÁNÍ **9**

SLOVNÍK **13**

1. ÚVOD **15**

2. ŘEŠENÍ OPTIMALIZAČNÍCH ÚLOH **17**

2.1 Operační výzkum 17
2.2 Matematické metody 17
2.2.1 Interpolace 17
2.2.2 Metoda nejmenších čtverců 18
2.3 Soft – computingové metody 19
2.3.1 Evoluční algoritmy 19
2.3.2 Neuronové sítě 20
2.3.3 Fuzzy logika 21

3. ZÁKLADNÍ EVOLUČNÍ ALGORITMY **23**

3.1 Genetické algoritmy 23
3.2 Hejnové algoritmy 23
3.3 Simulované žíhání 24

4. GRAMATICKÁ EVOLUCE **27**

4.1 Historie 27
4.2 Backus – Neumanova forma 27
4.3 Skladba gramatiky: 27
4.4 Přírodní předpoklady 28
4.5 Možnosti implementace 29
4.6 Změny v populaci 29
4.7 Zrychlení evoluce migračními modely 29
4.8 Postup dekódování jedince v populaci 31

5. IMPLEMENTACE V JAZYKU JAVA **33**

5.1 Popis jazyka 33
5.2 Netbeans IDE 34
5.3 Samotná implementace 34
5.3.1 Rozložení tříd 35
5.3.2 Popis hlavních tříd GE 36
5.3.3 Ostatní třídy 43
5.3.4 Tvorba gramatik 44

6. UŽIVATELSKÉ ROZHRANÍ V MATLABU **45**

7. NASTAVENÍ PARAMETRŮ EVOLUCE **51**

7.1 Výsledné optimální nastavení algoritmu 52
7.2 Female tournament 52

8. APROXIMACE FUNKCÍ **57**

8.1 Quintic 58
8.2 Sextic 60
8.3 SinusThree 62
8.4 SinusFour 65

9. DISKRÉTNÍ REGULÁTOR HENON MAP **69**

9.1 POUŽITÁ GRAMATIKA 71
9.2 VÝSLEDKY 71
10. ZÁVĚR 73
SEZNAM POUŽITÉ LITERATURY 75
PŘÍLOHY: 77

SLOVNÍK

GE – Gramatická evoluce
GA – Genetické algoritmy
HA – Hejnové algoritmy
BNF – Backus-Naurova forma
GA – Genetický algoritmus
OS – Operační systém
Java ME – Java Micro Edition
Java SE – Java Standard Edition
Java EE – Java Enterprise Edition
JVM – Java Virtual Machine – interpret Javy pro daný operační systém
OOP – Objektově Orientované Programování
GUI – Graphic User Interface
STE – Ohodnocení dynamickým pásmem
SALE – Ohodnocení lineární odchylkou
SSE – Ohodnocení metodou nejmenších čtverců
ITAE – Ohodnocení kritériem ITAE

1. ÚVOD

Řešení prohledávání stavového prostoru a nalezení optima je pro velké stavové prostory při použití klasických matematických postupů neefektivní z hlediska výpočetního času, případně přesnosti řešení v závislosti na zvolených počátečních podmínkách a parametrech. Může se stát, že řešení nenaleznou. Z tohoto důvodu se v posledních 30 letech začínají rozšiřovat metody, které hledají řešení blízké optimálnímu. Tyto metody prohledávají náhodně stavový prostor a na předpokladech z přírody směřují k řešení, které je nejblíže optimálnímu.

Tato práce se zabývá implementací GE v jazyce Java a jeho chování na základních testovacích úlohách genetických algoritmů. Tyto algoritmy se snaží napodobit přírodní výběr, při němž se výhodnější řešení problému postupuje do další fáze hledání a nahrazuje předchozí méně vhodná řešení. Základními prvky změn v populaci jsou mutace, křížení a selekce, kdy každý jedinec představuje jedno možné řešení problému. GE obsahuje oproti klasickým genetickým algoritmům gramatiku, což je prvek, který umožňuje velmi rychle změnit již jednou napsaný program jen editací gramatiky, která může být uvedena např. v externím textovém souboru. Gramatika výrazně ovlivňuje výsledky evolučního vývoje. GE je algoritmus používaný především pro prohledávání konkrétních velkých stavových prostorů. Pro správnou funkci GE je nutné dodržet BNF - Backus-Naurovu formu zápisu gramatiky. Základem BNF jsou terminály a neterminály, které se dále rozvíjejí podle daných pravidel a po rozvoji všech neterminálů dojde k vytvoření textového řetězce, který může reprezentovat řešení daného problému, např. funkce, syntéza logických obvodů, nastavení PSD regulátoru.

Při realizaci této práce byly využity technologie jazyka Java, Matlab a knihovna Matlabu – Simulink. Java umožnila vytvořit univerzální třídu implementující GE, Matlab a Simulink umožní rychlé a jednoduché ověření funkcionality na hledání funkcí i pro tvorbu modelu chaotické Henonovy mapy, a Matlab umožňuje spojení Javy a Simulinku. Výsledná data jsou exportována do textového souboru, odkud se dají v případě potřeby zpracovat v jiném programu.

V části implementace se podrobně zabývám vlastním programováním a schématem hlavní třídy implementující GE v prostředí Java, jde zde popis hlavních metod a návod na obsluhu daní třídy. Následující část Obsluha GUI popis a použití programu v Matlabu, protože Java samotná má jen třídu GE, která má konstruktor a výstupem jsou pole textový řetězců představující možné řešení hledaného problému.

Jako testovací úlohy se využila aproximace funkcí a diskrétní regulátor pro chaotický systém Henónova mapa.

2. ŘEŠENÍ OPTIMALIZAČNÍCH ÚLOH

2.1 Operační výzkum

Optimalizaci v inženýrské praxi představuje vědní obor operační výzkum. Jde využití moderních poznatků požadující nejlepší zhodnocení v oblasti řízení zdrojů, mezi kterými existují vazby. Může to být model libovolného systému např. dopravy, řízení podniku nebo zpracování zdrojů. Optimalizace zdrojů může být řešena využitím poznatků z oblasti matematiky nebo nepřímými metodami náhodného prohledávání stavového prostoru a jejich postupným zlepšováním na základě předem daných pravidel. Matematické metody řešení často selhávají u nelineárních problémů nebo neúplně zadaných počátečních podmínkách. Náhodně vyhledávání má větší možnosti se vypořádat s těmito neurčitostmi.

2.2 Matematické metody

Aproximovat funkci $f(x)$ znamená nahradit ji funkcí $\varphi(x)$, která má v měřených místech v daném intervalu hodnoty blízké $f(x)$. Píšeme $\varphi(x) \approx f(x)$, nejjednoduššími způsoby jsou interpolace a metoda nejmenších čtverců. Interpolace je aproximace, která má v požadovaných bodech přesně stejnou hodnotu jako funkce $f(x)$.

2.2.1 Interpolace

Aproximace funkce se využívá tam, kde je nutné proložit naměřené hodnoty křivkou, požadavky jsou aby funkce byla v přijatelném jednoduchém tvaru, volí se tedy nejčastěji tvar polynomu. Existenci interpolačního polynomu lze dokázat konstrukcí Lagrangeova tvaru interpolačního polynomu.

$$\mathrm{P_n(x)} = y_0 l_0(x) + y_1 l_1(x) \ldots . y_n l_n(x) = \sum_{i=0}^{n} y_i l_i(x)$$

Kde $l_i(x)$ jsou takzvané fundamentální polynomy definované předpisem

$$l_i(x) = \frac{(x - x_0)(x - x_1) \ldots (x - x_{i-1})(x - x_{i+1}) \ldots (x - x_n)}{(x_i - x_0)(x_i - x_1) \ldots (x_i - x_{i-1})(x_i - x_{i+1}) \ldots (x_i - x_n)}$$

Z podmínek lze vyvodit

$$l_i(x_k) = \pi r^2 \begin{cases} 1 \text{ pro k} = \text{i} \\ 0 \text{ pro k} \neq \text{i} \end{cases} \qquad \text{i, k} = 0,1,2 \ldots \text{n}$$

To splňuje interpolační podmínku

$$\mathrm{P_n(x_k)} = \sum_{i=0}^{n} y_i l_i(x_k) = \mathrm{y_k} \quad \mathrm{k} = 0,1,2 \ldots \mathrm{n}$$

Interpolační polynom je daty [xi,yi], i = 0,1,2 … n, určen jednoznačně. Jsou-li polynomy P a Q interpolační polynomy splňující interpolační podmínky $P(x_i) = Q(x_i) = y_i$,

pak polynom P – Q je roven nule v uzlových bodech x_0, x_1, x_2 … x_n. Polynom stupně n, nemůže mít více než n kořenů, pokud se nerovná identicky nule. V tomto případě je nutné P – Q = 0 a tedy P = Q. Tím je jednoznačnost interpolačního polynomu dokázána.

Tato metoda má nedostatky v tom, že při zvýšení počtu uzlů polynomu je nutné přepočítat všechny fundamentální polynomy. Dalším nedostatkem je poměrně velký počet početních úkonů kdy je potřeba $2n^2 + 2n$ násobení a $2n^2 + 3n$ sčítání. Tento nedostatek odstraňuje Newtonův tvar interpolačního polynomu.

$$Pn(x_n) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \ldots + n(x - x_0)(x - x_1) \ldots (x - x_{n-1})$$

Při přidání dalšího bodu $[x_{i+1}, y_{i+1}]$

$$P_{n+1}(x_n) = P_n(x) + a_{n+1}(x - x_0)(x - x_1) \ldots (x - x_n)$$

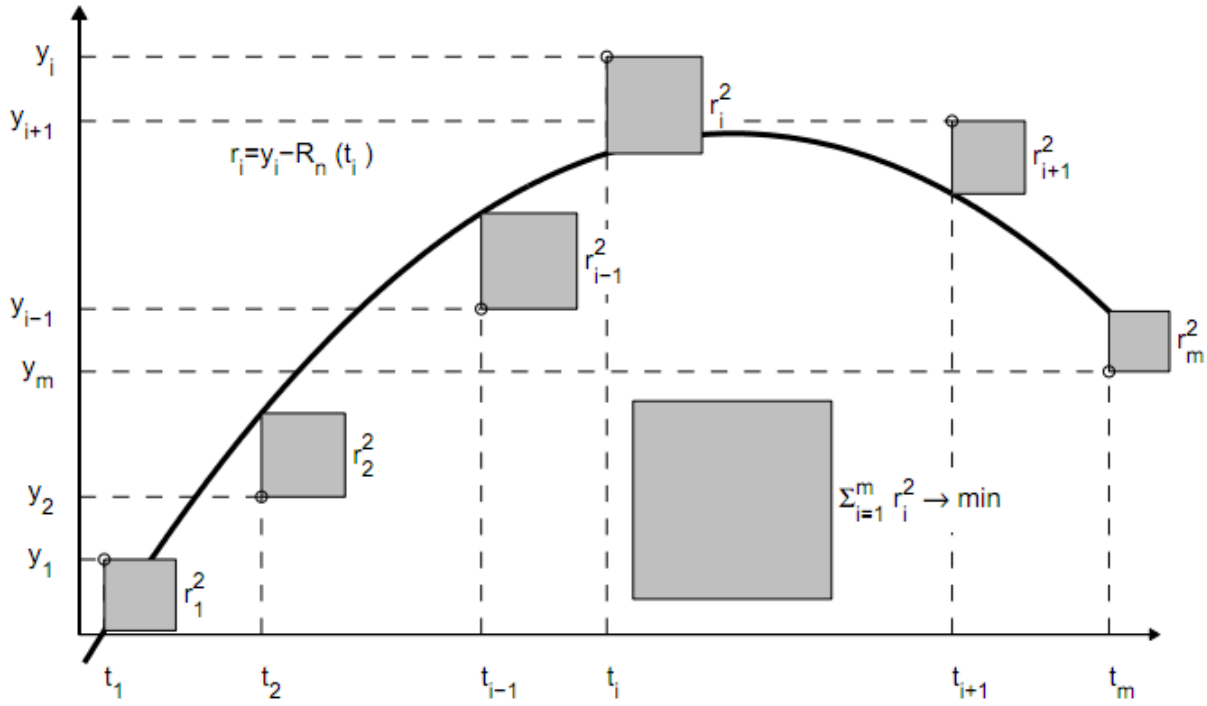
Hodnotu $z = P_n(\bar{x})$ určímě Hornerovým schematem:

$$z = a_n \; a \; pak \; pro \; i = n-1, n-2, \ldots 0 \quad z := z(\bar{x} - x_i) + a_i$$

Pro delší intervaly x na dlouhém úseku je výhodnější využít spline – rozdělení intervalu na menší úseky a prokládání polynomy nižšího stupně, protože při použití polynomů velkých stupňů dochází k velkým chybám mezi uzly.[7]

2.2.2 Metoda nejmenších čtverců

Metoda nejmenších čtverců je metoda kdy funkce $\varphi(x)$ prokládá hodnoty$[x_i,y_i]$ tak, aby suma kvadrátů vzdáleností mezi body funkce $f(x)$ a $\varphi(x)$ byla co nejmenší. Je pro ni charakteristické, že funkce φ neprochází body $[x_i, y_i]$. Označuje postup pro přibližné řešení přeurčených nebo nepřesně zadaných soustav rovnic, založený na miniaturalizaci kvadrátu jejich reziduí.



Obr. 1 Metoda nejmenších čtverců

2.3 Soft – computingové metody

Soft-computing je termín používaný v oblasti informatiky, kdy řešíme výpočetně složité úlohy, především z oblasti NP-problémů, pro které nelze přesné řešení odvodit v reálném čase. Zahrnuje oblast evolučních algoritmů, fuzzy logiky a umělé neuronové sítě. Nejdůležitějším požadavkem na funkčnost těchto metod je správně úlohu definovat, zadat požadované proměnné, omezující podmínky, podmínky konvergence, někdy se tyto metody využívají jen k přípravě počátečních podmínek pro klasické deterministické optimalizační algoritmy. Další nevýhodou těchto metod, kterou částečně řeší tato práce je nutnost přizpůsobit daný algoritmus pro konkrétní požadovaný problém. Důvodem většinou bývá snaha algoritmus zrychlit nebo snížit náročnost na výpočetní paměť na daném PC pro konkrétní problém.

2.3.1 Evoluční algoritmy

Evoluční algoritmy hledají řešení na základě Darwinovy evoluční teorie přírodního výběru, kdy jedinec s větší mírou adaptace na požadované podmínky má větší šanci přejít do další generace. Jedinec v populaci představuje jedno z možných řešení. V průběhu výpočtu se zjistí síla jedince (fitnes) a ta určuje jeho šance přežití pro další generaci a předání svých genů (řešení) do další generace. V informatice se jako evoluční algoritmus popisuje program, který má v jádru seznam jednotlivých řešení. Tato pole jsou v každé generaci podobně jako v přírodě křížena tj. v určeném místě dojde k rozdělení dvou polí a vzájemně si vymění jednu část. Další změnou v populaci je mutace, tato změna jednotlivých řešení je velmi důležitá, vnáší mezi již nalezené vhodné řešení nové prvky, které můžou mít potenciálně vyšší přínos, než již nalezené kombinace řešení. Po změnách v populaci dojde k přírodnímu výběru, každý jedinec je vyhodnocen a dostane číselnou

hodnotu, která určuje jeho vzdálenost od řešení daného problém, na příkladu u aproximace funkcí to může být vzdálenost nalezené funkce od hledané funkce. Tedy každá následná generace má lepší průměrné ohodnocení než předchozí.

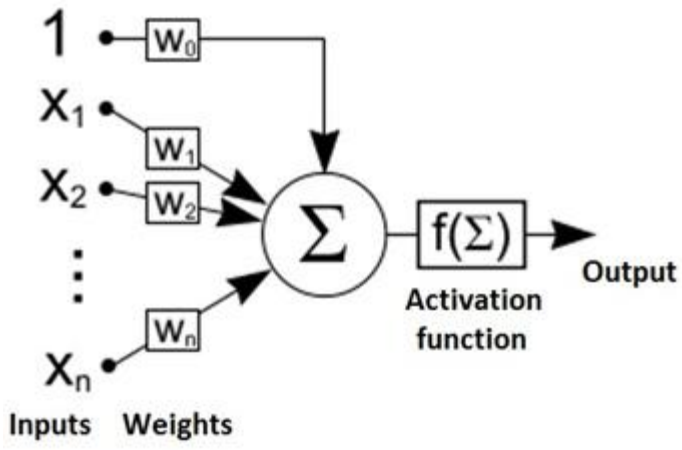
2.3.2 Neuronové sítě

Přírodním vzorem jsou mozkové neurony a jejich funkce, snímání dat a jejich následné zpracování a vyslání dalšího signálu, na obrázku 1 je vyobrazen neuron z pohledu informatiky, jde o model McCulloch-Pitts (MCP). Tento neuron má několik vstupů $x_1 - x_n$ a jeden výstup. Pokud je potenciál P generovaný ze vstupních hodnot,x a jejich váhy w dojde k vyslání signálu na výstup. Aktivační funkce by neměla být lineární, je vhodné používat například sigmoid nebo hyperbolický tangens.

