VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

EVOLUČNÍ VÝPOČETNÍ TECHNIKY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

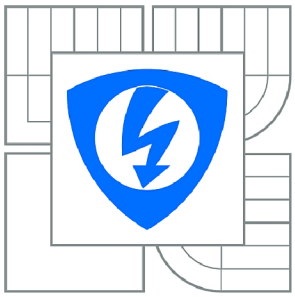
AUTOR PRÁCE
AUTHOR

JAN POPELKA

BRNO 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

EVOLUČNÍ VÝPOČETNÍ TECHNIKY

EVOLUTIONARY COMPUTING

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

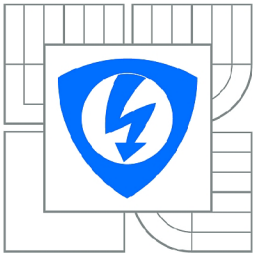
AUTOR PRÁCE — JAN POPELKA

AUTHOR

VEDOUCÍ PRÁCE — Ing. JAN KARÁSEK

SUPERVISOR

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Bakalářská práce

bakalářský studijní obor
Teleinformatika

Student: Jan Popelka ***ID:*** 106726
Ročník: 3 ***Akademický rok:*** 2010/2011

NÁZEV TÉMATU:

Evoluční výpočetní techniky

POKYNY PRO VYPRACOVÁNÍ:

Úkolem studenta bude prostudovat teorii týkající se evolučních optimalizačních technik, zejména genetických algoritmů a genetického programování. Na základě prostudované teorie provede student návrh jednoho ilustrativního příkladu využívajícího optimalizaci pomocí genetického algoritmu a jednoho příkladu využívajícího genetického programování, které budou sloužit ve výuce teoretické informatiky. Součástí bakalářské práce bude implementace navržených příkladů v jazyce JAVA.

DOPORUČENÁ LITERATURA:

[1] MIETTINEN, K., MAKELA, M. M. Evolutionary algorithms in engineering and computer science: recent advances in genetic algorithms, evolution strategies, evolutionary programming, genetic programming, and industrial applications. Miettinen Kaisa. 3rd Ilustrated edition. [s.l.] : Wiley, 1999. 483 s. ISBN 0471999024, 97804.
[2] GEN, M. Genetic algorithms and engineering optimization New York : John Wiley and Sons, 2000. xvi, 495 s. ISBN 0-471-31531-1.

Termín zadání: 7.2.2011 ***Termín odevzdání:*** 2.6.2011

Vedoucí práce: Ing. Jan Karásek

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

Cílem této bakalářské práce bylo seznámit se s evolučními optimalizačními technikami, převážně pak s genetickým algoritmem a genetickým programováním. Následně byla popsána optimalizační úloha obchodního cestujícího řešená pomocí genetického algoritmu, v další kapitole řešení symbolické regrese za pomoci genetického programování. V praktické části byly tyto optimalizační úlohy vytvořeny v programovacím jazyce JAVA.

KLÍČOVÁ SLOVA

Evoluční algoritmus (EA), Genetický algoritmus (GA), Genetické programování (GP), křížení, mutace, selekce, permutace, problém obchodního cestujícího, symbolická regrese, JAVA, Eclipse

ABSTRACT

The aim of this Bachelor's Thesis was to get acquainted with the Evolutionary Optimization Techniques, mainly with the Genetic Algorithm and Genetic Programming. It was subsequently described the role of optimization problem TSP solved using Genetic Algorithms and other Chapter solving Symbolic Regression using Genetic Programming. This optimalization problems were created in the programming JAVA and there are solved practical part of the thesis.

KEYWORDS

Evolutionary Algorithm (EA), Genetic Algorithm (GA), Genetic Programming (GP), Crossover, Mutation, Selection, Permutation, Travelling Salesman Problem (TSP), Symbolic Regression, JAVA, Eclipse

POPELKA, Jan *Evoluční výpočetní techniky*: bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2011. 54 s. Vedoucí práce byl Ing. Jan Karásek

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Evoluční výpočetní techniky“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

Brno ....

(podpis autora)

Poděkování

Rád bych poděkoval panu Ing. Janu Karáskovi za vedení při této bakalářské práci, ochotu, konzultace a praktické rady. Dále bych chtěl poděkovat své rodině za podporu při studiu.

V Brně dne ....

podpis autora

OBSAH

Úvod **11**

1 Evoluční algoritmy **12**

2 Genetický algoritmus **13**
2.1 Vznik a podstata Genetického algoritmu 13
2.2 Proces genetického algoritmu 15
2.3 Chromozom a Messy-chromozom 15
2.3.1 Kódování chromozomu 16
2.4 Mutace 16
2.5 Křížení 17
2.6 Selekce 19
2.7 Populace 20
2.8 Ukončující podmínka 20

3 Genetické programování **22**
3.1 Porovnání genetického programování s genetickým algoritmem 23
3.2 Reprezentace 23
3.3 Prvky v genetickém programování 24
3.3.1 Jedinec, populace jedinců 24
3.3.2 Terminály 24
3.3.3 Funkce 24
3.4 Počáteční populace 24
3.4.1 Úplná metoda 25
3.4.2 Růstová metoda 25
3.4.3 Metoda half and half 26
3.5 Genetické operátory 26
3.5.1 Operátor křížení 27
3.5.2 Operátor mutace 27
3.6 Fitness funkce 29
3.7 Algoritmus genetického programování 29

4 Problém obchodního cestujícího **31**
4.1 Historie TSP 31
4.2 Algoritmy řešící problém TSP 31
4.2.1 Hladový algoritmus 31
4.2.2 Heuristický algoritmus 32
4.3 Genetický algoritmus pro řešení TSP 33

4.3.1 Ordinální reprezentace 33
4.3.2 Sousedská reprezentace 34
4.3.3 Reprezentace cesty 34

5 Symbolická regrese **37**
5.1 Názorný příklad symbolické regrese 37
5.2 Řešení pomocí genetického programování 38

6 Praktická část **40**
6.1 Řešení problému TSP 40
6.1.1 Popis programu obchodního cestujícího 40
6.1.2 Měření v TSP 43
6.2 Řešení symbolické regrese 44
6.2.1 Popis programu pro výpočet symbolické regrese 45
6.2.2 Měření symbolické regrese 47

7 Závěr **49**

Literatura **50**

Seznam symbolů, veličin a zkratek **52**

Seznam příloh **53**

A Přílohy **54**
A.1 Přiložené DVD 54

SEZNAM OBRÁZKŮ

1.1 Struktura Evolučních algoritmů [9] 12
2.1 Schéma procesu Genetického algoritmu [8] 14
2.2 Dekódování messy-chromozomu pomocí templátu [6] 15
2.3 Ukázka chromozomu s binárním kódováním [4] 16
2.4 Ukázka chromozomu s permutačním kódováním [4] 16
2.5 Mutace binárního kódování [9] 17
2.6 Mutace permutačního kódování [9] 17
2.7 Jednobodové křížení [7] 17
2.8 Vícebodové křížení [7] 18
2.9 Uniformní křížení [7] 18
2.10 Křížení s permutačním kódováním [7] 18
2.11 Ruletová selekce [9] 19
3.1 Schéma genetického programování [14] 22
3.2 Syntaktický strom výrazu $y \cdot \log(x) + x \cdot y$ [7] 23
3.3 Syntaktický strom s hloubkou 3 vytvořený úplnou metodou [15] . . . 25
3.4 Syntaktický strom s hloubkou 3 vytvořený růstovou metodou [15] . . 26
3.5 Operace křížení v GP [7] 27
3.6 Operátor mutace v GP [7] 28
3.7 Algoritmus GP [15] 29
4.1 Ukázka náhodného cestujícího [7] 32
4.2 Trasa T před a po použití algoritmu 2-výměny [7] 32
4.3 Ordinální reprezentace [7] 33
4.4 Počáteční chromozomy [7] 35
5.1 Schéma elektrického obvodu předřadníku pro LED diodu 37
6.1 Výsledný výpis do konzole programu Eclipse 42
6.2 Graf při různém nastavení pravděpodobnosti mutace a 80 % křížení . 43
6.3 Výsledný syntaktický strom 47

SEZNAM TABULEK

4.1 Hranová tabulka [7] 35
4.2 Hranová tabulka po prvním kroku [7] 35
4.3 Hranová tabulka po druhém kroku [7] 36
5.1 Tabulka naměřených hodnot 38
6.1 Výsledky měření při 0 % mutace a 80 % křížení 44

ÚVOD

V této bakalářské práci jsem se všeobecně seznámil s evolučními optimalizačními technikami. V dnešní době se stále více setkáváme s optimalizačními problémy, které využívají při svém řešení některou z metod evolučních algoritmů. Tyto metody nemusí být použity pouze pro konkrétní druh příkladů, ale lze je využívat i ve větším rozsahu. Evoluční algoritmy vycházejí z darwinovské teorie evoluce, kdy z každé populace přežívají jen silní jedinci, kteří se pak podílejí na vzniku nové populace.

Evoluční algoritmy se dělí na 4 části. V mém případě jsem se zaměřil na genetické algoritmy a genetické programovaní. V obou případech se setkáme s prvky, které jsou známy z biologie. Bude se jednat o operátory mutace, křížení, selekce a dědičnost. Operátor mutace mění náhodně a s malou pravděpodobností hodnoty jednotlivých genů. Při křížení vzniknou ze dvou rodičů jejich potomci nesoucí informace po každém z rodičů. U procesu selekce se vybírají jedinci, kteří budou vstupovat přímo do nové populace bez použití operátoru křížení nebo mutace. Důležitým prvkem v genetických algoritmech je chromozom. Ten představuje řetězec informací o vlastnostech a chování jedinců. Jinak je tomu v genetickém programovaní. V tomto případě již nebudou použity chromozomy pevné délky, ale hierarchicky strukturované programy. Výsledkem budou syntaktické stromy. Ty mohou vznikat růstovou metodou, úplnou metodou, nebo kombinací obou těchto metod. Také budou popsány nové pojmy terminály a neterminály. Terminály budou obsaženy v koncových uzlech stromů a neterminály ve vnitřních uzlech. Společnou funkcí pro obě metody je ohodnocení fitness, které udává kvalitu původní a nově vzniklé populace.

V následujících kapitolách budou popsány dva optimalizační příklady. První optimalizační úlohou je problém obchodního cestujícího řešený pomocí genetického algoritmu. V této úloze bude hledána nejkratší vzdálenost mezi několika zvolenými městy tak, aby žádné město nebylo navštíveno vícekrát. Druhou úlohou je příklad symbolické regrese využívající pro své řešení genetické programování. V této úloze byl vybrán elektrický obvod, u kterého bylo předpokládáno, že vzorec pro výpočet tohoto obvodu není znám. Při hledání výsledného vzorce budou použity vstupní a výstupní naměřené hodnoty. Na základě získaných znalostí z teoretické části budou obě tyto optimalizační úlohy prakticky řešeny programovacím jazykem JAVA v programu Eclipse.

1 EVOLUČNÍ ALGORITMY

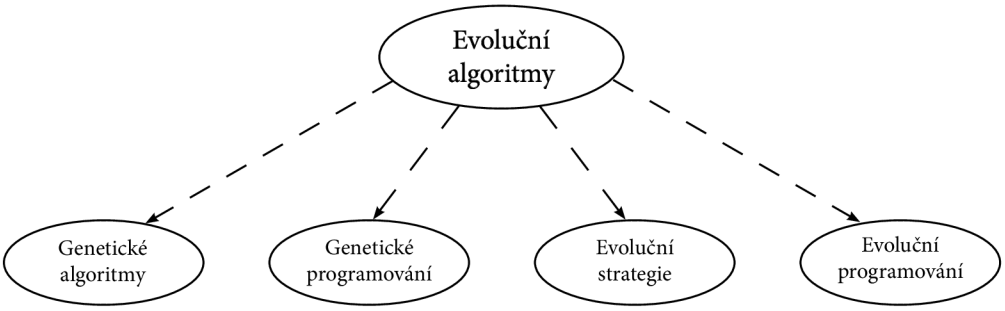
Evoluční algoritmy (EA) patří mezi základní prostředky moderní numerické matematiky při řešení složitých optimalizačních problémů, které nejsou řešitelné běžnými metodami. Jsou používány tehdy, pokud hledáme globální optimum, které je obklopeno lokálními optimy. EA jsou založeny na Darwinově evoluční teorii, která se z biologie přesunula také do informatiky [6]. Podle Darwinovy teorie při vzniku více jedinců, než je možné se v okolním prostředí uživit, vzniká mezi jedinci konkurence a bojují o přežití. V rámci jedné populace mají větší šanci na přežití jedinci s lepšími dědičnými vlastnosti a slabší jedinci zanikají. Tito silní jedinci se budou dále rozmnožovat, přenášet svoji genetickou náklonnost na další generace a přizpůsobovat se na okolí. Během několika generací se změní genetický obsah populace [5].

Darwinova evoluční teorie je založena na třech složkách:

Přirozený výběr je proces ve kterém jsou do reprodukce vybráni jedinci, kteří mají větší pravděpodobnost na přežití.

Náhodný genetický drift můžeme chápat jako náhodné vlivy u jedinců, které ovlivňují populaci. Může to být např. náhodná mutace genetického materiálu, nebo tzv. náhodná smrt jedince, který zanikl dříve, než se zúčasnil genetického procesu.

Reprodukční proces. V tomto procesu se z rodičů vytvářejí potomci. Potomci zdědí genetické informace po svých rodičích. Probíhá to tak, že se z genetické informace obou rodičů náhodně vyberou části chromozomů - informace, ze kterých se náhodně sestaví genetická informace nového jedince. Tomuto procesu je říká křížení a detailněji si ji rozebereme v další kapitole [6].



Obr. 1.1: Struktura Evolučních algoritmů [9]

Na obrázku 1.1 lze vidět, že EA se dále dělí na genetický algoritmus, genetické programování, evoluční strategii a evoluční programování. V dalších kapitolách se podrobně seznámíme jen s genetickými algoritmy a genetickým programováním, které budou pro nás důležité při vytváření programu na výpočet složitých matematických úloh.

2 GENETICKÝ ALGORITMUS

Základní myšlenkou genetických algoritmů je pokusit se napodobit vývoj a učení nějakého živočišného druhu a takto vzniklý algoritmus použít při řešení úloh, které se vyskytují ve složitém, případně i měnícím se prostředí [3]. ***Genetické algoritmy*** (GA) spolu s evolučním programováním, genetickým programováním a evoluční strategií patří do skupiny evolučních algoritmů. Jedná se o vyhledávací algoritmy, které jsou založeny na mechanismu přirozeného výběru a na principech genetiky [1]. GA patří do skupiny evolučních výpočtů, což je rychle se rozvíjející oblast umělé inteligence. GA používají při řešení problémů prvky, které jsou známé z biologie. Jedná se o křížení, mutaci, dědičnost a přirozený výběr.