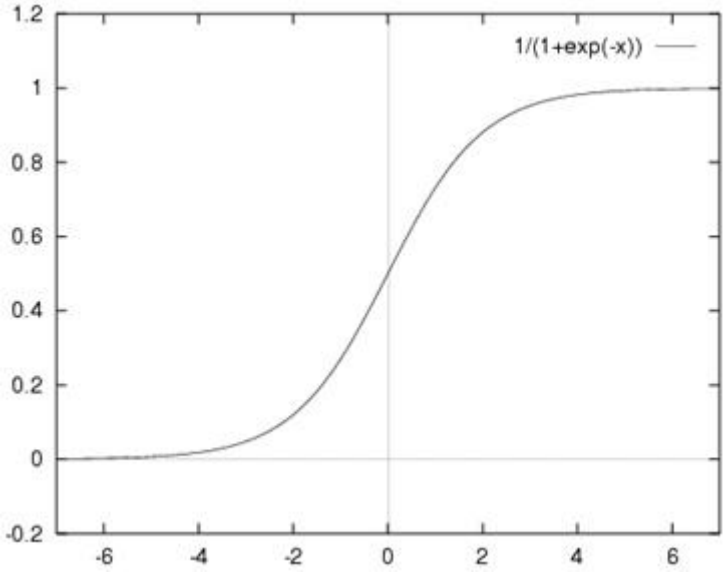
$$P = w_1x_1 + w_2x_2 + w_3x_3 \ldots w_nx_n$$

Pokud nastaven dostatečný potenciál P dojde k vyslání výstupního signal Y

$$Y = 1, jestliže\ P > W0\ jinak\ Y = 0$$



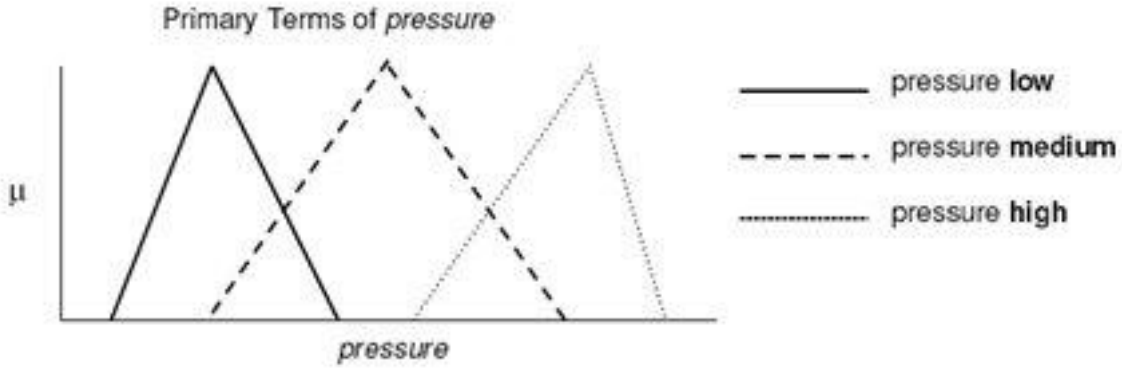
Obr. 2 McCulloch-Pits model neuronu [1]



Obr. 3 Graf aktivační funkce [1]

2.3.3 Fuzzy logika

Fuzzy logika operuje s tzv. mlhavými pojmy (fuzziness = mlhavost) tj. s pojmy, které mají méně ohraničené hranice (neostré). Jedná se např. o pojmy: menší, větší, více menší, vlažný, téměř studený apod. Tedy pohybujeme se na intervalu <0;1> … tedy prvek do dané množiny patří více či méně. Na rozdíl od klasické logiky 0 nebo 1 ve které je přesně definováno, jestli prvek do množiny patří nebo ne (tedy jsou zde jen dvě možnosti (Ano=1/Ne=0). Fuzzy logiku jinak nazýváme jako vícehodnotovou logiku. Co se týká vícehodnotové logiky, tak byla "objevena" Mr. J. Lukadiewiczem v roce 1920. Fuzzy logika a teorii množin představil Mr. Lotfi Zadeh v roce 1965. První kdo prakticky použil fuzzy logiku byl Abraham Mandani v roce 1970 (kontrolní systém pro turbíny). Fuzzy logika byla zprvu velmi rozvíjena v USA, rapidní nárůst zájmu o tuto technologii začal v Japonsku a v dnešní době se opět objevil v USA a také v Evropě. [2]



Obr. 4 příklad funkční příslušnosti tlaku ve fuzzy logice [2]

3. ZÁKLADNÍ EVOLUČNÍ ALGORITMY

3.1 Genetické algoritmy

Genetické algoritmy (GA) se používají asi od 60 let 20 století, Autorem byl J. Holland a jeho tým na Michiganské univerzitě. Tehdy se jejich použití omezovalo hlavně na umělou inteligenci a až v počátku 90 let, se začaly využívat pro řešení úloh operačního výzkumu, případně pro optimalizaci jiných problémů praxe, např. tvarování strojních součástí.

Název GA se používá kvůli analogii s mezi reprezentací složité struktury vektoru složek a genetickou strukturou chromozomu. V případě šlechtění živých organismů se v dalších vývoji prosazují jedinci s požadovanou vlastností, která je dána kombinací rodičovských chromozomů. U zrodu GA byla myšlenka, že lze najít výhodnější řešení problému kombinací dvou již nalezených řešení. Chromosom označujeme jako genotyp, jde o seznam genů jedince, které jsou kombinovány do tzv. kodonů, každý kodon a jeho stav určuje jednu část nebo vlastnost jedince – tzv. fenotyp. Manipulace s jedinci v populaci a jejich chromosomy je dána několika možnými úpravami. Nejdůležitější jsou selekce, křížení a mutace. Selekce probíhá tak, že se porovnává hodnota fitness různých jedinců, a jedinec s vyšším ohodnocením postupuje do další generace. Křížení kombinuje dva různé řetězce a vytváří třetího jedince – potomka. Mutace je proces, při kterém se invertuje daný počet genů v chromosomu, čímž dojde také ke změně genotypu jedince. Standardně se jako fitness používá kladná hodnota a cílem je maximalizovat tuto hodnotu. GA pracují tak, že nejprve se vytvoří náhodná populace p s počtem jedinců m, a pak se v dané populaci vytváření nové generace při použití selekce, mutace a křížení. GA je ukončena v případě, že dosáhneme požadované hodnoty fitnes, vyprší určený výpočetní čas nebo se vygeneruje požadované množství generací.

Z principu GA vyplývá, že jsou velmi závislé na vygenerování počáteční populace. Proto je výhodné u nich využívat metod heuristiky a již první generaci částečně nasměrovat k řešení. Tímto způsobem, ale vnášíme riziko uvíznutí algoritmu v některém lokálním optimu, které může být výrazně vzdáleno od globálního optima. Velikost populace musí být dostatečně velká, aby dostatečně pokryla prostor řešení, ale zároveň co nejmenší, aby se zkrátil požadovaný výpočetní čas. Z experimentálních zkušeností vyplývá, že dostatečná velikost populace je kolem 30, resp. mezi n až 2n, kdy n je délka bitového řetězce. [3]

3.2 Hejnové algoritmy

Rojová inteligence je inspirována chováním živočichů, kteří žijí v hejnech, např. mravenců, včel, ptáků či ryb. Jednotlivci jsou v kolonii bez nějakého přímého řízení rozdělováni podle požadavků k potřebným činnostem. U mravenců jsou to třeba průzkumníci, ochránci mraveniště, uklízeči, atd. Každý jedinec provádí jen jednoduché akce, ale každý přispívá ke správnému fungování systému.

Pro správnou funkci celé kolonie je nutné, aby se lokální informace co nejlépe zpracovala v daném místě bez okamžitého ovlivňování dalších částí kolonie. Tím, že se lokálně mění chování části hejna, dochází postupně i ke změnám chování celku. Tento způsob samoorganizace vytváří dočasné stabilní struktury, některé můžou přetrvat delší dobu, jiné zaniknou po pár generacích, a jiné se periodicky opakují, nebo pohybují po kolonii. Ty nejlepší lokální optima jsou dále posilována zpětnou pozitivní vazbou, dokud

některé z nich výrazně převáží a stane se hlavním optimem kolonie. Tento způsob hledání je velmi náchylný pro uvíznutí v lokálních extrémech. Proto se v HA vyskytuje 5-15 procent náhodných jedinců, kteří hledají v prostoru výhodnější pozice.

Fungování algoritmu je ovlivněno pozitivní zpětnou vazbou, negativní zpětnou vazbou, zesílením změn a vícenásobnou interakcí. Pozitivní zpětná vazba podporuje vznik struktur, jedinec s výhodnější hodnotou, k sobě přitahuje další jedince, a ti stahují zase další. Dochází ke zvýšení hustoty řešení v okolí lokálního optima. Negativní zpětná vazba tato vazba vyvažuje zpětnou vazbu a přesouvá jedince od méně výnosných řešení k těm, která dávají lepší hodnotu. Zesilování změn jsou již zmínění náhodní hledači nových lokálních optim. V přírodě například mravenec opustí již nalezenou cestu a teoreticky může najít kratší. Vícenásobná interakce je proces, kdy je jedinec schopen vytvářet stabilní samoorganizační strukturu díky interakcím s ostaními jedinci, kteří danou strukturu dále posilují.

Využití hejnových algoritmů zahrnuje ekonomické modely chování zákazníků, fyzikální modely, matematické systémy jako jsou například celulární automaty, v informatice jejich použití zahrnuje plánování tras robotů, rozpoznávání tvarů, rozmístění dobíjecích stanic pro autonomní roboty ve skladech a spoustu dalších optimalizačních problémů spojených s rozmístěním prvků pro optimální pokrytí prostoru.[4]

3.3 Simulované žíhání

Základní myšlenky metody spadají do roku 1953, když při definici pojmu žíhání – změně mřížky a struktury slitin na základě rychlosti ochlazování. Při simulaci procesu bylo na tuhnoucí materiál nahlíženo jako na systém částic. Simulované žíhání pak algoritmicky popisuje změnu teploty v systému, dokud není dosaženo zchlazeného stavu. O 30 let později byly formulovány myšlenky využít tento algoritmus k hledání přípustných řešení optimalizačního problému s cílem zajistit konvergenci k optimálnímu řešení. Mapování systému chlazení na systém optimalizace je znázorněn v následující tabulce číslo 1.

Termodynamický simulace	Kombinatorická optimalizace
Stav systému	Přípustné řešení
Energie	Hodnota účelové funkce
Změna stavu	Přechod k sousednímu řešení
Teplota	Řídící parametr
Zmrazený stav	Heuristické řešení

Řešení optimalizačního problému na základě simulovaného žíhání – Tab. 1

Navržený přístup lze považovat za hledání lokálního minima, při kterém je dovoleno jít zpátky na horší řešení. Frekvence kroků je řízena jistou pravděpodobnostní funkcí, která se při běhu výpočtu mění. Základem výpočtu jsou myšlenky termodynamiky, podle které je při teplotě t pravděpodobnost růstu energie o δE dána rovnicí

$$\boldsymbol{p(\delta E) = e^{-\frac{\delta E}{Kt}}}$$

K je Boltzmannova konstanta. Během algoritmu se generuje změna stavu systému a počítá se odpovídající změna energie, při poklesu energie se přejde do nového stavu, při růstu energie je stav akceptován podle vzorce. Tento postup se opakuje při každé teplotě dokud neproběhne požadovaný počet iterací, po dokončení se teplota sníží. Proces probíhá tak dlouho, dokud algoritmus nezamrzne na stabilním řešení. Parametr t je pouze řídící parametr, který nemá fyzikální obsah. Konstanta K by tak mohla být vypuštěna z pravděpodobnosti přijetí. Většinou se ale t označuje jako teplota a způsob snižování jeho hodnoty jako plán chlazení systému.[3]

4. GRAMATICKÁ EVOLUCE

Patří do skupiny genetického programování. Je pro ni charakteristická gramatika v BNF formě, která ovlivňuje výsledek dekódování jedince, který představuje možné řešení.

4.1 Historie

Jedná se o jednu z nejnovějších metod GP (Ryan a kol., 1998), (O'Neill a Ryan, 2001). GE je podobná genetickému programování rozšířené o gramatiku. Proti GP má výhodu v tom, že je obecnější a je použitelná pro hledání programů v libovolném jazyce, pokud dodržíme Backus-Naurovu formu - BNF formu zápisu gramatiky. S tím je spojena reprezentace jedinců, kdy na rozdíl od stromů v GA využívá přímo binární řetězce. [5].

4.2 Backus – Neumanova forma

BNF je systém zápisu bezkontextové gramatiky. Jde o metasyntaxi používanou pro popis formálních jazyků. Autoři John Backus a Peter Naur vytvořily bezkontextovou gramatiku pro jazyk ALGOL. BNF se využívá pro zápis gramatik počítačových programovacích jazyků, sad instrukcí a komunikačních protokolů ale také jako notace zastupující části gramatik skutečných jazyků. [5]

Základním kamenem gramatiky jsou terminály a neterminály. Terminal je dále rozvíjen, dokud se z něj nestane neterminál. Dalším důležitým prvkem je startovní symbol a pravidla transformace neterminálu na další stav.

4.3 Skladba gramatiky:

G = {N, T, P, S}

N – množina neterminálů

T – množina terminalů

P – množina pravidel transformací stavů

S – startovní symbol

Forma zápisu BNF jsou produkční pravidla:

<symbol neterminálu > ::= <výraz se symboly>

Chceme-li např. identifikovat jednoduchou funkci sin(2x) v daném rozsahu využijeme gramatiku pravidel:

(First):((ZN)((Fcn)(OP)(Enum))

(Fcn):((sin(Enum))):((cos(Enum))):((sin(Enum))/(cos(Enum)))

(Enum): ((Num)(OP)(Var))

(Var):(X)

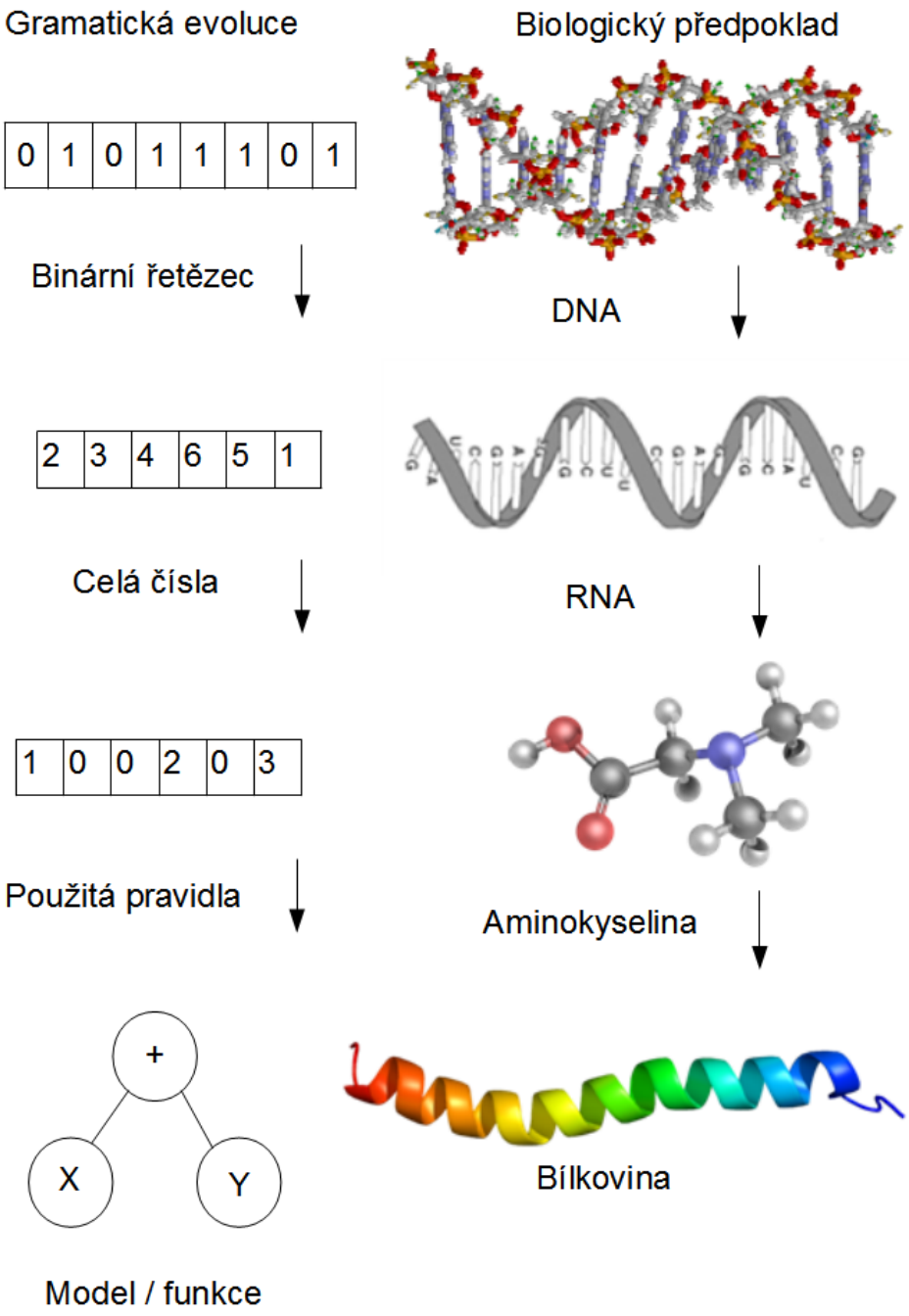
(ZN):+:-

(OP):+:-:/:*

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

4.4 Přírodní předpoklady

GA vycházejí z Darwinovy evoluční teorie. Napodobují přírodní výběr mezi jednotlivými jedinci a lepší jedinec získává právo pokračovat do další generace. V oblasti optimalizace jedinec představuje možné řešení a je ohodnocen z požadovaného hlediska problému (časového hlediska, celková odchylka od hledaného řešení např.).



Obr. 5 Porovnání gramatické evoluce a biologických předpokladů

4.5 Možnosti implementace

V GE reprezentuje jedince nejčastěji vektor nul a jedniček, který se v průběhu ohodnocení převede na celá čísla, V této práci je využit jako základní prvek reprezentující bitové pole objekt ze třídy Colt vytvořené v Cernu ©. [6]

4.6 Změny v populaci

Selekce

Je základním prvkem množiny změn v populaci. V této práci je využita turnajová selekce, při které se náhodně vybere zvolený počet jedinců a mezi nimi probíhá výběr nejlépe ohodnoceného jedince, který postupuje do další generace. Tento způsob nezaručuje postup nejlepšího jedince do další generace, proto je vždy nejlepší jedinec vyjmut ze staré populace a vložen do nové. Poté proběhne turnajová selekce a po jejím skončení se v nové generaci nahradí nejhorší jedinci nejlepšími z původní populace, tzv. elitismem.

Druhou možností je pravděpodobnostní selekce, každý jedinec má procentuální šanci se dostat do nové generace a jeho šance je tím větší čím je větší jeho ohodnocení.

Křížení

Křížení je změna jedinců v populaci výměnou části jejich binárních řetězců. Křížení může probíhat v náhodném bodě, ale protože většinou dojde k příliš destruktivnímu výsledku, náhodné křížení se příliš nepoužívá a v této práci ani není implementováno. Lepších výsledků dosahuje křížení strukturální, kdy se vyměňují celé kodony a podstromy jedinců.

Mutace

Mutace invertuje jeden bit v genetické informaci jedince, čímž může, ale taky nemusí dojít ke změně fenotypu. Mutace je náhodná, a celková míra mutace – počet genů, které zmutují by neměla přesáhnout 2 procenta, při vyšších hodnotách dochází k ničení již nalezených struktur a degeneraci řešení.

Elitismus

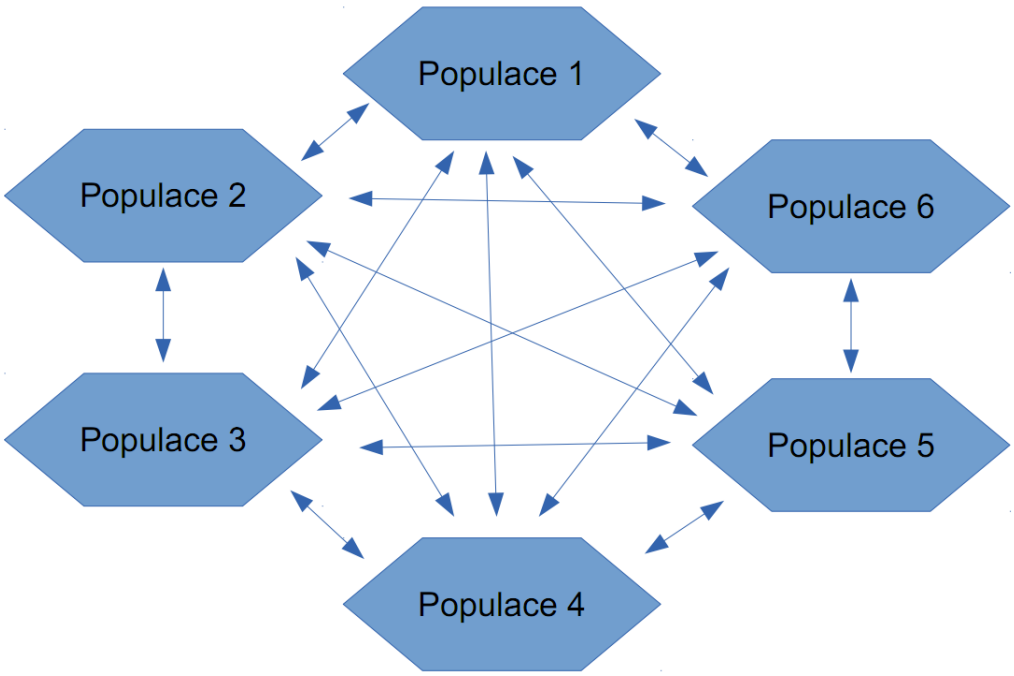
Elistismus je systém ukládání nejlepších již nalezených řešení, protože při selekci nelze zaručit přechod nejlepších jedinců do dalších generací, případně může dojít vlivem mutací a křížení k destrukci nejlepšího řešení.

4.7 Zrychlení evoluce migračními modely

V případě této práce nejsou přímo využity metody paralelních populací, třída samotná však obsahuje metody, které umožňují předat daný počet jedinců do jiné instance stejné třídy, která umí dané jedince přijmout a vložit do své populace. Automaticky vkládá jedince na náhodná místa v populaci. Způsoby migrace mezi jednotlivými populacemi jsou okružní migrace, kdy se jedinci přesouvají jen mezi sousedními populacemi po kružnici. Při tomto způsobu se zachovává velká míra rozmanitosti mezi jednotlivými populacemi, které spolu přímo nesousedí. Nalézá více rozmanitých řešení ale po delším čase. Migrace náhodná, kdy se přesouvají jedinci náhodně mezi populacemi dochází k rychlejší konvergenci, ale populace jsou si velmi podobné. Nevýhody obou způsobů migrace lze vyřešit zavedením hlavní populace, kdy nedochází k migraci mezi jednotlivými populacemi, ale mezi hlavní populací jednotlivými populacemi. V hlavní

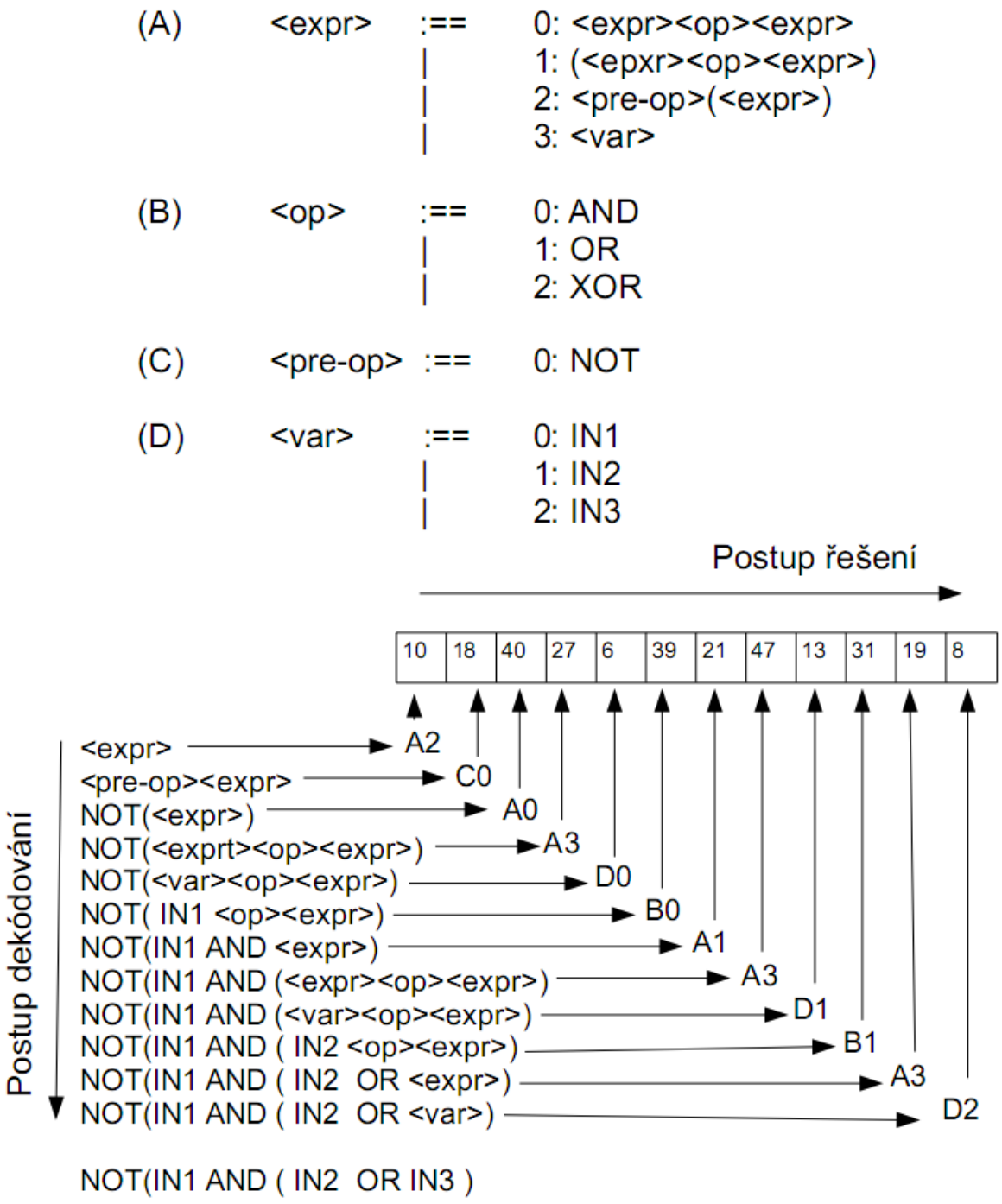
populaci jsou shromážděna nejlepší řešení z jednotlivých populací. Konvergence k řešení je tímto způsobem výrazně urychlena.

Dalšího urychlení evoluce lze dosáhnout zavedením samčích a samičích populací. Rozdíl mezi samčí a samičí populací je dán mírou rozmanitosti a procentuální mutací jedinců. V hlavní populaci se kombinují nejlepší řešení z obou populací. Samičí populace obsahuje vysokou míru rozmanitosti danou tím, že se sleduje při vkládání, jestli už daný jedinec v dané populaci existuje, pokud ano, do populace se nevkládá. Míra mutace v této populaci je velmi nízká nebo spíše nulová. Každý jedinec je v dané populaci pouze jednou. Selekce v této populaci probíhá na základě délky života = počtu generací a fitnes hodnoty. Samčí populace obsahuje velkou míru mutace až kolem 30 procent, pokud je zachována nízká mutace na strukturálnch genech – to jsou terminály. Pro nejlepší jedince je mutace nedestruktivní, tedy pokud se má mutovat jedinec, který je ve skupině elitismu, zařazuje se do populace jeho klon. Při stagnaci postupu řešení se počet jedinců postupně zvětšuje, až dojde k posunu v postupu řešení. Šanci na postup do další generace má každý jedinec, i ten nejhorší, i když jen velmi malou většinou méně než 0.02%. [8]



Obr. 6 Náhodná migrace

4.8 Postup dekódování jedince v populaci



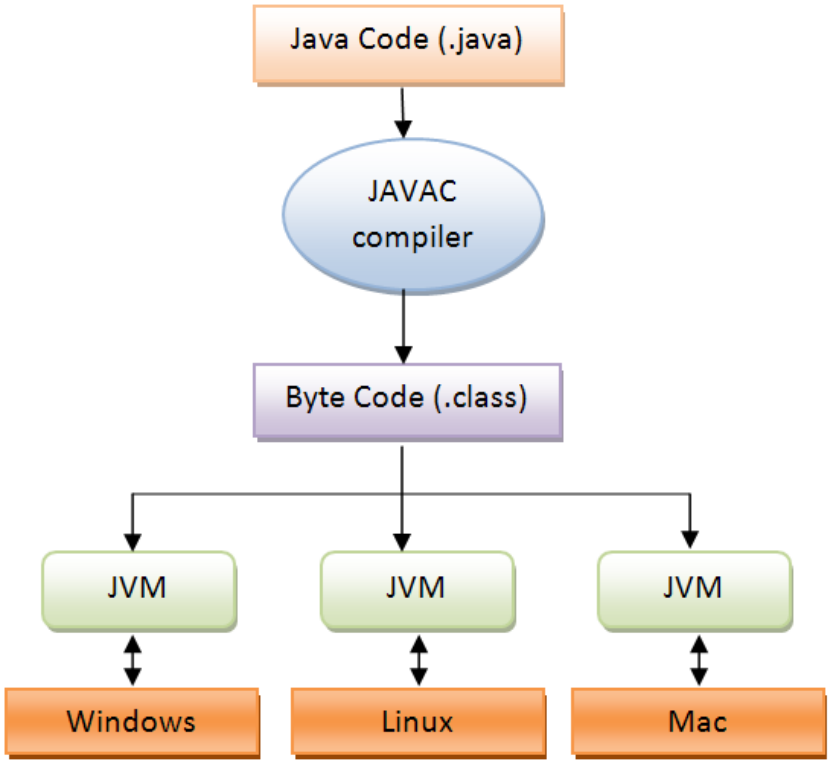
Obr. 7 Proces dekódování jedince

5. IMPLEMENTACE V JAZYKU JAVA

5.1 Popis jazyka

Java je programovací jazyk syntaxí podobný C# nebo C++. Pro běh programu je nutný JVM – Java Virtual Machine – interpret programu, určený pro danou platformu operačního systému. Java nevytváří spustitelný soubor, ale jen mezikód, který se v interpretu znovu přeloží pro konkrétní OS. Historie jazyka začíná v roce 1995, kdy společnost Sun Microsystem představila jako nový objektově orientovaný jazyk, v roce 2007 byl uvolněn zdrojový kód a dále je distribuován jako OpenSource. Této výhody ve velkém využívají banky, mobilní telefony, některé balíky kancelářských programů. Jazyk Java má několik různých platforem, JavaCard pro platební karty, Java ME pro mobilní telefony, Java SE pro použití na stolních počítačích a Java EE pro využití v informačních systémech. Jednou napsaná aplikace je v kombinaci s příslušným JVM spustitelná na kterémkoliv OS. Problém nastává při pokusu o spuštění vyšší verze na nižším interpretu, který nemusí obsahovat všechny potřebné funkce pro běh aplikace. Rozšíření jazyka není jen díky aplikacím a nízké ceně, ale hlavně díky množství možností využití např. java aplety pro internetové bankovnictví, výborná možnost propojení s Oracle databází od roku 2009, když firma Oracle koupila Sun Microsystem.

Starší verze Javy byly výrazně pomalejší než stejná aplikace napsána v C++ z důvodu horší správy paměti a nucenému překladu mezikodu při spouštění aplikace. V současnosti již není výrazný rozdíl mezi Javou, C# a C++. JVM využívá technologie správy paměti známou jako “Garbage collector”. Funguje tak, že jakýkoliv objekt nebo proměnná je z paměti odstraněna ihned jak na něj přestane odkazovat jiný objekt. Dalším urychlením startu aplikace je systém překladu “Právě včas” pro překlad jen zrovna potřebné části mezikódu.



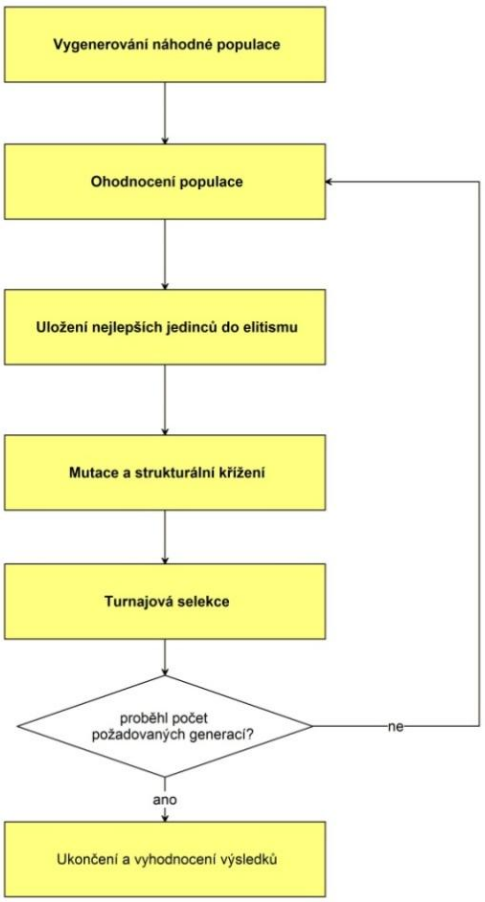
Obr. 8 Struktura fungování jazyka Java [9]

5.2 Netbeans IDE

NetBeans IDE 7.3 je vývojové prostředí integrující spoustu užitečných nástrojů pro efektivný vývoj software. Proti Eclipse obsahuje velké množství nástrojů pro podporu vývoje už v základní instalaci jako jsou profilery pro sledování využití procesoru. Netbeans umožňuje psát jak desktopové, tak mobilní či webové aplikace. Plná verze nepodporuje jen Javu, ale i jiné jazyky jako PHP, C++ či Javascript.