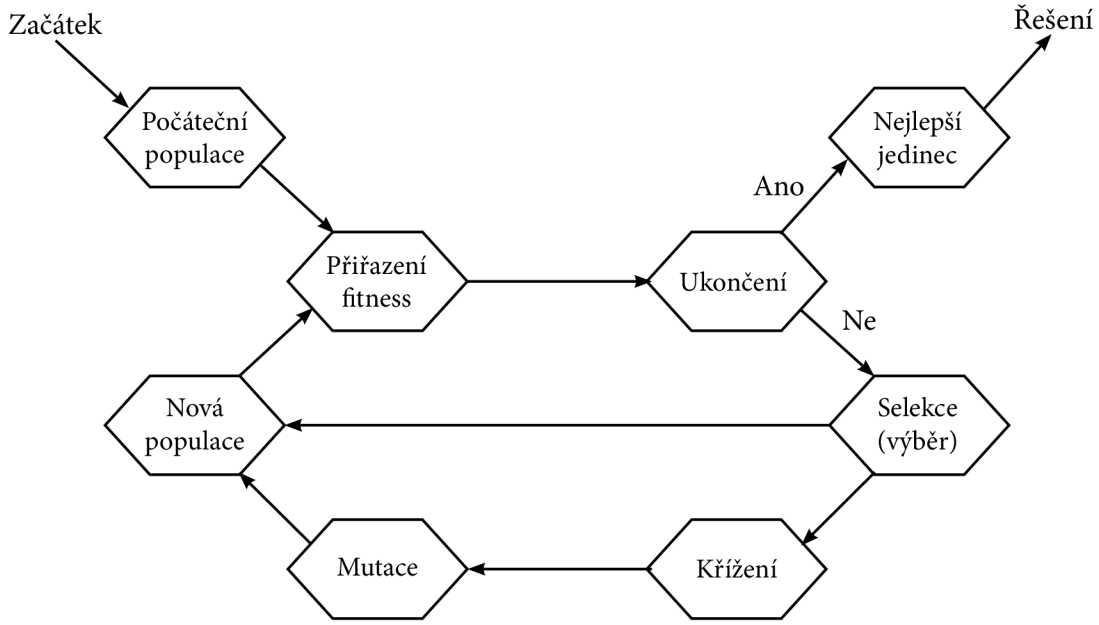
Mezi hlavní výhody GA patří skutečnost, že nevyžadují žádné speciální znalosti o cílové funkci a jsou odolné vůči sklouznutí do lokálního optima. Mají velmi dobrý výsledek u problémů, kde se setkáváme s rozsáhlými množinami přípustných řešení a mohou být využity pro nejrozmanitější optimalizační problémy. S GA se často setkáváme při vytváření rozvrhů práce pro stroje v továrnách, v teorii her, v managementu, pro řešení optimalizačních problémů multimodálních funkcí, při řízení robotů, v rozpoznávacích systémech a v neposlední řadě v úlohách umělého života [3]. U GA se však setkáme i s nevýhodami. Potýkají se s problémem při nalezení optima, dále vyžadují velké množství vyhodnocování cílové funkce, a jejich implementace není vždy přímočará [2].

2.1 Vznik a podstata Genetického algoritmu

S myšlenkou o evolučních výpočtech přišel již v roce 1960 I. Rechenberg ve své práci *Evolutionsstrategie*. Pojem GA pak jako první formuloval John Holland v roce 1975 spolu se svými studenty a kolegy. Tuto teorii popisuje ve své knize A*daptation in Natural and Artificial Systems*, vydanou v roce 1975 na Univerzitě v Michiganu [4]. Prezentoval GA jako účinný prohledávací mechanismus pro adaptivní systémy umělé inteligence. Definoval operátor *křížení* (crossover operátor) a operátor inverze. Operátor křížení je považován za hlavní rozlišovací znak GA, které tento rekombinační operátor považují za primární [5]. GA převážně slouží pro řešení optimalizačních problémů. Jejich činnost je založena na principech genetiky a mechanismu přirozeného výběru.

GA jsou inspirovány z Darwinovy teorie o vývoji druhů a simulují boj jednotlivých organizmů (jedinců) o přežití a schopností množit se. Tak jako v biologii i v genetických algoritmech se setkáváme s pojmy jako je *genotyp* a *fenotyp*. Genotyp představuje dědičný genetický kód, který se dědí po rodičích. Pojem fenotyp

si můžeme vysvětlit jako jedince, který vznikl podle instrukcí obsažených v jeho genotypu. K tomu, aby bylo možné posoudit, kteří členové populace mají větší šanci se podílet na dalším vývoji hledaného řešení, musí být tato schopnost jedinců (fenotypů) kvantifikovatelná. V této souvislosti se mluví o ohodnocení, míře kvality, vhodnosti, síle či reprodukční schopnosti jedince nazývanou převážně pojmem *fitness*. Jedinci, kteří mají lepší ohodnocení fitness mají přirozeně větší šanci přežít déle a podílet se na vytváření následné generace. Použitím metod jako je křížení a reprodukce potom vznikne nová generace jedinců, kteří částečně zdědily vlastnosti po rodičích a částečně jsou ovlivněny náhodnými mutacemi v procesu reprodukce. Pokud se evoluční cyklus opakuje vícekrát, z velké části po desítkách až stovkách opakování dojde ke vzniku jedince, který má vysoké ohodnocení a může tedy představovat optimální řešení daného problému. Avšak, protože evoluční proces zahrnuje v sobě i značný díl náhodnosti, je zřejmé, že každý běh příslušného algoritmu se bude odvíjet jiným způsobem. Z tohoto důvodu se však může stát, že se celá populace v procesu vývoje zdegeneruje a nejlepší jedinec bude představovat pouze lokální optimum, které se může velmi lišit od globálního optima. Chování genetických algoritmů bývá popisováno pomocí různých statistik, které shrnují hodnoty nejlepších, průměrných a nejhorších hodnot sledovaných ukazatelů [7].



Obr. 2.1: Schéma procesu Genetického algoritmu [8]

2.2 Proces genetického algoritmu

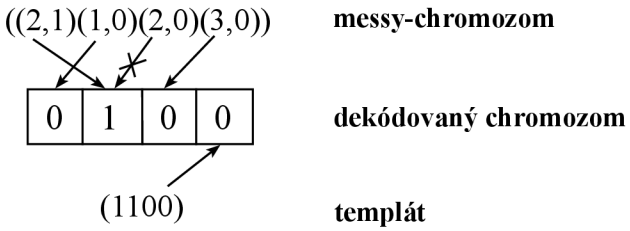
Na následujícím příkladu je názorně vysvětleno, jak probíhá celý proces genetického algoritmu.

Z obrázku 2.1 je patrné, že na začátku celého procesu je počáteční populace, která se většinou získá náhodným vygenerováním jejich genů . V dalším kroku je každému jedinci v populaci přiřazeno jeho ohodnocení fitness . Všichni jedinci populace se navzájem porovnají a potom se vybere skupina jedinců, kteří se beze změny dostanou přímo do nové populace, a zbývající jedinci se náhodně spárují, a v dalším kroku dochází mezi těmito páry ke „křížení“. Následně po křížení dochází k „mutaci“ jedinců. Po takovém genetickém procesu se z jedinců vytvořila nová populace. Proces selekce (výběr), křížení a mutace se opakuje do té doby, dokud se nevytvoří silní jedinci, kteří mají největší pravděpodobnost na přežití [8].

2.3 Chromozom a Messy-chromozom

Chromozom je řetězec informací, která v sobě nese vlastnosti a chování každého jedince, neboli jeho genotyp. Nejčastěji jde o řetězec nul a jedniček, kterým je zakódována pozice jedince v prostoru možných řešení. Chromozom se dále dělí na jednotlivé geny, které jsou uspořádány lineárně. Znamená to, že pokud jsou dva chromozomy stejného typu, pak n-tý gen bude představovat stejné vlastnosti v obou chromozomech. Aby byla určena vlastnost chromozomu musí každý gen nabývat různých hodnot [7].

Dle [6] je M-chromozom definován takto: Messy-chromozom má na rozdíl od běžných chromozomů proměnlivou délku a jejich složky jsou specifikovány jak indexem, tak i jejich hodnotou. Nechť $Q = \{1, 2, \ldots, k\} \times \{0, 1\}$ je množina obsahující uspořádané dvojce (μ, ν), kde $\mu \in \{1, 2, \ldots, k\}$ je index a $\nu \in \{0, 1\}$ je binární hodnota. M-chromozom délky l je definován jako $\chi = ((\mu_1, \nu_1), (\mu_2, \nu_2), \ldots, (\mu_l, \nu_l)) \in \boldsymbol{Q}^l$. Obrázek 2.2 představuje dekódování M-chromozomu pomocí templátu.