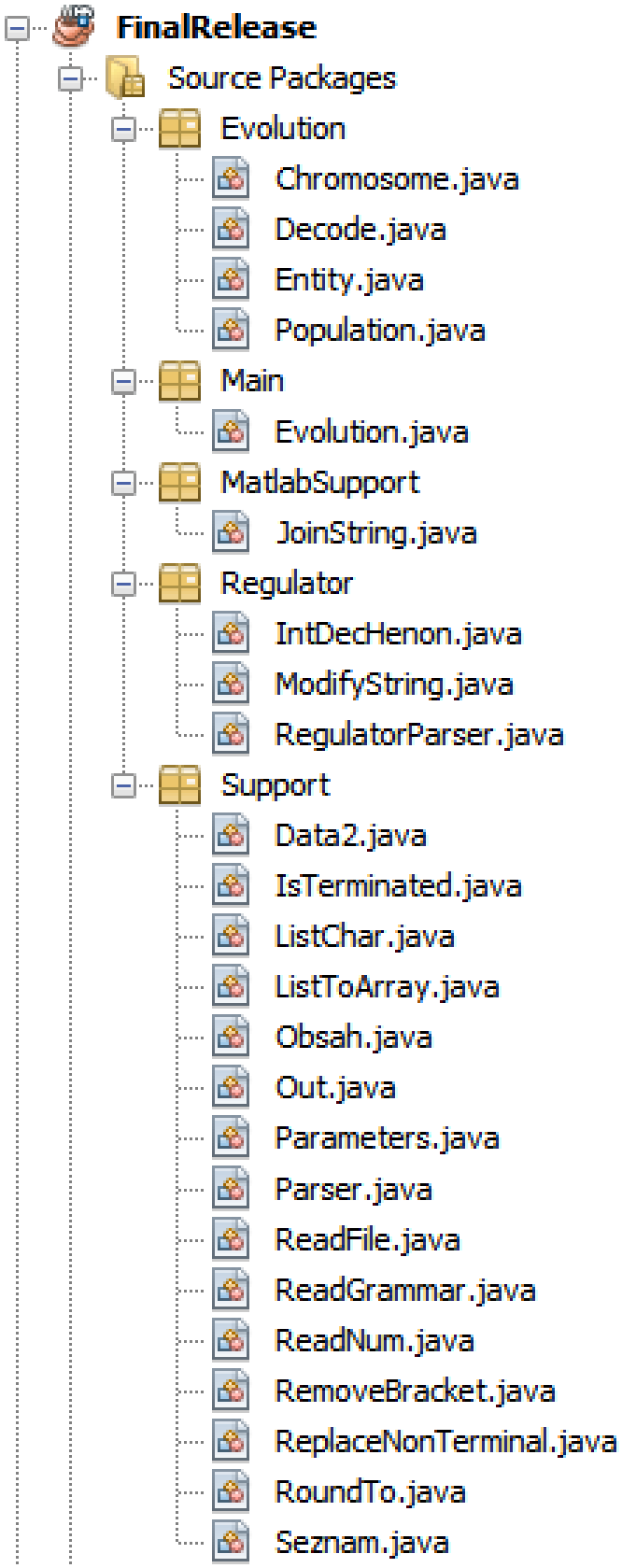
5.3 Samotná implementace

Základní třídou je Main.Evolution(). Konstruktor této třídy vyžaduje jinou třídu Support.Parameters(). Ta má prázdný konstruktor, ale má 15 Set metod a 15 get metod na nastavení jednotlivých parametrů evoluce. Tento systém je výhodnější z hlediska možného využití při paralelních populacích, kdy se nastaví jen jedna instance třídy Parametres() a ta se využije k inicializaci několika instancí evoluce. Je taktéž jednodušší měnit jeden parametr pro novou instanci Evolution, než kopírovat a znovu nastavovat všechny parametry znovu. Třída evolution() je jen schránka se základním přístupem ke třídě Population(), se kterou komunikuje a volá její metody.



Obr. 9 Postup vyhodnocení populace

5.3.1 Rozložení tříd



Obr. 10 Struktura programu

5.3.2 Popis hlavních tříd GE

Main.Evolution	
Evolution(Parameters p)	Vstupem konstruktoru je objekt třídy Paramaters, která obsahuje všechny potřebná data pro běh aplikace, odkud a jak se naplní není důležité, je to prostředník mezi GUI, které muže být z Matlabu, Javy nebo jiným spolupracujícím programem.
GetBest() java.lang.String	Vrací jedince s nejlepším fitnes – jeho textovou reprezentaci.
GetBestValue() double	Vrací nejlepší fitnes v populaci
GetDecodedPopulation() java.lang.String[]	Vrací pole jedinců v populaci, kterým je potřeba pomocí hodnotící funkce přiřadit fitnes
GetFittnes() double[]	Vrací hodnoty fitnes všech jedinců
GetReadNum() int[]	Vrací pole hodnot podle počtu čtení jedinců.
GetUndecoded() Int	Vrací počet selhání dekódování jedinců, pokud dojde k překročení počtu čtení jedince.
NextGeneration1() void	Najde jedince s nejvyšším fitnes a toho uloží, následně vyplní podle pole jedinců pro elitismus, a následně provede mutaci a stromové křížení v populaci
NextGeneration2() void	Provede turnajovou selekci a zařazení elitismu do další generace.
SetFittnes(double[] values) void	Nastavuje fitnes pro populaci, hodnoty jsou generovány externí třídou / skriptem, v tomto případě ohodnocuje Matlab
GetMigration(int pocet) Entity[]	Vrací daný počet jedinců k migraci

SetMigration(Entioty[] migrace) void	Nastaví dané jedince jako členy nové v populaci
Evolution.Chromosome	
Chromosome(Chromosome vstup) Chromosome (int BitNumbers, int Kodons)	Konstruktor přijme buď jiný chromosom a zkopíruje ho, nebo nastaví nový chromosom podle daných parametrů kodonů a počtu bitů, je nutné zachovat pravidlo, že počet bitů = celočíselný násobek kodonů.
GetBitvector() cern.colt.bitvector.BitVector	Vrací pole jedniček a nul daného jedince
GetKodonValue() int[]	Vrací hodnoty jednotlivých kodonů jedince
ChangeFirstPart (cern.colt.bitvector.BitVector a) void	Zamění část bitvektoru jinou částí – slouží pro křížení
MutateBit() void	Změní náhodný bit v jedinci
ReplaceMe(Chromosome vstup) void	Nahradí daného jedince jiným jedincem
SetRandomBits() void	Resetuje veškeré bity v jedinci, a přidá jim vysokou zápornou hodnotu fitness
Evolution.Decode	
Decode(Chromosome chrom, java.util.List Grammar,int ReadNum, java.lang.String UnReaded) void	Třída zajišťuje dekódování daného jedince, vstupem je daný jedinec, list gramatiky, počet povolených čtení jedince a řetězec, kterým se v případě překročení počtu čtení jedince nahradí jeho hodnota
GetCteni() int	Vrací počet čtení jedince

GetReaded() int	Zjišťuje, jestli se daný jedinec přečetl úspěšně
GetWord() string	Vrací dekódované slovo určené k ohodnocení.

Evolution.Entity	
Entity(Entity other)	zkopírování jedince
Entity(int bits, int kodons)	Vytvoření nového jedince podle zadaných parametrů
GetChromosom() double	Navrací chromosom daného jedince
GetValue() int	Navrací fitnes jedince
GetWord() String	Vrací slovo reprezentující daného jedince
ChangePart(cern.colt.bitvector.BitVector b) void	Vymění část chromosomu jedince za jinou
ReplaceMe(Entity other) void	Nahrazení jedince jiným jedincem, včetně všech jeho interních vlastností
ResetChromosom() void	Resetuje v jedinci bitvektor na náhodné bity
ResetValue() void	Resetuje hodnotu jedince na - 1000000
SetValue(double value) void	Nastaví jedinci danou hodnotu fitnes
SetWord(java.lang.String word) void	Nastaví jedinci daný string jako jeho reprezentaci.

Evolution.Population	
Population(Parameters par)	Ze třídy parameters si nastaví všechny potřebné údaje. Celkem 15 hodnot.
AddFemale(Entity[] female) void	Vloží do populace nového jedince
Decode() void	Dekóduje celou populaci na textové řetězce reprezentující jednotlivá řešení.
Elite() void	Najde elitní jedince pro fitnes a jednoho top a uchová je do následující generace.
GetCteni() int[]	Vrací počet čtení jedinců
GetDecoded() java.lang.String[]	Vrací pole dekódovaných jedinců k ohodnocení
GetEntity(int pocet) Entity[]	Vrátí zadaný počet jedinců k migraci
GetFittnes() double[]	Vrací fitnes jedinců populace
GetMax() Entity	Vrací jedince s nejlepším fitnes
GetUndecoded() int	Vrací počet nedekódovaných jedinců
Change() void	Provede mutaci, a strukturální křížení
CheckFemale(Entity e) boolean	Metoda vrací informaci o tom, jestli je v populaci daný jedinec přítomen
Mutate() void	Provede mutaci v populaci
ReplaceEntity(Entity[] pole)	Vloží jedince do populace na náhodná místa
SetValues(double[] values)	Nastaví dané populaci ohodnocení

Tournament()	Provede turnajovou selekci
Support.Parameters	
GetBits() int	Vrací počet bitů jedince
SetBits(int b) void	Natavuje počet bitů jedince
GetElitismus() int	Vrací počet elitních jedinců
SetElitismus(int s) void	Nastavuje počet elitních jedinců
GetGrammar() java.util.List	Vrací gramatiku řešení
SetGrammar(java.lang.String s) void	Nastavuje gramatiku řešení
GetKodons() int	Vrací počet kodonů jedince
SetKodons(int k) void	Nastavuje počet kodonů jedince
GetMutation() double	Vrací mutaci jedince
SetMutation(double m) void	Nastavuje mutaci jedince
GetPopulation() int	Vrací počet jedinců v populaci
SetPopulation(int p) void	Nastavuje počet jedinců v populaci
GetReadNum()	Vrací počet čtení jedince

int	
SetReadNum(int sp) void	Nastavuje počet čtení jedince
GetSelection() int	Vrací hodnotu turnaje
SetSelection(int s) void	Nastavuje hodnotu turnaje
GetStruct() int	Vrací počet křížení
SetStruct(int s) void	Nastavuje počet křížení
GetUnRead() java.lang.String	Vrací slovo reprezentující nedekódovaného jedince
SetUnReaded(java.lang.String s) void	Nastavuje slovo reprezentující nedekódovaného jedince
Support.Data2	
AddNejlepsi(double value, int generace, int test) void	Vloží hodnotu do matice výsledků
AddSLE(double hodnota, int generace, int test) void	Vloží SLE hodnotu do matice výsledku
AddString(java.lang.String string, int generace, int test) void	Vloží textovou reprezentaci dekódovaného jedince
GetAllNejlepsi() double[][]	Vrací matici nejlepších dosažených výsledků
GetAllSle() double[][]	Vrací matici všech SLE výsledků
GetAllString()	Vrací matici textových reprezentací nejlepších jedinců

java.lang.String[][]	
GetAritmetic() double[]	Vrací aritmetický průměr z matice nejlepších jedinců
GetAritmeticSLE() double[]	Vrací aritmetický průměr z matice SLE
GetBestAritmetic() int[]	Vrací počet jedinců kteří mají hodnotu lepší než 80 procent průměru v dané generaci napříč všemi populacemi
GetMedian() double[]	Vrací medián fitnes v jednotlivých generacích
GetnejlepsiHodnota() double	Vrací nejlepší dosažený fitnes
GetNejlepsiJedinec() java.lang.String	Vrací nejlepšího jedince ze všech nalezených
Save(java.lang.String[] pole) void	Uloží data do souboru
SetFile(java.lang.String s) void	Nastaví cestu do souboru určeného k uložení dat
SetInterval(java.lang.Double[] pole) void	Nastaví počet testů
SetNejlepsi(java.lang.Double[][] pole) void	Nastaví matici nejlepších jedinců
SetNejlepsiString(java.lang.String[][] pole) void	Nastaví matici textových reprezentací nejlepších jedinců.
SetSLE(java.lang.Double[][] pole) void	Nastaví matici nejlepších SLE odchylek

5.3.3 Ostatní třídy

Balík Support obsahuje třídy podporující samotnou evoluci, jejich určení je zjednodušit přístup a práci s proměnnými v hlavních třídách evoluce. Uvádím zde jen seznam názvů a funkcí. Většinou jde o několikařádkové objekty fungující s jednou vstupní proměnnou a jednou výstupní.

Balík Support

RemoveBracket – Odstraní závorky ze stringu

Cutter – Rozřízne string z třídy data a umožní jej vykreslit a zpracovat

Data2 – zde se ukládají postupně nalézané hodnoty a řešení

IsTerminated – určuje jestli je již daný string nahrazen a neobsahuje terminály

ListChar – pomáhá ve třídě RemoveBracket nahrazovat závorky

ListToArray – Umožňuje pracovat s listem listů, jako by se jednalo o pole

Parameters – hlavní konstrukční třída

Parser – rozděluje daný string

ReadFile – načte po řádku celý textový soubor

ReadGrammar – Zpracuje gramatiku do použitelného formátu

ReadNum – upravuje výstup a počet čtení jedinců v populaci

ReplaceNonterminal – Nahrazuje terminály neterminálním symbolem

RoundTo – zaokrouhlování

Seznam – implementace seznamu a práce s ním

Balík Regulátor

IntDecHenon – nastavení Ti Td a r pro diskretní regulátor

ModifyString – uprava stringu do použitelné podoby

RegulatorParser – rozdělení stringu pro použití v regulátoru

Balík MatlabSupport

JoinString – spojení dvou stringu do jednoho

5.3.4 Tvorba gramatik

Program akceptuje pouze gramatiku vytvořenou podle zde uvedeného postupu. Systém tvorby odpovídá BNF ale přesto je nutné dodržet níže uvedená pravidla.

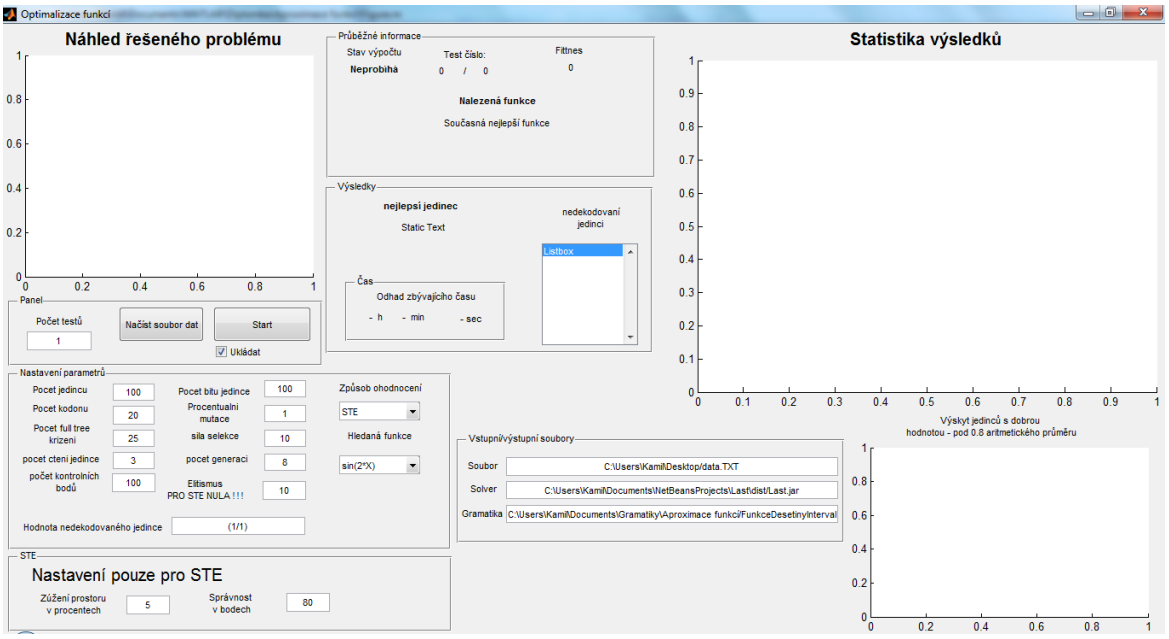
1) Terminály a neterminály odděluje dvojtečka.
2) Existují klíčová slova: Int a Dec. Int je hodnota integer, a double je desetina v intervalu 0 + 1/2^n * hodnota kodonu kdy *n* je délka kodonu.
3) Číslo tedy vytvoříme pomocí Int+Dec pokud fitnes funkce umí sčítat, nebo formátem Int_Dec v případě že neumí, jako např. diskrétním regulátoru, a pomáhá speciální třída Support.IntDecHenon(string) .
4) Pro každý problém je nejlepší napsat vlastní parser na daný string reprezentující jedince
5) Rozvoj stavu Int a Dec je nutný napsat ve tvaru: (Int) :(Int) a (Dec):(Dec) jinak nebude aplikace fungovat