Obr. 2.2: Dekódování messy-chromozomu pomocí templátu [6]

2.3.1 Kódování chromozomu

Binární kódování je nejpoužívanějším druhem kódování v GA. Při binárním kódování nabývá každý gen pouze dvou stavů, a to buď 0 nebo 1. Chromozomy jsou pak tedy tvořeny řetězcem těchto stavů v dané délce. Avšak binární kódování má nevýhodu, která může ovlivnit celý genetický algoritmus. Pokud totiž nastane nepatrná změna vlastností jedince, pak se tato změna projeví jen nepatrně v jeho chromozomu. Pro tuto skutečnost se však dá použít *Grayův kód*. Jeho hlavní vlastnost spočívá v tom, že sousední hodnoty jsou zakódovány binárními řetězci o dané délce tak, že se tyto řetězce liší v jednom bitu. Malé změny v chromozomu projeví díky Grayovu kódu lépe, než u běžného binárního kódování [7]. Ukázka chromozomu s binárním kódováním je zobrazena na obrázku 2.3.

Chromozom A	**0**	**1**	**0**	**1**	**0**	**1**	**1**	**0**
Chromozom B	**1**	**1**	**0**	**1**	**1**	**0**	**0**	**1**

Obr. 2.3: Ukázka chromozomu s binárním kódováním [4]

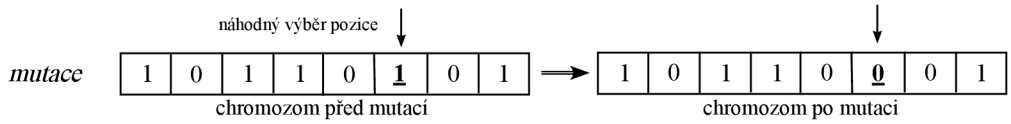
Permutační kódování je dalším způsobem kódování se kterým se setkáváme v GA. Rozdíl permutačního kódování od binárního spočívá v tom, že při permutačním kódování je chromozom tvořen řetězcem čísel, kde každé číslo představuje pozici v pořadí [11]. Permutační kódování se používá při řešení problému obchodního cestujícího, který bude popsán později. Chromozom s permutačním kódováním ukazuje obrázek 2.4.

Chromozom A	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Chromozom B	**6**	**2**	**5**	**1**	**8**	**3**	**7**	**4**

Obr. 2.4: Ukázka chromozomu s permutačním kódováním [4]

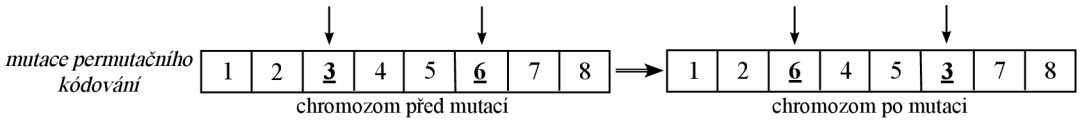
2.4 Mutace

Mutace je operátor, který velmi jednoduchým způsobem a s malou pravděpodobní mění hodnoty jednotlivých genů. Při mutaci binárního kódování se změní hodnota genu v chromozomu z nuly na jedničku a obráceně [7]. Na obrázku 2.5 je znázorněn operátor mutace při binárním kódování.



Obr. 2.5: Mutace binárního kódování [9]

Mutace permutačního kódování (viz. obrázek 2.6), probíhá jiným způsobem. Nelze totiž náhodně vybrat a změnit pouze jedno číslo, protože by došlo k tomu, že původní hodnota z permutace zmizí a nová se objeví dvakrát. Proto je potřeba, aby byli v chromozomu vybrány dvě pozice, které se mezi sebou vzájemně vymění [7].

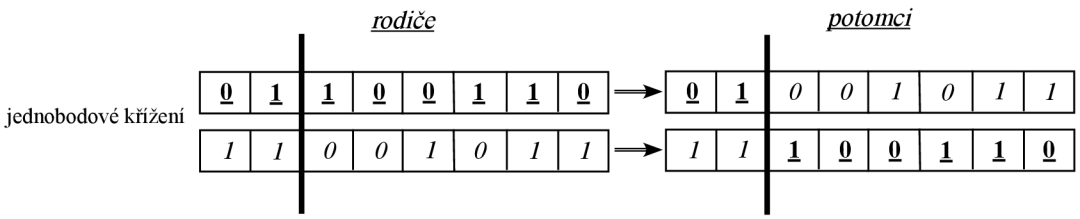


Obr. 2.6: Mutace permutačního kódování [9]

2.5 Křížení

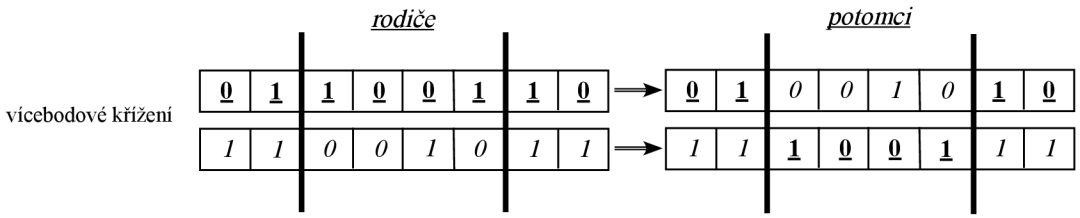
Při procesu křížení se z genomu dvou rodičů vytvoří dva potomci [9]. Každý z těchto potomků nese geny po každém z rodičů.

Jednobodové křížení zobrazeno na obrázku 2.7 je nejjednodušším způsobem křížení. Při tomto procesu se náhodně vybere z chromozomu jeden gen za kterým se zbylé části chromozomu obou rodičů vymění a tím se vytvoří dva noví jedinci. Každý z těchto nově vzniklých jedinců tak nese geny po obou rodičích [7].



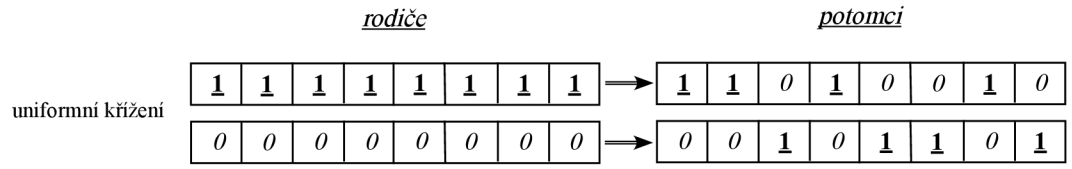
Obr. 2.7: Jednobodové křížení [7]

Vícebodové křížení má stejnou podobnost jako jednobodové jen s tím rozdílem, že se chromozom náhodně rozdělí na více částí řetězce mezi kterými pak dochází ke křížení obou rodičů [5]. Princip vícebodového křížení popisuje obrázek 2.8.



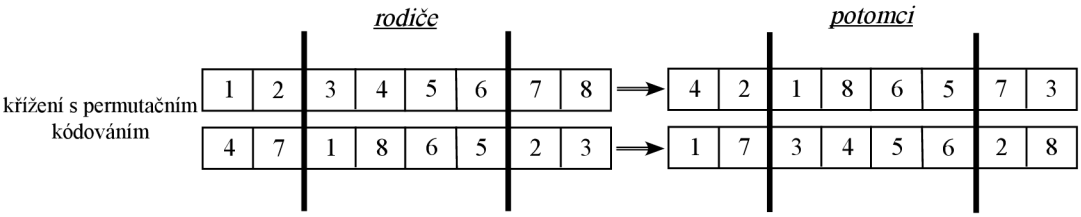
Obr. 2.8: Vícebodové křížení [7]

Při **uniformním křížení** jsou náhodně vybrány jednotlivé geny z chromozomu, které se zkříží [5]. Tuto operaci představuje obrázek 2.9.