Ukázka diskrétní gramatiky pro Henon map:

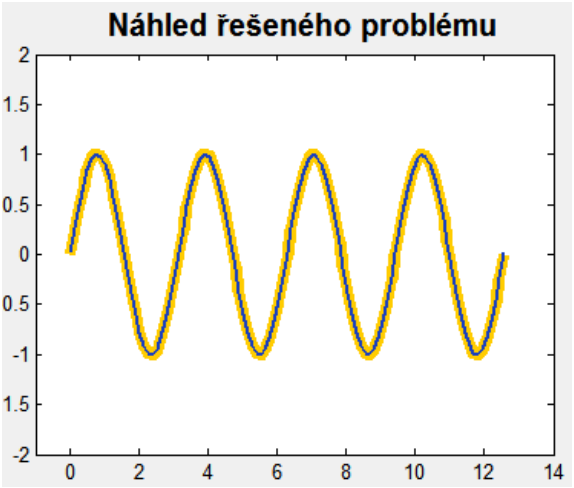
(First):((Ti)/(Td)/(R))
(Ti):((Int)_(Dec))
(Td):((Int)_(Dec))
(R):((Int)_(Dec))
(Int):(Int)
(Dec):(Dec)

6. UŽIVATELSKÉ ROZHRANÍ V MATLABU

Software Matlab je vysoko-úrovňový programovací jazyk od firmy MathWorks, umožňující numerické výpočty, zpracování obrazu a vyhodnocování simulačních modelů. Další výhodou je snadnější tvorby GUI a možnosti rychlé tvorby a editace modelů v jeho toolboxu Simulink. Název Matlab je zkratkou slov Matrix laboratory. Veškeré proměnné jsou v Matlabu považovány za matice a stejným způsobem se s nimi pracuje. Z toho vyplývají určitá omezení, přesněji nutné úpravy pro práci s proměnnými při komunikaci mezi Javou a Matlabem. Například při spojení dvou datových matic obsahující string, nevznikne matice o jednom prvku s jedním stringem, ale matice o dvou prvcích datového typu string. Tento problém je řešen v Balíku MatlabSupport ve třídě JoinTwoString. Při generování rovnic je také nutné dát pozor na počet závorek ve vygenerované rovnici. Matlab dokáže vyhodnotit maximálně 32 vnořených závorek. Programovací prostředí Matlabu je intuitivní zaměřené pro rychlé vytvoření modelů bez hlubších znalostí programování. Vytvořené proměnné nemají pevný typ, do proměnné lze vložit typ double, hodnotu poslat do jazyka Java a v další části programu do této proměnné uložit datový typ string a vložit jej do jiné metody v Javě. Tato filozofie na jedné straně urychluje vývoj aplikací, cenou za toto urychlení je větší riziko vzniku chyb, které se velmi špatně odhalují a mohou výrazně ovlivnit výsledek aplikace, která při běhu nevykazuje chyby. Strukturu programu lze dělit do souborů s příponou *.m. Soubor může být např. funkce nebo třída. Třídy v Matlabu umožňují využít výhod OOP, jako je zapouzdření, které snižuje výskyt chyb u globálních proměnných. [12]



Obr. 11 Obslužné prostředí v Matlabu



Obr. 12 Náhled problému

Tento graf zobrazuje žlutou barvou hledaný problém – funkci, nebo požadovaný výstup systému Henónovy mapy . Modrá křivka je současné nejlepší nalezené řešení.

Způsoby ohodnocení

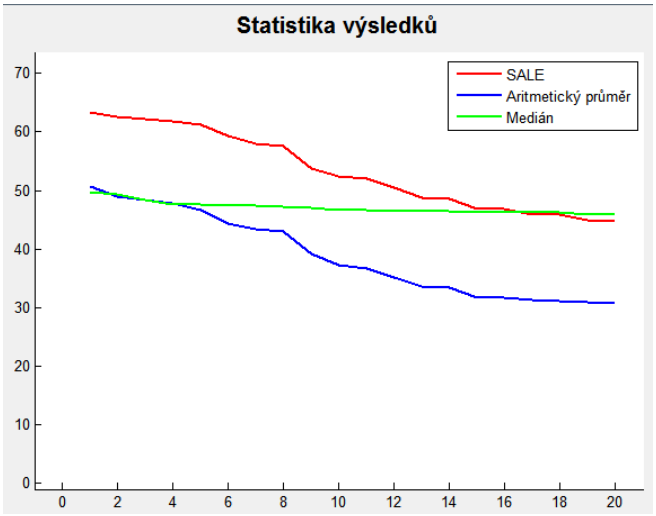
STE – dynamické pásmo – pokud funkce leží v daném intervalu v daném počtu bodů, rozměr pásma se zmenší o dané procento.

SSE – kvadratická odchylka – kvadrát rozdílu mezi funkční hodnotou a hodnotou z funkce nalezené řešením

SLE – Lineární odchylka

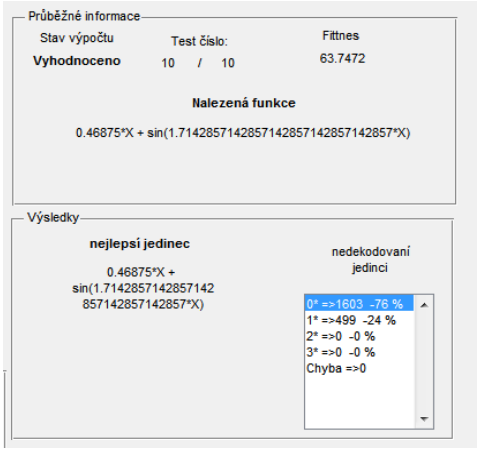
ITAE – Integrální kriterium

$$ITAE = \int_0^{\infty} t * |y(t) - y(\infty)| \, dt$$

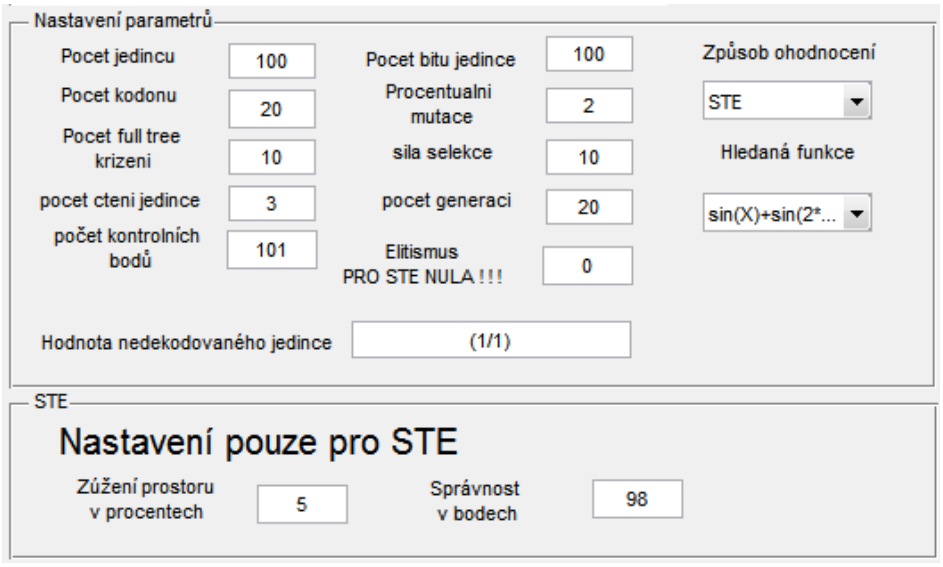


Obr. 13 Statistika výsledků

Zde jsou zobrazeny po dokončení simulace dosažené výsledky, v průběhu simulace se zobrazuje nejlepší nalezená hodnota, list výsledků nedekódovaní jedinci ukazuje počet a procentuální hodnoty čtení jedinců, nejvýhodnější je, když jsou všichni jedinci přečtení během prvního čtení, v případě vícenásobného čtení při mutaci dochází k velmi výrazným destrukčním změnám. Poslední záznam chyba je počet a procento jedinců, které se nepovedlo dekodovat ani v daném počtu násobného čtení. Pokud tento případ nastane dojde k totálnímu resetu hodnot jedince na náhodné hodnoty a jeho textová reprezentace je nastavena na standardní hodnotu, která je zadána při startu evoluce.


Průběžné informace
Stav výpočtu
Vyhodnoceno
Test číslo:
10 / 10
Fittnes
63.7472
Nalezená funkce
0.46875*X + sin(1.714285714285714285

Nalezená funkce se dynamicky mění podle aktuálně nalezené nejlepší funkce. Nedekódovaní jedinci je kontrola kolikrát došlo ke čtení jedince. V případě složitějších problémů nemusí stačit délka jedince k jeho dekódování, pokud k tomu dojde, jedinec se začne číst podruhé od začátku. Kolikrát k tomuto případu dojde, se zobrazí v této tabulce. Po vyčerpání možností čtení jedince se jedinec resetuje a zde se započítá jako chyba.



Obr. 15 Vstupní parametry

Počet jedinců – velikost populace samostatných řešení

Počet kodonů – na kolik informačních celků se bude dělit jedinec

Počet full tree křížení – kolikrát v populaci bude probíhat křížení celých podstromů

Počet čtení jedince – omezení aby se nezacyklilo čtení jedince

Počet kontrolních bodů – v kolika bodech se bude hodnotit interval funkce

Počet bitů jedince – kolik informačních bitů je v jednom jedinci

Procentuální mutace – míra mutace v jedinci

Síla selekce – kolik jedinců vstoupí do turnaje

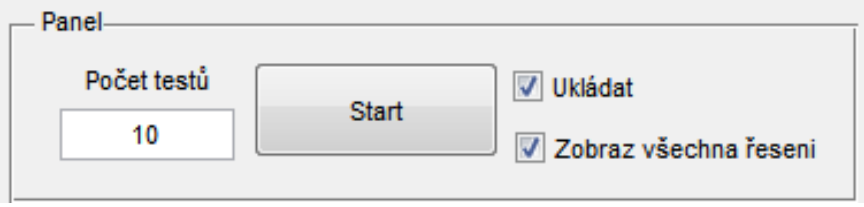
Počet generací – kolik generací bude simulace generovat v jednom testu

Elitismus – kolik nejlepších jedinců bude vybráno a přesunuto do následující
Pro STE ohodnocení vyhodnocení je nutné nastavit nulu.

Hodnota nedekódovaného jedince – tato hodnota se nastaví, pokud dojde k překročení počtu čtení jedince a tato hodnota se použije k výpočtu fitnes jedince, protože v tomto případě není jedinec správně dekódován.

Nastavení pouze pro STL

Tyto hodnoty jsou využity jen při STL ohodnocení, při tomto nastavení je bezpodmínečně nutné zachovávat nastavení elitismus na hodnotě 0. Jinak nelze při zúžení intervalu dosáhnout toho, že elitní jedinci z minulé širší generace, kteří již nesplňují podmínky vypadnou z dalšího vývoje.



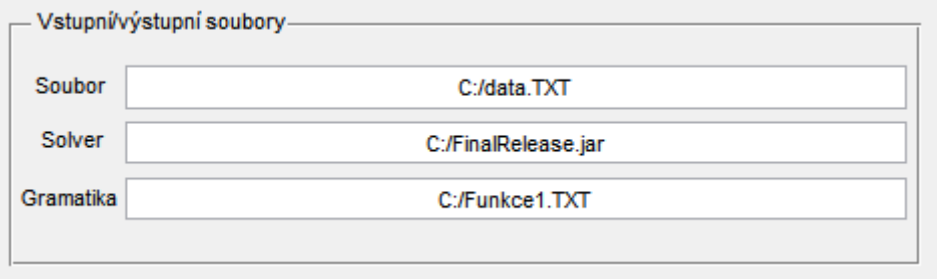
Obr. 16 Spuštění aplikace

Počet testů – kolikrát se bude opakovat generování populací

Start – spustí aplikaci

Ukládat – Uloží výsledky do TXT souboru

Zobraz všechna řešení – po vygenerování všech testů se zobrazí všechna řešení, pokud se nezvolí tato možnost, vykreslí se jen to nejlepší nalezené řešení.

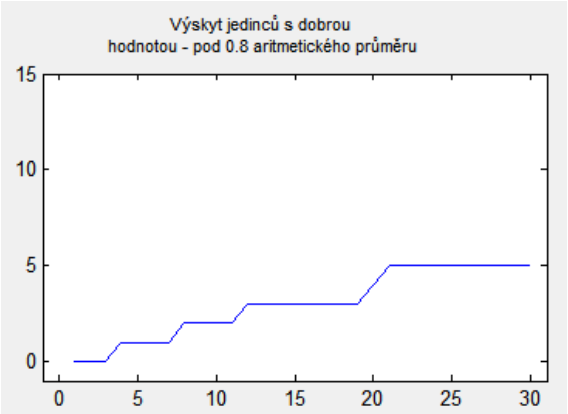


Obr. 17 Vstupní a výstupní soubory

Soubor – kam se budou ukládat výsledky

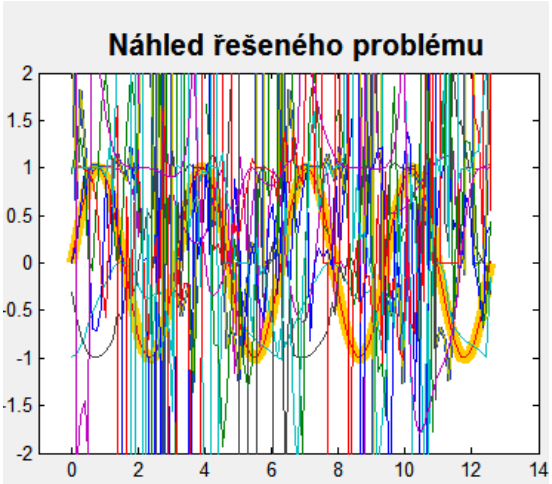
Solver – umístění k java knihovně GE

Gramatika – umístění TXT souboru s gramatikou



Obr. 18 Výskyt lepších jedinců

Graf výskyt jedinců ukazuje, ve které generaci se začnou vyskytovat výrazně lepší jedinci než je 0.8 aritmetického průměru celé populace.

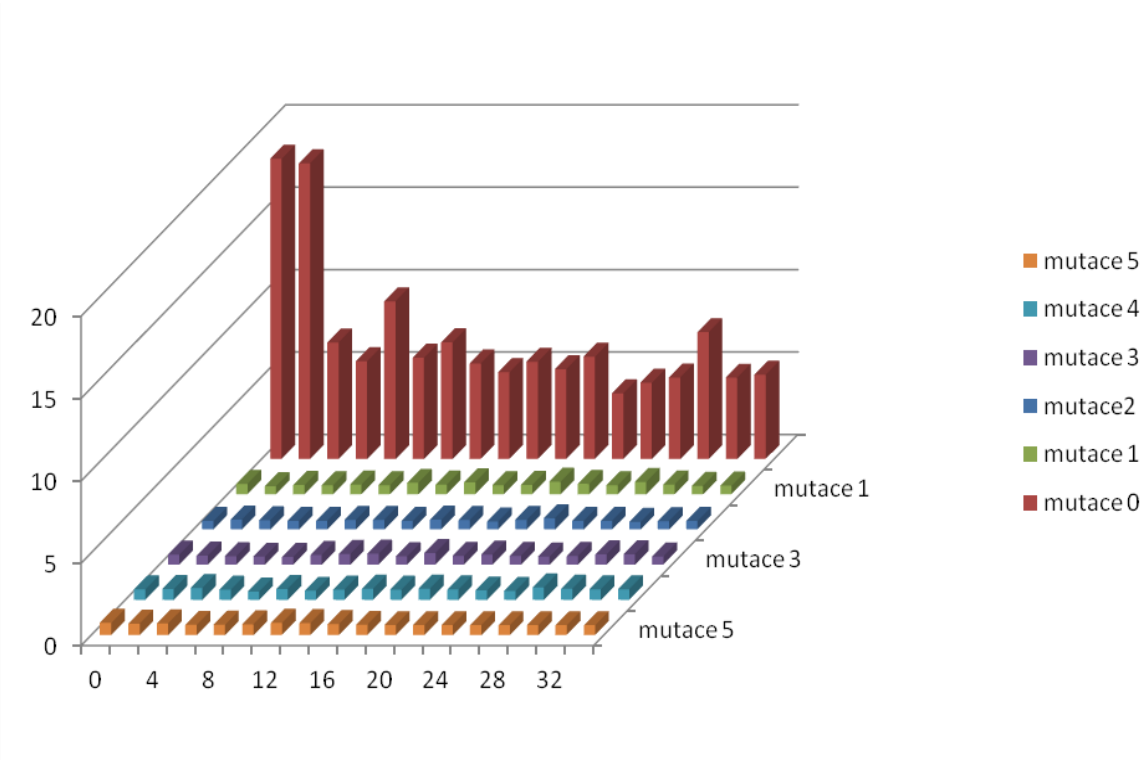


Obr. 19 Nesprávně sestavená gramatika

Pokud se použije správně sestavená gramatika, ale se špatně sestavenou návazností terminálů a neterminálů, aplikace bude fungovat, ale nebude dávat použitelné výsledky, v takovém případě je nutné upravit gramatiku.