Obr. 2.9: Uniformní křížení [7]

Permutační křížení na obrázku 2.10 má podobný postup jako jednobodové nebo vícebodové křížení, kdy je chromozom náhodně rozdělen na více částí. V tomto případě chromozom obsahuje řetězec čísel označující pořadí v chromozomu. Při křížení by však docházelo k tomu, že při výměně genů by některá hodnota byla v chromozomu obsažena dvakrát a jiná by zmizela. Proto se dosazují hodnoty do řetězce postupně, aby každá hodnota v chromozomu byla obsažena jen jednou [7].

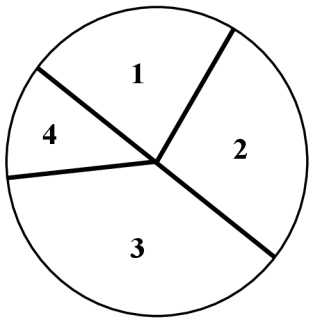


Obr. 2.10: Křížení s permutačním kódováním [7]

2.6 Selekce

Selekce GA je takový proces, při kterém se vybírají jedinci, kteří se následně budou podílet na vytvoření nové populace. V tomto procesu mají větší šanci na vytvoření nové generace jedinci s vyšším ohodnocením fitness. Avšak, musí být vybrání i jedinci s menším fitness, protože v případě že jsou vybrání jen silní jedinci, může dojít k tzv. předčasné konvergenci (lokální optimum funkce). V druhém případě, pokud by byli vybráni jen slabší jedinci, řešení by trvalo velmi dlouho (pomalá konvergence algoritmu) [5].

Ruletová selekce je nejčastěji používaný proces selekce. V tomto případě si můžeme představit ruletové kolo, které je rozděleno na několik částí s různými velikostmi výseče, kde každá výseč představuje velikost ohodnocení fitness každého jedince. Jedinci s větším fitness bude přiřazena větší část výseče. Má tedy větší šanci, že bude vybrán. Jedinci s menším fitness mají menší část výseče a tedy menší šanci, že budou vybráni. Při sestavování ruletového kola se nejdříve sečtou ohodnocení fitness všech jedinců a následně je každému jedinci proporcionálně přiřazena odpovídající velikost výseče. Tato metoda má však nevýhodu. Pokud existuje jedinec, který převyšuje hodnotou fitness výrazně před ostatními, tak může dojít k tomu, že ostatním jedincům bude téměř znemožněno podílet se na vytvoření populace [7]. Ukázku ruletové selekce popisuje obrázek 2.11



Obr. 2.11: Ruletová selekce [9]

Pořadová selekce. Tato metoda zamezuje tomu aby výrazně silný jedinec znemožnil podílet se na vytvoření nové populace slabším jedincům. V této metodě vzniká pořadí, ve kterém jsou všichni jedinci seřazeni vzestupně podle jejich ohodnocení fitness. Takto vzniklé pořadí se pak využívá při výběru jedinců. Při použití

pořadové selekce jsou tak vybráni i slabší jedinci a dochází tak ke zrovnoměrnění šance jedinců [7].

Turnajová selekce V této metodě mají oproti ruletové selekci větší šanci i jednici, kteří mají menší ohodnocení. Celý princip funguje tak, že se nejdříve náhodně vyberou jedinci, nejčastěji dva, kteří se spolu setkají v simulované souboji. Zvítězí ten jedinec, který má větší ohodnocení fitness a tak postupuje dál výběrem. Pro výběr jedinců v souboji se používá náhodný index nebo ruletová selekce [9].

2.7 Populace

V předchozích textech jsme si podrobně vysvětlili postup při vytvoření nové populace. Z původní generace, která prošla selekcí, křížením a mutací vnikla nová populace, která následně představuje vhodné řešení optimalizovaného problému. Při celém tomto procesu však dochází k tomu, že původní generace nemůže přežít delší dobu než jeden generační cyklus. Dochází k tomu, že silný jedinec nebo celá skupina můžou být navždy ztraceny pokud se jim nepodaří projít selekčním procesem nebo pokud jejich genetická informace bude procesem křížení a mutace změněna. Proto jsou využívány dva přístupy, které umožňují silným jedincům přežít více než jen jeden generační cyklus. První z přístupů je ***elitizmus***, který umožňuje GA ponechat si několik silných jedinců a beze změny je překopírovat do nové populace. Další přístup je ***setrvalý stav***, která uchovává většinu populace nezměněnou a do celého genetického procesu vytvoření nové populace jde jen pár jedinců, kteří mají nejmenší ohodnocením. Tím jsou nejlepší jedinci uchováváni a dochází velmi pomalu k postupnému vývoji nové populace [7].

2.8 Ukončující podmínka

V genetickém algoritmu lze použít několik ukončujících podmínek, které se dají i vzájemně kombinovat.

Jednou z podmínek je **počet generací**. Tato podmínka udává maximální počet generací za který GA skončí a je vyhlášen nejlepší jedinec jako řešení daného problému.

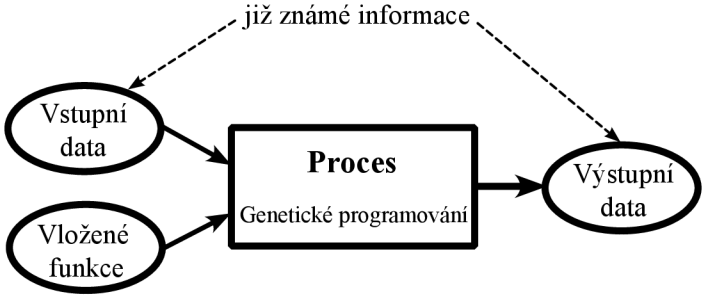
Účelová funkce nejlepšího jedince je další z ukončujících podmínek. Při této ukončující podmínce je zadán požadavek na ohodnocení nejlepšího jedince v celé populaci. Pokud některý jedinec získá lepší ohodnocení než je stanovena tato hranice, algoritmus skončí.

Podmínka **časového omezení** není používána často, avšak lze ji použít tehdy, pokud chceme vyhodnotit poměr rychlosti konvergence k časové náročnosti různých metod [9].

Další ukončující podmínka může být pokud je nalezeno optimální řešení, nebo pokud během několik posledních generací k došlo nepatrné změně dosud nalezeného nejlepšího řešení [7].

3 GENETICKÉ PROGRAMOVÁNÍ

Genetické programování (GP) tvoří jednu ze čtyř částí evolučních algoritmů (EA). Stejně jako genetický algoritmus (GA), tak i GP slouží k nalezení řešení optimalizačních úloh. I v tomto případě se vychází z Darwinovy evoluční teorie o přirozeném výběru. V GP je pro nalezení problému potřeba vložit vstupní data, které odpovídají výstupním hodnotám. Dále se zadají možné funkce, které mohou být použity při hledání řešení. Ostatní už řeší GP sám. Při použití GP není potřeba znát postup řešení daného problému [14]. Jednoduché schéma GP je zobrazeno na obrázku 3.1.



Obr. 3.1: Schéma genetického programování [14]

Historie GP sahá do roku 1957, kdy Richard Friedberg začal vytvářet první stopy v této oblasti pomocí samoučícího algoritmu, který dále zlepšoval. Tento program byl reprezentován jako sled instrukcí pro teoretickou informatiku. Friedberg zde však nepoužil evoluční populaci, protože v té době tato myšlenka nebyla plně vyvinuta a také z důvodu omezené výpočetní kapacity počítačů v té době [14].