7. NASTAVENÍ PARAMETRŮ EVOLUCE

Pro optimální fungování algoritmu je potřeba nastavit vstupní parametry systému. Hodnoty mutace a počet křížení velmi ovlivňují výsledek. Měření probíhalo na populaci 100 jedinců a 20 generací a testoval jsem vliv velikosti mutace a křížení na velikost odchylky od funkce $sin(2x)$. Výsledky měření ukazuje následující graf a celková tabulka hodnot. Měření sleduje vliv mutace od nuly do 5 procent, efekt křížení od nuly do 34 procent a následné kombinace.



Obr. 20 Vliv křížení a mutace na výslednou hodnotu

křížení / mutace	0	1	2	3	4	5	Aritmetický průměr
0	18,22	0,62	0,51	0,6	0,64	0,74	3,56
2	17,95	0,49	0,59	0,55	0,65	0,68	3,49
4	7,06	0,56	0,56	0,51	0,72	0,71	1,69
6	5,93	0,55	0,51	0,48	0,62	0,62	1,45
8	9,56	0,58	0,51	0,47	0,5	0,63	2,04
10	6,15	0,55	0,57	0,58	0,65	0,68	1,53
12	7,08	0,69	0,58	0,64	0,55	0,74	1,71

14	5,80	0,58	0,52	0,67	0,62	0,72	1,49
16	5,27	0,72	0,59	0,51	0,67	0,67	1,41
18	5,90	0,54	0,57	0,71	0,61	0,62	1,49
20	5,45	0,57	0,48	0,53	0,68	0,72	1,41
22	6,20	0,76	0,58	0,61	0,63	0,62	1,57
24	3,99	0,63	0,68	0,53	0,57	0,59	1,17
26	4,63	0,55	0,51	0,49	0,54	0,76	1,25
28	4,94	0,74	0,52	0,53	0,75	0,55	1,34
30	7,70	0,58	0,46	0,62	0,67	0,69	1,79
32	4,93	0,51	0,5	0,63	0,64	0,62	1,31
34	5,12	0,52	0,52	0,49	0,64	0,62	1,32
Aritmetický průměr	7,33	0,60	0,54	0,56	0,63	0,67	1,72

Tabulka zobrazujicí výsledky měření Tab. 2

7.1 Výsledné optimální nastavení algoritmu

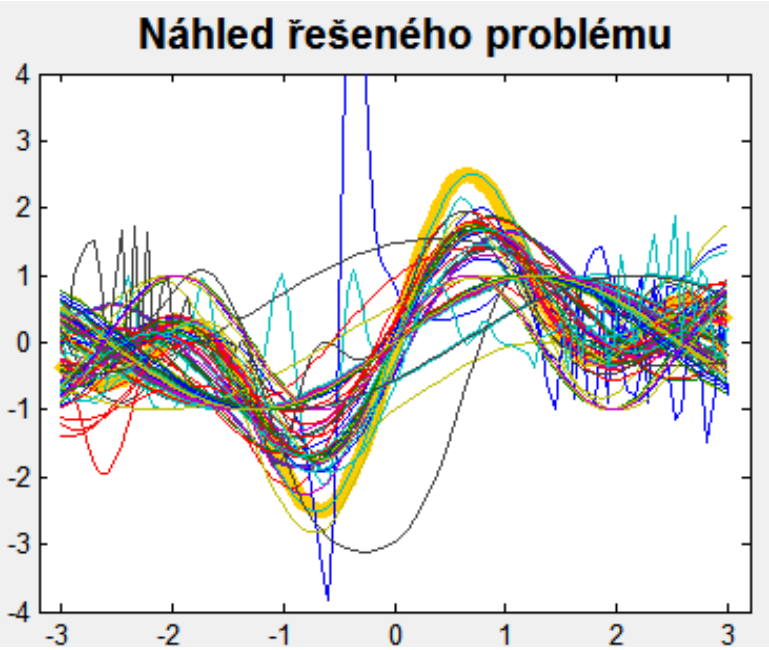
Ze změřených výsledků volím nastavení 20 procent strukturálního křížení a 2 procenta mutace.

7.2 Female tournament

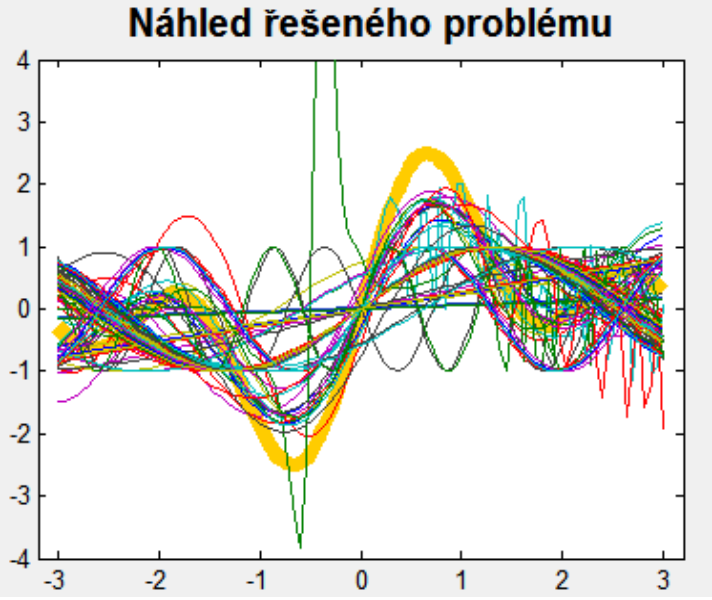
Female tournament se od klasického turnaje liší tím, že při postupu jedince do další generace se testuje, jestli již daný jedinec v populaci je. Pokud již v populaci je, vloží se jeho zmutovaná verze. Tímto způsobem se zvyšuje mutační procento u jedinců, kteří jsou v populaci vícenásobně obsazeni. Tabulka obsahují výsledky vzorku 200 měření a 50 generací. Vlastností female tournamentu je výrazné snížení statistických rozptylů měřených výsledků. Cenou za tuto vlastnost je snížení kvality řešení. Aritmetický průměr těchto řešení stoupá, ale medián hodnot se drží na stejné hodnotě.

STE	standard	female	rozdíl %
Aritmetický průměr	77.78	83.46	+7,3
modus	86.26	86.80	-
medián	86.04	86.06	+0.02

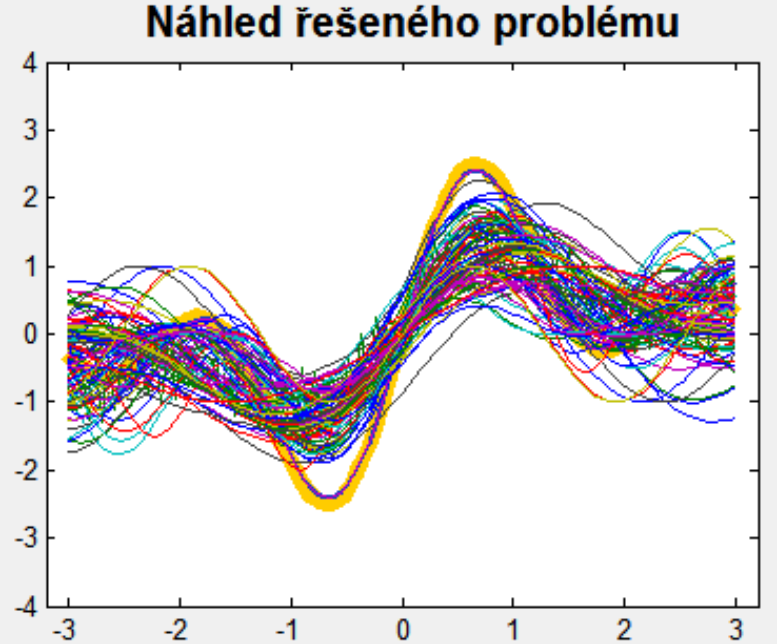
rozptyl	351.29	146.57	-125
odchylka	18.76	12.11	-54.91
průměrné zůžení	2.17	2.29	+5.52
počet zůžení	12.5	11.08	+12.81



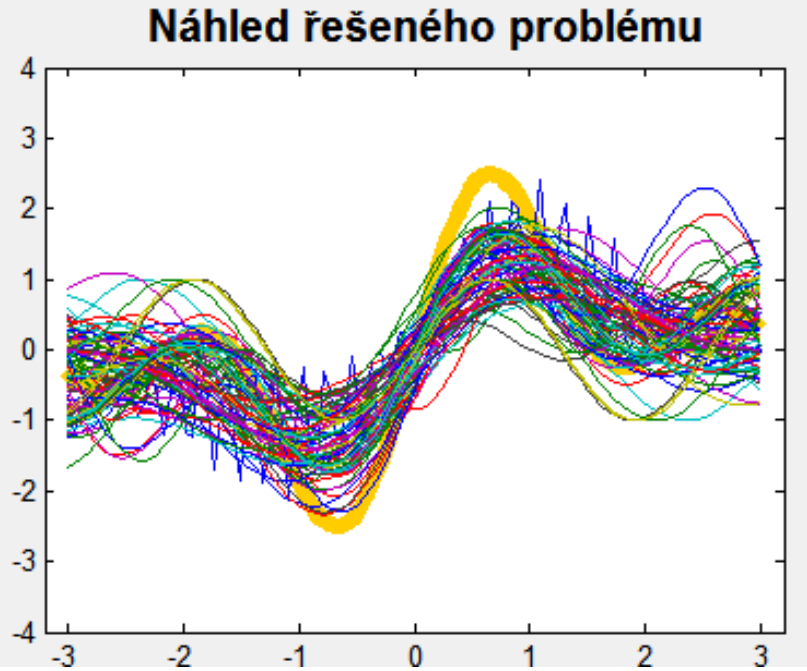
Obr. 21 200 testů tournament STE



Obr. 22 200 testů female STE-F



Obr. 23 200 testů standard tournament Sse



Obr. 24 200 testů female tournament Sse

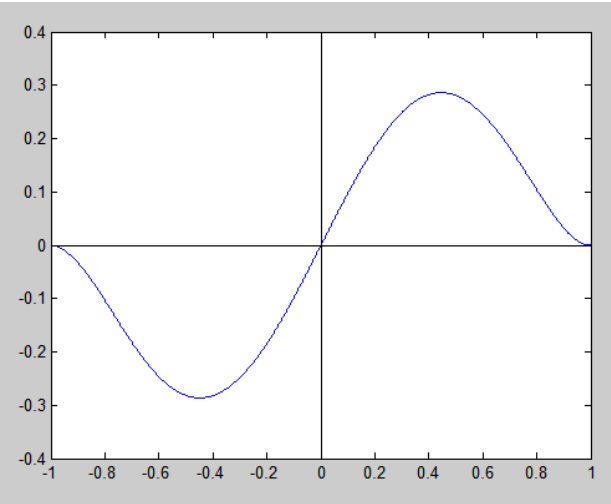
SSE	standard	female	rozdíl %
Aritmetický průměr	73.42	72.67	-1.02
modus	85.30	85.63	-
medián	84.02	84.05	0
rozptyl	274.50	307.90	+10,84
odchylka	15.56	17.54	+11,28

8. APROXIMACE FUNKCÍ

Aproximace funkcí se v oblasti genetických algoritmů využívá velmi často. Jde o jednoduchý způsob jak ověřit schopnost algoritmu najít požadované řešení. Využívají se již známé funkce Quintic, Sextic, SinusThree a SinusFour. Testování porovnává vygenerovanou náhodnou populaci jedinců, kdy jeden jedinec obsahuje samostatné řešení pro matematickou funkci a určí přesnost tohoto řešení podle zvolené metody kvadratické odchylky nebo dynamickým pásmem.

Velikost populace	250
Počet full tree křížení	50
mutace	2 %
Počet čtení jedince	3
Počet kontrolních bodů	100
Počet bitů jedince	100
Síla selekce	10
Počet generací	30
Elitismus	10
Nedekodovaný jedinec	1/1
Počet testů	20

8.1 Quintic



Obr. 25 Graf funkce x^5 – 2x^3 + x

Použitá gramatika

(First):((Fcn)(OP)(Enum)):((ZN)(Enum)):(Enum)

(Fcn):((sin(Enum))):((cos(Enum))):((sin(Enum))/(cos(Enum))):(Int)

(Enum):(((Num)(OP)(Var))(OP)((Var)(OP)(Num)))(OP)(Fcn):((Num)(OP)(Var))(OP)(Fcn)

(Var):(X):1

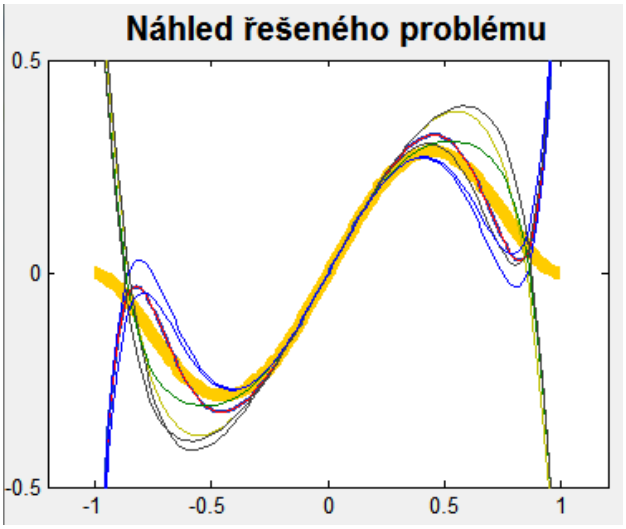
(ZN):+:-

*(OP):+:-:/:**

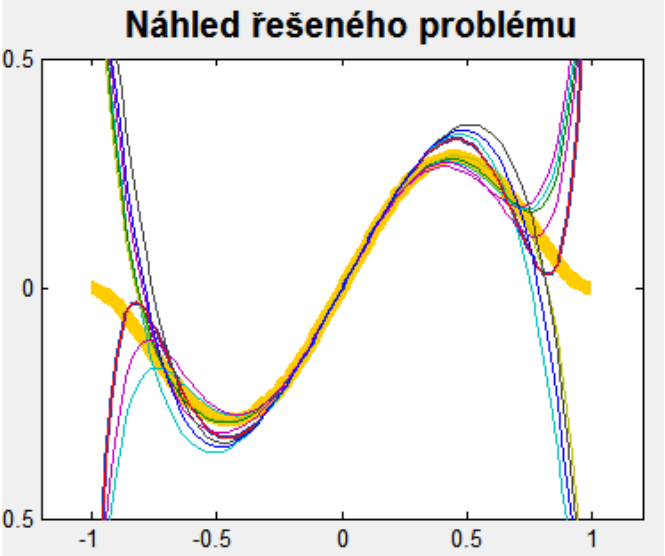
(Num):((Int)+(Dec)):(Int):(Dec)

(Int):(Int)

(Dec):(Dec)



Obr. 26 SSE



Obr. 27 STE

	SSE	*STE*
A:průměr SLE	*10.27*	*11.87*
Modus SLE	*9.08*	*10.42*
Medián SLE	*10.72*	*11.62*
Rozptyl SLE	*6.10*	*13.96*
Odchylka SLE	*2.47*	*3.73*
Průměrné zůžení	-	*1.75*
Počet zůžení	-	*16.15*
Nejlepší hodnota SLE	*9.08*	*8.85*

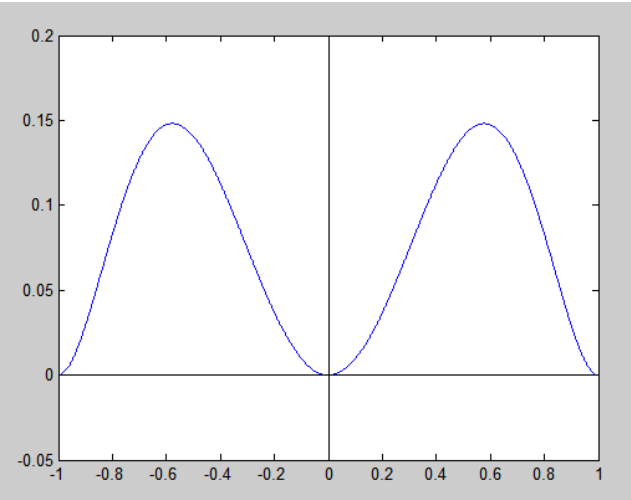
Nejlepší jedinec pro SSE

$$x\text{^}9 + x\text{^}7 - 2x\text{^}3 + x$$

Nejlepší jedinec pro STE

$$3x\text{^}9 - 2x\text{^}5 - x\text{^}3 + x$$

8.2 Sextic



Obr. 28 Graf funkce x^6 – 2x^4 + x^2

Použitá gramatika

(First):((Fcn)(OP)(Enum)):((ZN)(Enum)):(Enum)

(Fcn):((sin(Enum))):((cos(Enum))):((sin(Enum))/(cos(Enum))):(Int)

(Enum):(((Num)(OP)(Var))(OP)((Var)(OP)(Num)))(OP)(Fcn):((Num)(OP)(Var))(OP)(Fcn)