O velký rozmach se však postaral na přelomu 80-tých a 90-tých let stanfordský informatik John Koza. Ten navrhl originální modifikaci genetického algoritmu, který nazval *genetické programování*. V tomto případě jsou chromozomy nahrazeny složitějšími funkcemi. Tyto funkce bývají reprezentovány pomocí syntaktických stromů (*parse tree*). Chromozomy zde také nemají pevnou délku, ale jedná se o programy které mají hierarchickou strukturu [6].

Pro úspěšnou a efektivní implementaci GP je důležitá správná metoda kódování stromové struktury. John Koza vytvořil GP zapsané v jazyku LISP. Tento jazyk umí dokonale pracovat se syntaktickými stromy a obsahuje programovací prostředky vhodné pro kódování těchto stromů [6].

3.1 Porovnání genetického programování s genetickým algoritmem

Přestože genetické programování vychází z genetického algoritmu, liší se v následujících směrech:

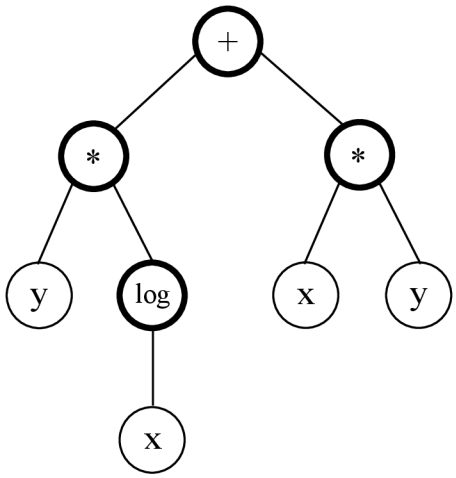
Reprezentace. U GP se setkáváme se spustitelnými strukturami. Převážně to jsou programy, které mají stromovou strukturu proměnné délky. Může se například jednat o elektrické obvody, mechanické nebo optické systémy.

Operátory používané při GP pracují nad spustitelnými strukturami. Z GA jsou již známy operátory křížení a mutace, ale existují i další pokročilé operátory které umožňují vygenerovat program s podprogramem.

Fitness. Pro zjištění ohodnocení *fitness* se nejdříve provede kód pro vstupní data daného programu a následně se dosažené výsledky vyhodnotí [5].

3.2 Reprezentace

V GP se námi hledaná optimální řešení zapisují ve tvaru stromové struktury. Tento výsledný strom se skládá ze dvou množin symbolů. Jedná se o množinu terminálů T a neterminálů F, neboli funkcí. Terminály tvoří koncové listy každého stromu a funkce představují vnitřní (nelistové) uzly [15].



Obr. 3.2: Syntaktický strom výrazu y · log(x) + x · y [7]

Na obrázku 3.2 je znázorněn jednoduchý syntaktický strom. Pro lepší orientaci jsou neterminály (funkce) F znázorněny silněji a terminály T jsou v koncových uzlech. Při vytváření takového stromu se do neterminálů zapisují funkce nebo operace, do terminálů pak vstupní proměnné, konstanty nebo výstupy [15].

3.3 Prvky v genetickém programování

3.3.1 Jedinec, populace jedinců

Stejně jako v GA i tady se setkáváme s pojmy jedinec a populace jedinců. Stejně tak budeme vytvářet nové jedince, používat operace selekce a každý jedinec bude ohodnocen kvalitou *fitness*, která pak představuje potenciální řešení daného problému. Populace je tvořena daným počtem jedinců [15].

3.3.2 Terminály

Jako terminály v GP se používají vstupy programu, neboli proměnné, konstanty a funkce bez argumentů. Terminály tvoří tzv. listy syntaktického stromu. Po zpracování terminálu je programu vrácena určitá hodnota. Proměnná bývá velmi potřebná při procesu učení, kdy se přes tuto proměnnou přenáší data z trénovací množiny do programu. Kromě proměnné lze do programu přivést také konstanty. V základních variantách GP se vybere pro celou jednu populaci množina konstant, která během celého procesu nezmění svou hodnotu. Je-li při následném vytváření stromu přidávána konstanta, jedná se o kombinaci těchto konstant a aritmetických funkcí [5].

3.3.3 Funkce

Jako množinu funkcí představují standardní aritmetické funkce (plus, mínus, krát, děleno), logické funkce (or, and, nor, xor, atd.), standardní matematické funkce (sin, cos, tan, cotg, atd.), konstrukce z programovacích jazyků (skoky, cykly, podprogramy, apod.) a jiné. Velikost množiny není nijak omezena. Je potřeba však použít takové funkce, které by nebyly příliš složité na výpočet a časově náročné [5].

Při vytváření programu je potřeba, aby množiny funkcí a terminálů splňovaly požadavky uzavřenosti (*closure*) a postačitelnosti (*sufficiency*). Požadavek uzavřenosti říká, že jakákoliv funkce v množině F může přijímat různé hodnoty a výsledná data z kterékoliv funkce a také libovolnou hodnotu z terminálu T. Tímto požadavkem se zabraňuje vzniku chybám během doby po kterou programu běží. Požadavek postačitelnost uvádí, že každá množina funkcí a terminálů by měla být zvolena tak, aby obsahovala prvky, které bude možné použít při nalezení řešení problému [7].

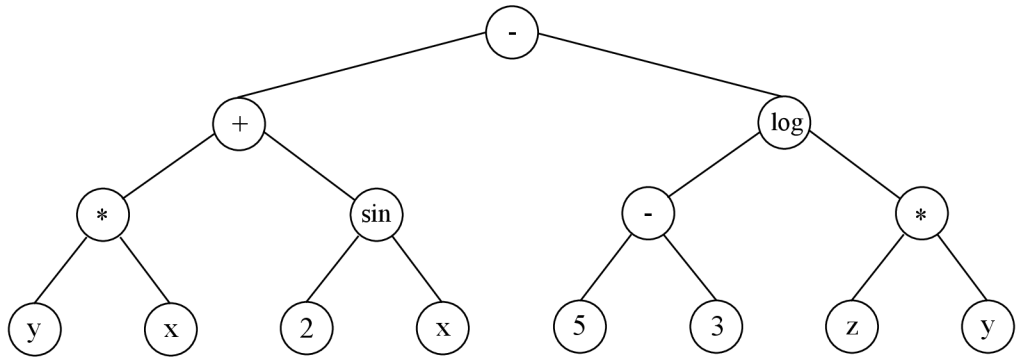
3.4 Počáteční populace

Pokud jsou již definovány množiny terminálů T a funkcí F, lze začít vytvářet počáteční populaci jedinců. Počáteční populace se vytváří pomocí náhodného výběru

terminálů a funkcí z jednotlivých množin F a T. Při vytváření je použit další parametr, kterým se definuje maximální velikost programu. Tato velikost představuje počet uzlů v syntaktickém stromu konkrétního programu, nebo také označována jako hloubka stromu. Při vytvoření první populace je její ohodnocení fitness malé. To je způsobeno tím, že na začátku program zatím vůbec nezná řešení daného problému a celá populace je vytvořena zcela náhodně. Pro vytvoření syntaktického stromu se používá jedna z metod: *úplná* (full method), *růstová* (grow method) a *metoda půl napůl* (ramped half and half) [5].

3.4.1 Úplná metoda

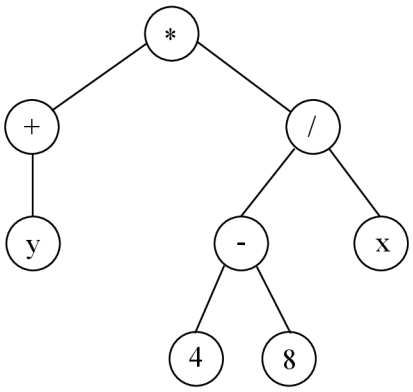
U této metody se vytváří všechny větve syntaktického stromu pouze do maximální nastavené hloubky. Pokud nastane okamžik, kdy bude generován uzel jehož hloubka bude rovna maximální nastavené hloubce, pak bude vybrán terminál a větev se ukončí. V maximální hloubce stromu budou tedy obsaženy pouze terminály [7]. Schéma syntaktického stromu s maximální hloubkou $h_{max} = 3$ vytvořeného úplnou metodou je uveden na obrázku 3.3.



Obr. 3.3: Syntaktický strom s hloubkou 3 vytvořený úplnou metodou [15]

3.4.2 Růstová metoda

V této metodě se pro každý uzel stromu, kromě posledního povoleného, náhodně vybere funkce nebo terminál. Pokud dojde k tomu, že byl vybrán terminál, větev stromu se ukončí a nezáleží na tom, zda byla dosažena maximální definovaná hloubka stromu. Výsledkem této metody jsou syntaktické stromy, které mají nepravidelný tvar [5]. Na obrázku 3.4 je názorná ukázka syntaktického stromu s maximální hloubkou $h_{max} = 3$ za použití růstové metody.



Obr. 3.4: Syntaktický strom s hloubkou 3 vytvořený růstovou metodou [15]

3.4.3 Metoda half and half

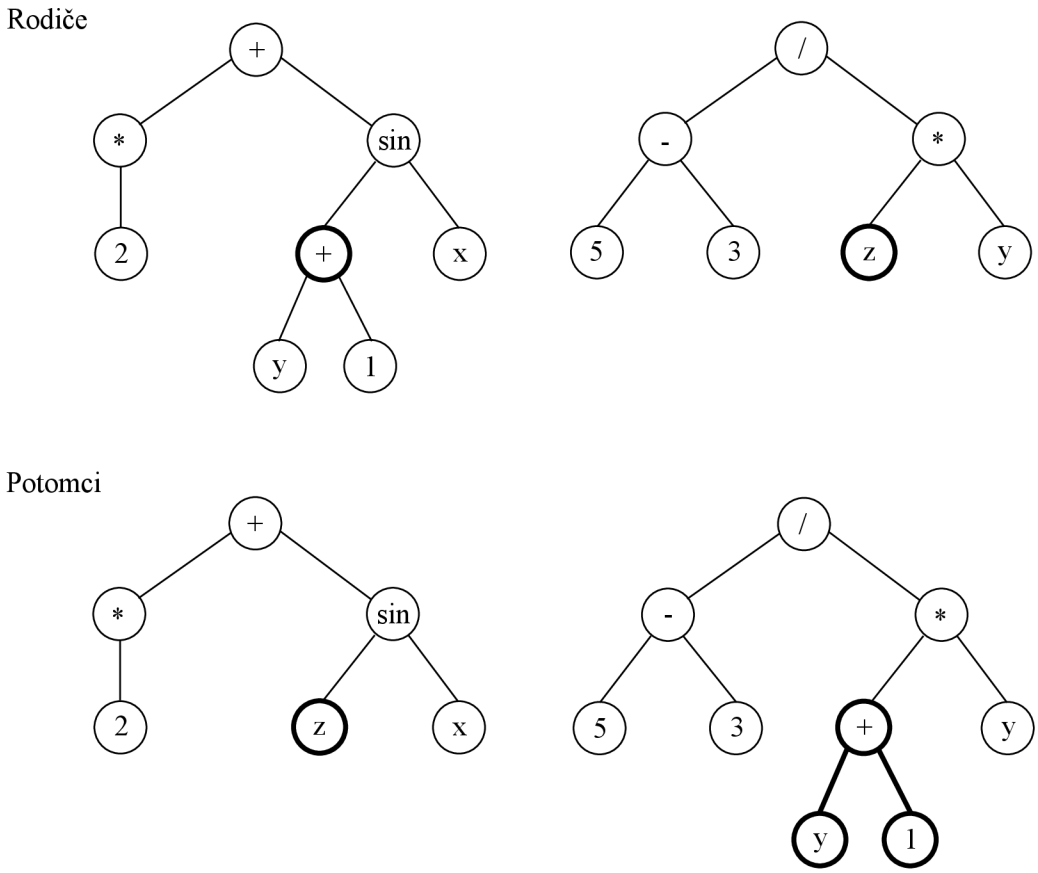
Pro praktické využití doporučuje John Koza používat tuto metodu. Tato metoda spočívá v tom, že při vytváření jedinců počáteční populace se použije jak úplná, tak i růstová metoda. Také zde dochází k rovnoměrnému zastoupení stromů, které mají v počáteční populaci různou hloubku. Pokud bude například velikost počáteční populace $N = 200$ s maximální hloubkou $h_{max} = 6$, tak pro hloubku 2 se vygeneruje 40 stromů, které se rozdělí na dvě poloviny a pro 20 stromů bude použita metoda *úplná* a pro dalších 20 metoda *růstová*. Dalších 40 stromů se vygeneruje pro hloubku 3, které budou opět rozděleny napůl. Takto se pokračuje až do poslední definované hloubky, v tomto případě po hloubku 6. Touto metodou vznikají rozmanité počáteční populace [7].

3.5 Genetické operátory

Stejnou funkci jako v GA má i tady operátor selekce. V GP může být opět použita selekce turnajová, ruletová a pořadová. Proto již tento operátor není potřeba znovu popisovat. Ostatní operátory jako je křížení a mutace budou však popsány z důvodu odlišného způsobu reprezentace. Při vytvoření těchto operátorů je potřeba si uvědomit, že v GP je používána proměnná délka chromozomu a hierarchická struktura syntaktických stromů. Operátory křížení a mutace mohou být použity i samostatně [5], [7].

3.5.1 Operátor křížení

Při provedení operátoru křížení se ze dvou vstupujících jedinců (rodičů) vytvoří jejich noví potomci, kde struktura každého potomka je tvořena po obou rodičích. Tento operátor je uveden na obrázku 3.5. Princip funguje tak, že z chromozomu každého rodiče se vybere náhodně jeden uzel. Podstromy, které z tohoto uzlu vychází se následně mezi rodiči vzájemně vymění. Během křížení však může nastat i situace, kdy se budou prohazovat pouze listové uzly stromů, a proto se používá pravděpodobnostní rozdělení, kdy se provádí křížení s pravděpodobností 90 % na vnitřních uzlech stromů (funkcích) a s pravděpodobností 10 % na listových uzlech (terminálech) [15].

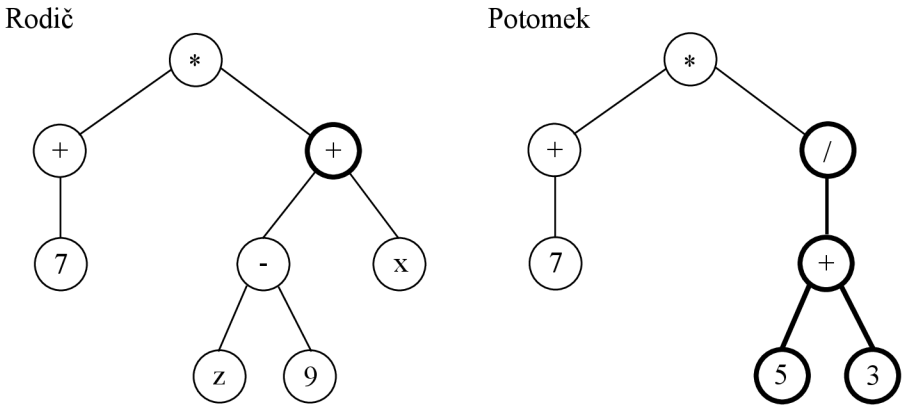


Obr. 3.5: Operace křížení v GP [7]

3.5.2 Operátor mutace

Jedná se o operátora, který náhodně mění strukturu stromu. Tento operátor vybere náhodně v syntaktickém stromu místo mutace (jedná se opět o uzel stromu) a stejně jako v předchozím případě je celý podstrom, který z tohoto uzlu vychází,

nahrazen novým náhodně vytvořeným podstromem. Na obrázku 3.6 je zobrazena ukázka operátoru mutace. Pro vytvoření nového podstromu může být použita metoda úplná nebo růstová, a lze také nastavit maximální možnou hloubku stromu, aby nedošlo k situaci, kdy bude po mutaci vygenerovaný podstrom příliš hluboký. Pravděpodobnost mutace bývá obvykle 5 % [7].