(Var):(X):1

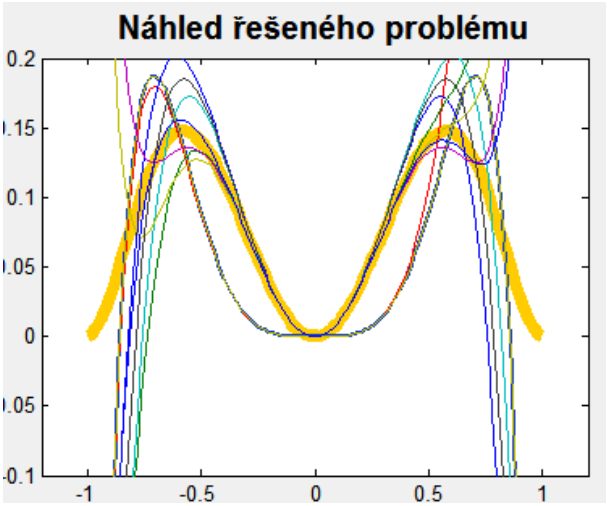
(ZN):+:-

*(OP):+:-:/:**

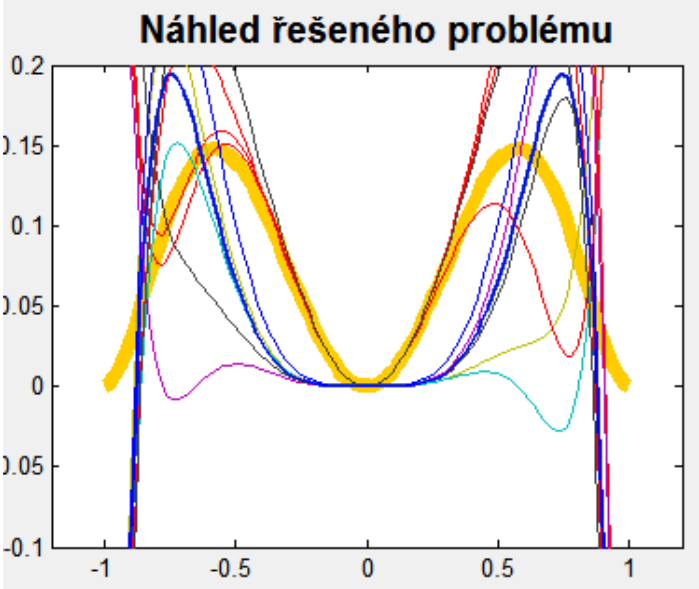
(Num):((Int)+(Dec)):(Int):(Dec)

(Int):(Int)

(Dec):(Dec)



Obr. 29 STE



Obr. 30 SSE

	SSE	*STE*
A:průměr SLE	*53.76*	*63.65*
Modus SLE	*66.37*	*84.02*
Medián SLE	*57.25*	*66.29*
Rozptyl SLE	*376.53*	*282.51*
Odchylka SLE	*19.40*	*16.80*
Průměrné zůžení	-	*1.88*
Počet zůžení	-	*14.9*
Nejlepší hodnota SLE	*42.15*	*53.14*

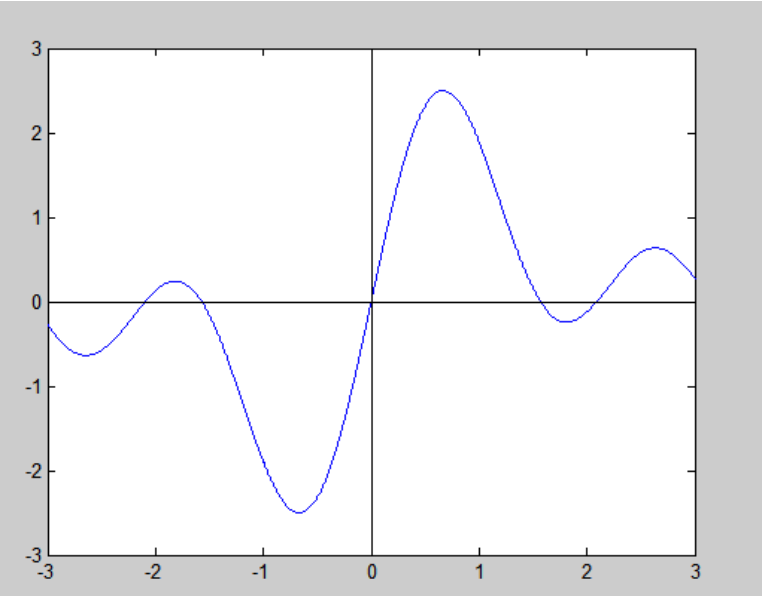
Nejlepší jedinec pro SSE

$$-3x\text{^}8 + x\text{^}6 + x\text{^}4$$

Nejlepší jedinec pro STE

$$x\text{^}9 - x\text{^}7 - x\text{^}6 - x\text{^}4 + x\text{^}2$$

8.3 SinusThree



Obr. 31 Graf funkce sin(x)+sin(2x)+sin(3x)

Použitá gramatika

(First):(gon)

(gon):(sin(fcn))(ZN)(gon):(cos(fcn))(ZN)(gon):(sin(fcn)):(cos(fcn))

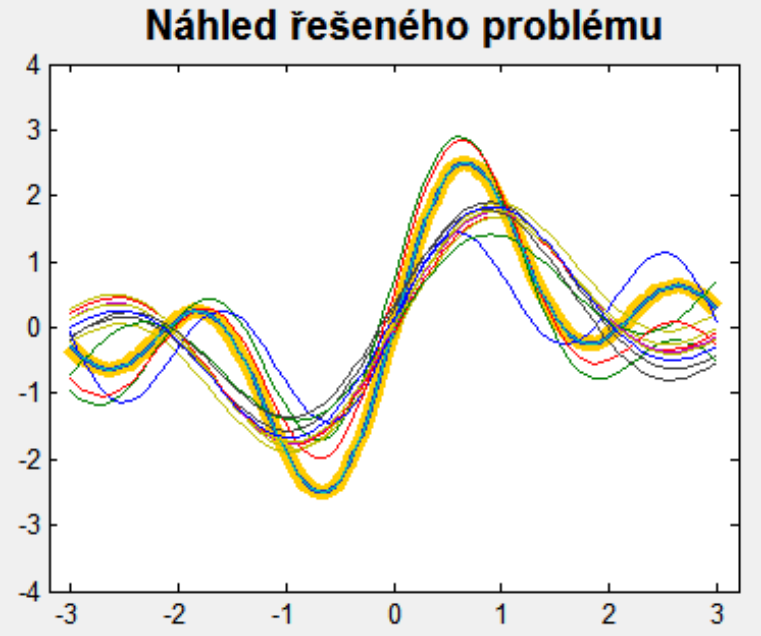
(fcn):(Var)(ZN)(Num):(Var)(OP)(Num)

(Var):(X)

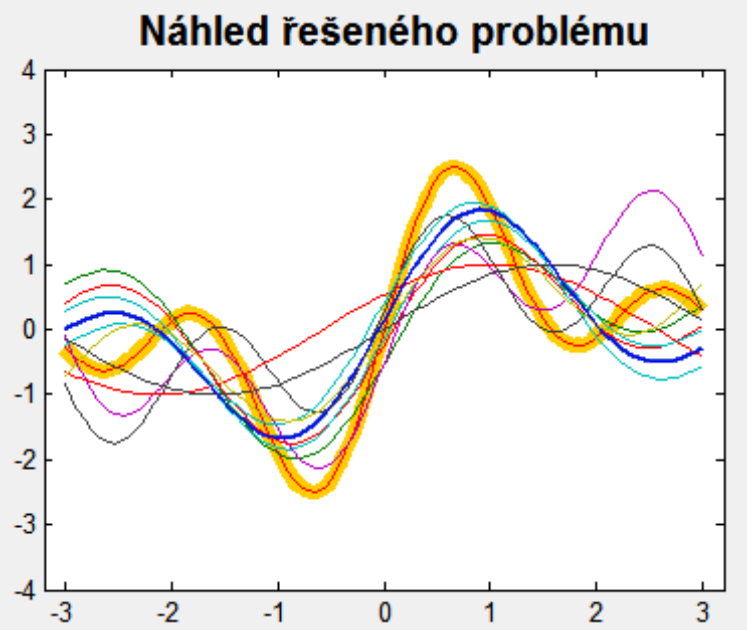
(ZN):+:-

(OP):^::/*

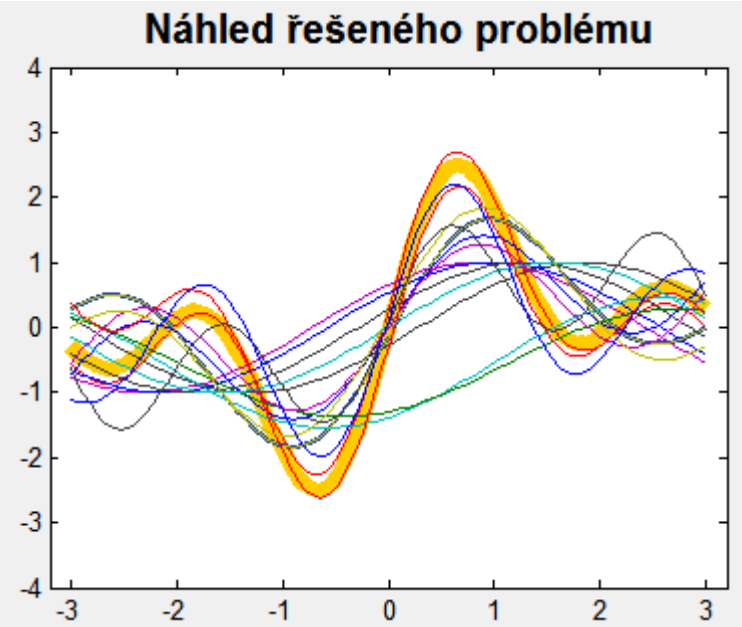
(Num):0:1:2:3:4:5:6:7:8:9



Obr. 32 SSE



Obr. 33 STE



Obr. 34 STE - F

	SSE	*STE*	*STE-F*
A:průměr SLE	*53.76*	*63.65*	*69.07*
Modus SLE	*66.37*	*84.02*	*65.12*
Medián SLE	*57.25*	*66.29*	*66.38*
Rozptyl SLE	*376.53*	*282.51*	*502.12*
Odchylka SLE	*19.40*	*16.80*	*22.4*
Průměrné zůžení	-	*1.88*	*2.06*
Počet zůžení	-	*14.9*	*13.55*
Nejlepší hodnota SLE	*0*	*0*	*17.79*

Nejlepší jedinec pro SSE

$$sin(2x) + sin(3x) + sin(x)$$

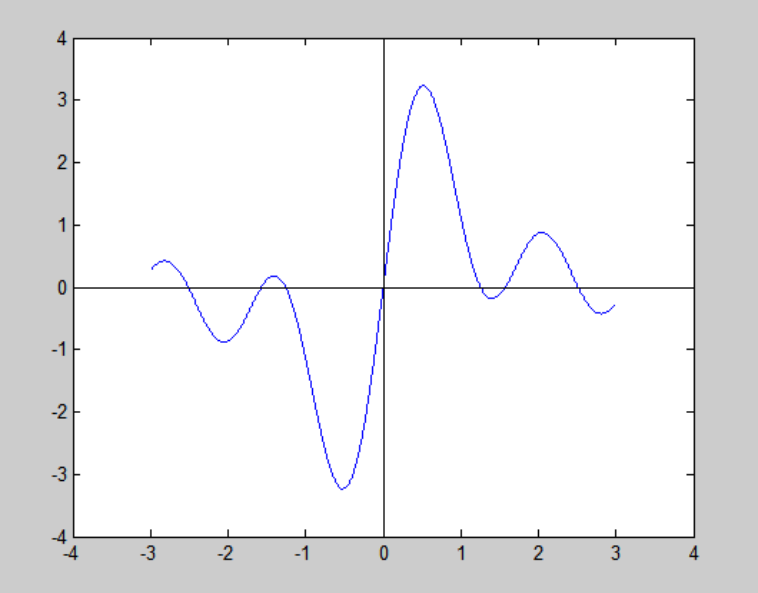
Nejlepší jedinec pro STE

$$sin(2x) + sin(3x) + sin(x)$$

Nejlepší jedinec pro STE-F

$$sin(2x) + sin(3x) + sin(-6x)$$

8.4 SinusFour



Obr. 35 Graf funkce sin(x)+sin(2x)+sin(3x)+sin(4x)

Použitá gramatika

(First):(gon)

(gon):(sin(fcn))(ZN)(gon):(cos(fcn))(ZN)(gon):(sin(fcn)):(cos(fcn))

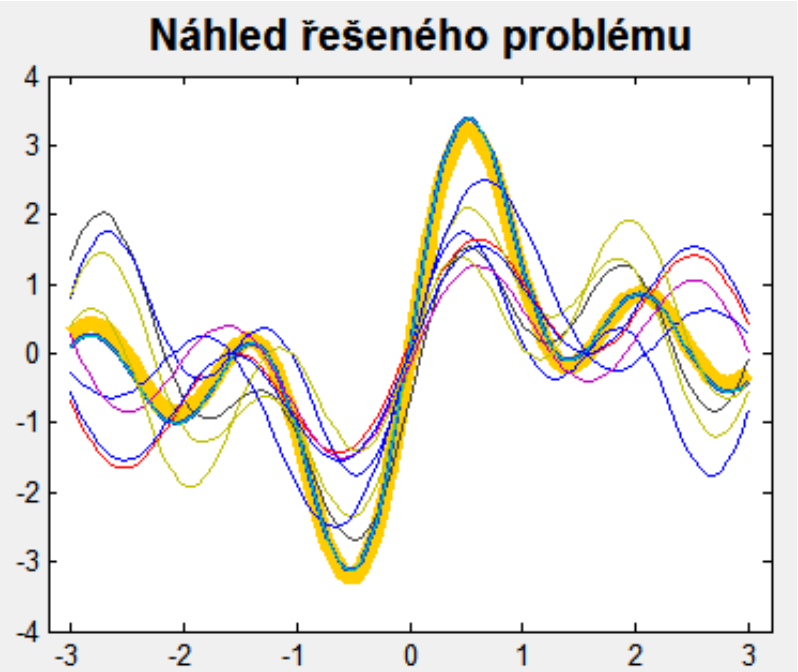
(fcn):(Var)(ZN)(Num):(Var)(OP)(Num)

(Var):(X)

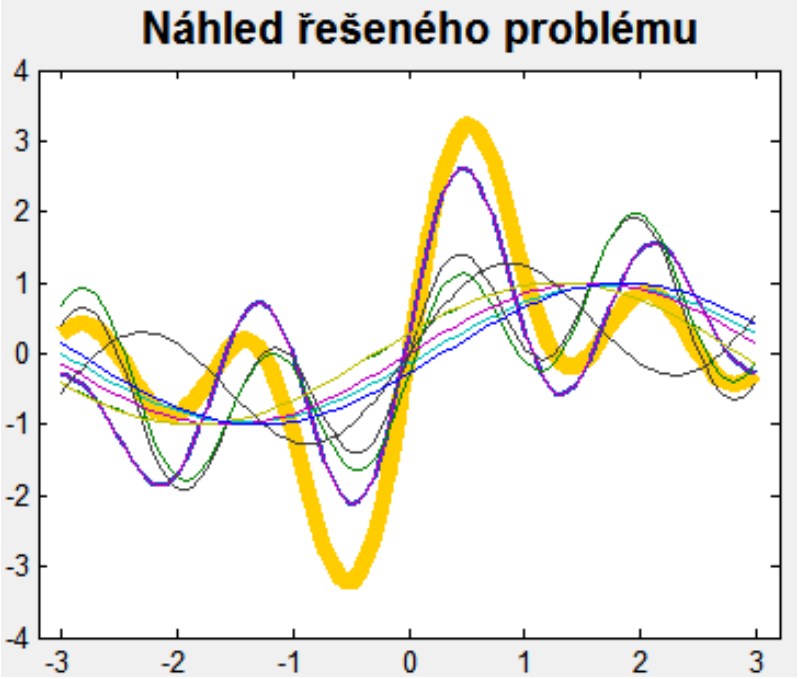
(ZN):+:-

(OP):^::/*

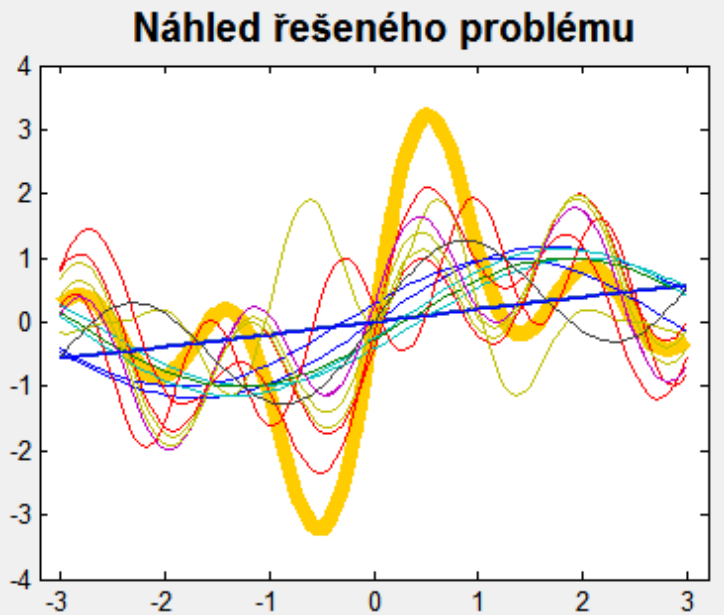
(Num):0:1:2:3:4:5:6:7:8:9



Obr. 36 SSE



Obr. 37 STE



Obr. 38 STE-F

	SSE	*STE*	*STE-F*
A:průměr SLE	*75.99*	*91.92*	*92.51*
Modus SLE	*82.07*	*66.96*	*84.02*
Medián SLE	*82.07*	*96.59*	*89.72*
Rozptyl SLE	*267.91*	*55.99*	*114.56*
Odchylka SLE	*16.36*	*7.48*	*10.70*
Průměrné zůžení	-	*2.54*	*2.51*
Počet zůžení	-	*8.9*	*9.4*
Nejlepší hodnota SLE	*10.61*	*66.96*	*66.28*

Nejlepší jedinec pro SSE

$$sin(4x) - (cos(x + 9) + (sin(x - 4) - (sin(2x) + (sin(3x)))))$$

Nejlepší jedinec pro STE

$$sin(3x) - (cos(x - 7) - (sin(x - 6) + (sin(4x) + (cos(x - 7)))))$$

Nejlepší jedinec pro STE-F

$$sin(4x) + (cos(X - 8) + (sin(2x)\ ))$$

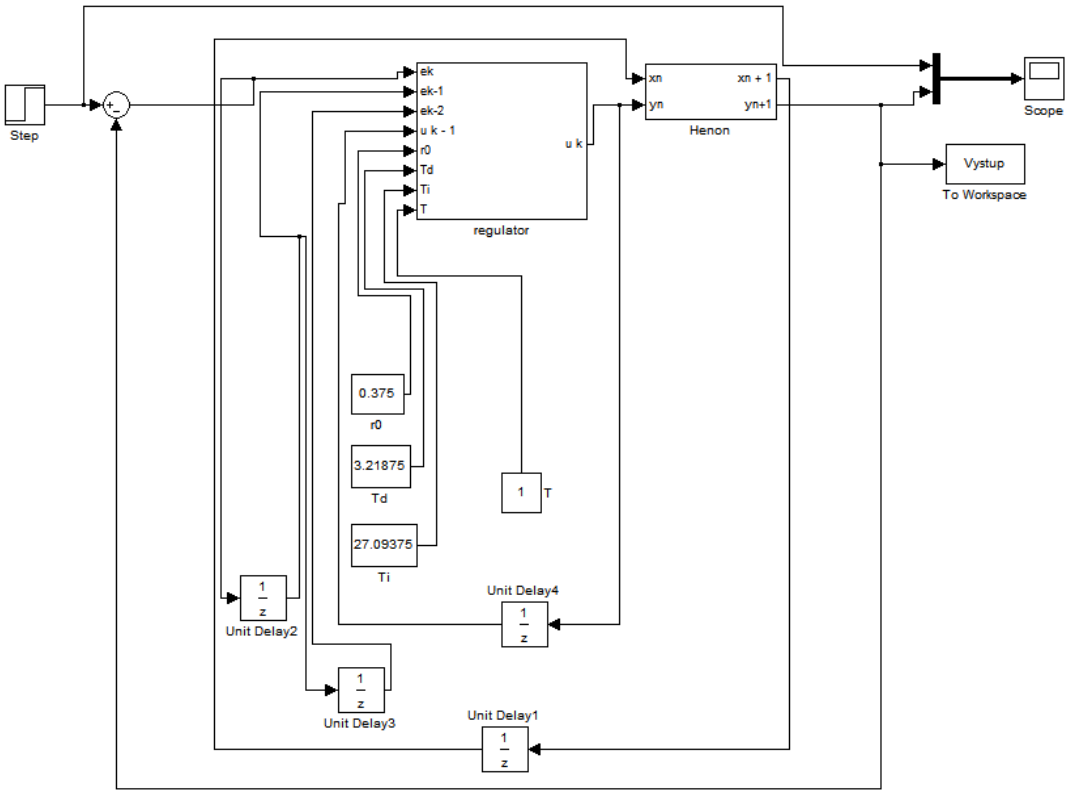
9. DISKRÉTNÍ REGULÁTOR HENON MAP

Henónova mapa je diskrétní dynamický systém pro matematickou simulaci chaotického systému. Jde o dvourozměrné vyjádření jednorozměrné kvadratické mapy[4].

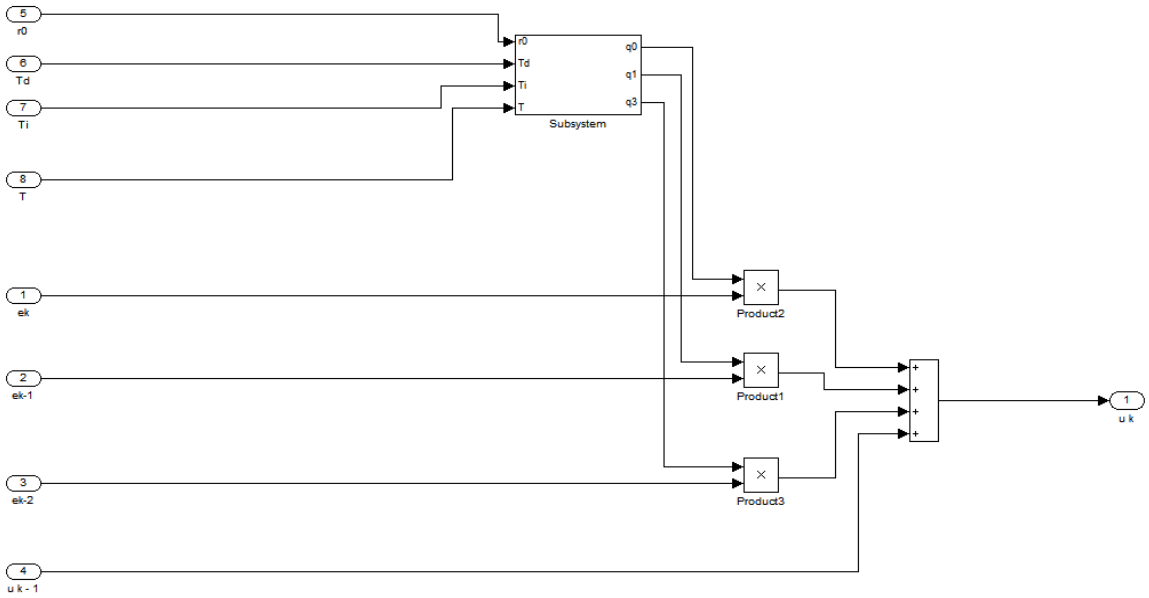
Henónova mapa je vyjádřena rovnicemi:

$$x_{n+1} = 1 + y(n) - ax_n^2$$
$$y_{n+1} = bx_n$$

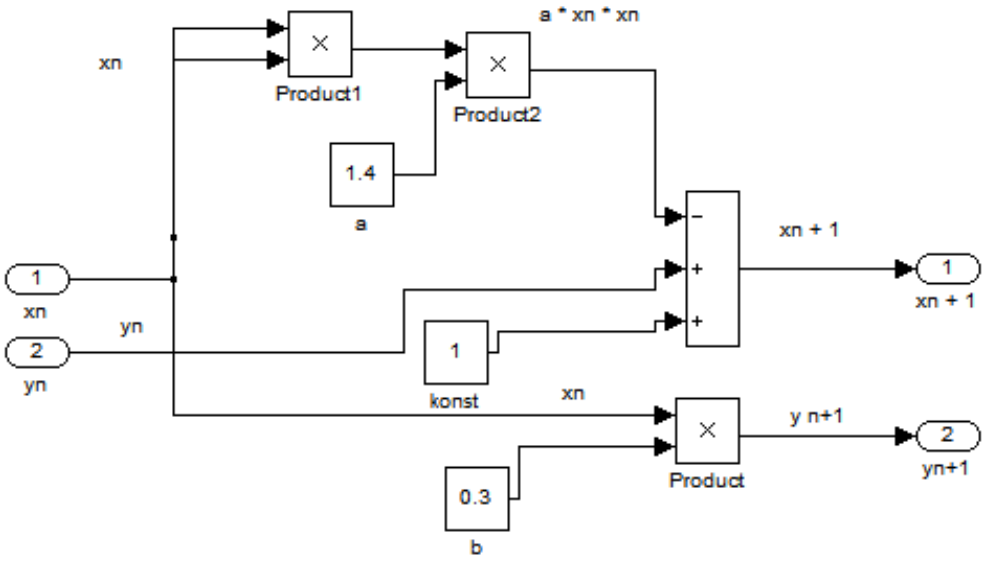
Z rovnic je patrné, že systém závisí na dvou proměnných *a* a *b*, které pro kánonickou Henonovu mapu mají hodnoty parametrů a = 1,4 a b = 0,3. Pro tyto hodnoty se systém určený Henonovou mapou chová chaoticky. Této vlastnosti využijeme a budeme se snažit nastavit regulátor, který po jednotkovém skoku udrží systém poblíž požadované hodnoty.



Obr. 39 Model systému



Obr. 40 Schéma regulátoru



Obr. 41 Schéma Henonovy mapy

9.1 Použitá gramatika

(First):((Ti)/(Td)/(R))

(Ti):((Int)_(Dec))

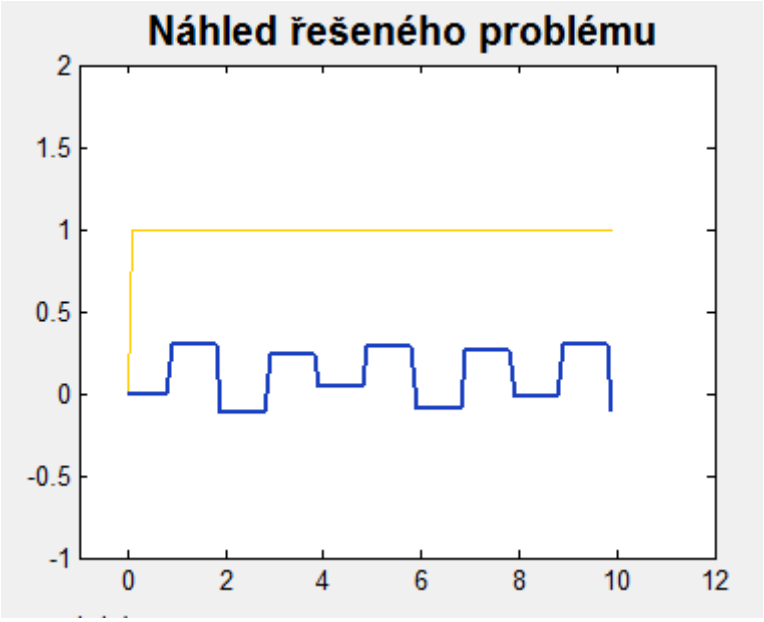
(Td):((Int)_(Dec))

(R):((Int)_(Dec))

(Int):(Int)

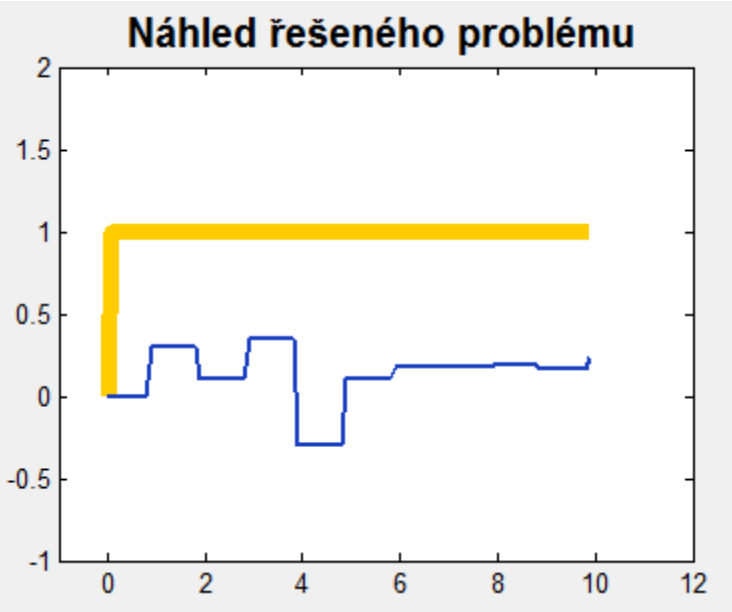
(Dec):(Dec)

9.2 Výsledky



Obr. 42 Nalezená regulace pro 1. měření

měření	R0	Ti	Td
1	0	27,03125000	9,93750000
2	0,125	24.09375	7,25



Obr. 43 2. měření

10. ZÁVĚR

Zadáním této diplomové práce bylo implementovat v jazyku Java gramatickou evoluci. Výslednou Java třídu otestovat na schopnost řešit standardní matematických funkcích Quintic, Sixtic, SinusThree a SinusFour. Dalším testovacím příkladem bylo nastavení regulátoru pro chaotický model Henónovy mapy. Testování třídy dokázalo schopnost algoritmu najít stejnou případně velmi blízkou funkci, která řeší hledaný problém jak pro regresy funkcí, tak pro nastavení regulátoru. Samostatná Java třída je použitelná pro řešení jakéhokoliv problému při zachování pravidel tvorby gramatiky bez ohledu na použitou hodnotící funkci nebo software. V případě této práce byl hodnotícím softwarem Matlab a jeho nádstavba Simulink. Pro použití v jiném projektu řešícím odlišný úkol, stačí vytvořit gramatiku, obslužné GUI, hodnotící funkci a smyčku průběžně volající metody instance této třídy. Z toho vychází i filozofie migračních modelů, kdy lze vytvořit pro jednotlivé populace samostatné vlákno a mít několik paralelně generovaných populací.

V práci též testuji funkci přídavné mutace, kdy již v průběhu tournamentu testuji jestli se jedinec v nově tvořené populaci již vyskytuje. Pokud ano dojde k přídavné mutaci jednoho genu. Tímto se zvýší hodnota mutace ale pouze u jedinců kde je potřeba, díky tomuto uspořádání lze použít vyšší hodnotu tournamentu a tím urychlit nalezení řešení. Další výhodou je snížení šance na uváznutí v lokálních extrémech při vyšším tournamentu, bez ohrožení toho, že by celková vysoká mutace rozbíjela již nalezená řešení.

Cíl této práce byl splněn, při testování algoritmu se nevyskytly funkční problémy či omezení vyplývající ze zvolené struktury řešení. Program dokázal svou funkčnost a schopnost řešit různorodé problémy.

SEZNAM POUŽITÉ LITERATURY

[1] Vitalij Chalupník root.cz [online] 25. 4. 2012
http://www.root.cz/clanky/biologicke-algoritmy-4-neuronove-site/

[2] Martina Husáková, Znalostní technologie IV - Fuzzy Clips [online] 2006
http://lide.uhk.cz/fim/ucitel/fshusam2/lekarnicky/zt_old/fuzzypr.html

[3] Klapka, J.: Metody operačního výzkumu. 2001.

[4] Zbyněk Molnár, Testování výkonnosti evolučních algoritmů při
aplikaci na deterministický chaos [online] 2010
http://dspace.k.utb.cz/bitstream/handle/10563/13778/moln%C3%A1r_2010_dp.pd f?sequence=1

[5] Pavel Bezděk, Gramatická Evoluce – Java, [online], 2009
http://autnt.fme.vutbr.cz/szz/2009/DP_Bezdek.pdf

[6] Cern, [online], 09.09.2004
http://acs.lbl.gov/software/colt/
Copyright (c) 1999 CERN - Evropská organizace pro jaderný výzkum.

[7] FSI VUT Brno, Matematika online, [online], 2012
http://mathonline.fme.vutbr.cz/Aproximace-funkci/sc-88-sr-1-a-88/default.aspx

[8] Doc. Ing. Pavel OŠMERA, CSc., Paralelní gramatická evoluce pro optimalizaci elektronických obvodů, [online], 2009
http://dai.fmph.uniba.sk/events/kuz2009/prispevky-pdf/osmera.pdf

[9] Viral Patel, Java Virtual Machine, An Inside Story!!, [online], 2008
http://viralpatel.net/blogs/java-virtual-machine-an-inside-story/

[10] Doc. Ing. Pavel OŠMERA, CSc., Informační systémy, [online], 2003
http://drogo.fme.vutbr.cz/opory/pdf/InformSys.pdf

[11] Mathworks, [online]
http://www.mathworks.com

Obrázky:

http://commons.wikimedia.org/wiki/File%3ACodon.gif

http://commons.wikimedia.org/wiki/File%3ADimethylglycine_3D.png

http://commons.wikimedia.org/wiki/File%3AProtein_UCN_PDB_2RMF.png

http://commons.wikimedia.org/wiki/File%3AADN_static.png

PŘÍLOHY:

CD s elektronickou verzí práce, Java aplikací a Matlab skriptu.