Obr. 3.6: Operátor mutace v GP [7]

Mimo základní operátor mutace existují i další obecně definované operátory mutace. Mezi ně patří:

- ***uzlová mutace*** (*point mutation*), kdy dochází k nahrazení neterminálního uzlu neterminálem, který má stejný počet argumentů, nebo k nahrazení terminálního uzlu jiným terminálem.

- ***vyzvedávající mutace*** (*hoist mutation*) nahrazuje celý syntaktický strom některým z jeho podstromů

- ***smršťující mutace*** (*shrink mutation*) nahradí náhodně vybraný podstrom jediným terminálem.

Každá z těchto variant mutace může být použita v programu při hledání řešení daného problému [7].

Kromě primárních genetických operátorů jako je křížení a mutace se občas v GP používají i operátory sekundární (doplňkové). Tyto operátory však nejsou potřebné pro fungování algoritmu. Patří sem například:

- permutace,
- editace,
- zapouzdření,
- decimování.

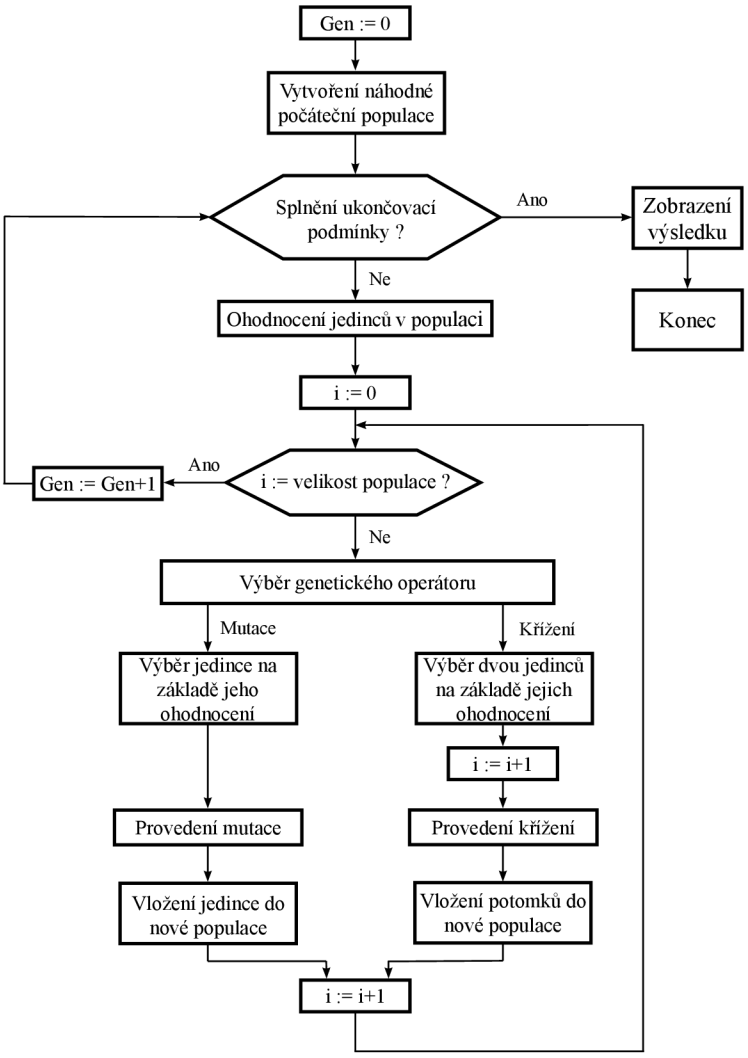
Permutace, editace a zapouzdření pracují pouze s jedním jedincem [15].

3.6 Fitness funkce

Fitness funkce se používá pro ohodnocení každého jedince. Podle ohodnocení fitness lze zjistit o jak kvalitního jedince se jedná. Výsledná hodnota fitness je porovnávána s požadovanou hodnotou. Měření fitness je důležitým prvkem při hledání řešení daného problému [5].

3.7 Algoritmus genetického programování

Algoritmus GP je tvořen z několika částí, které na sebe navazují. Tento podrobnější vývojový algoritmus uvedl John Koza. [15]. Algoritmus je zobrazen na obrázku 3.7.



Obr. 3.7: Algoritmus GP [15]

Na začátku celého algoritmu se nejdříve vygeneruje náhodná počáteční populace. Takto vzniklá populace již má svoji velikost, která se jí vytvoří podle toho, jak náročné je řešení daného problému. Poté se provede kontrola, zda již takto vzniklá populace nesplňuje ukončovací podmínky, které jsou v programu nastaveny uživatelem. Pokud podmínky jsou splněny, zobrazí se výsledek a proces se ukončí. Pokud však splněny nejsou, ohodnotí se všichni jedinci v populaci pomocí funkce *fitness* a následně se vytvoří nová generace jedinců. V dalším kroku je náhodně vybrán genetický operátor mutace nebo křížení. Pro operátor mutace se vybere pouze jeden jedinec, pro operátor křížení je potřeba vybrat jedince dva. Takto nově vzniklí jedinci tvoří nově vzniklou populaci. Proces vytváření nové populace se stále opakuje, dokud nová populace nenabude takové velikosti jaká byla velikost původní populace. Po vytvoření nové populace správné velikosti se prověří, zda splňuje ukončovací podmínky. Celý cyklus se opakuje do doby, kdy jsou podmínky pro ukončení splněny. Jako ukončují podmínky mohou být zvoleny maximální počet provedených generací v procesu nebo pokud je nalezeno optimální řešení daného problému [16].

4 PROBLÉM OBCHODNÍHO CESTUJÍCÍHO

Problém obchodního cestujícího, který je označován zkratkou TSP z anglického překladu Traveling Salesman Problem, je klasická úloha na optimalizaci řešení kombinatorických problémů. V této úloze je třeba nalézt nejkratší trasu mezi n - městy tak, aby začátek a konec trasy byl ve stejném městě, celková vzdálenost byla co nejkratší a každé město bylo navštíveno pouze jednou. I když problém vypadá sám o sobě velmi jednoduše, s přibývajícím počtem navštívených měst se zvyšuje výpočet tohoto problému a stává se časově náročným. Tato úloha je NP-těžká. Je velmi pravděpodobné, že u takovéto úlohy nebude existovat algoritmus, který by umožňoval naleznout v omezeném čase optimální řešení [10], [7].

4.1 Historie TSP

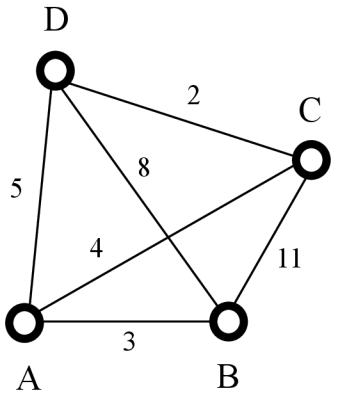
Problém obchodního cestujícího TSP má poměrně bohatou historii. Již v roce 1759 se podobnou úlohou tohoto typu zabýval švýcarský matematik Leonhard Euler [7]. Matematický problém, který následně vedl k současnému TSP se poprvé objevil na začátku 19. století, kdy se o tomto problému zmínil irský matematik William Rowan Hamilton a britský matematik Thomas Penyngton Kirkman. Současný problém TSP představil v roce 1930 matematik Karl Menger. I když se neustále výpočty zlepšují a stále se objevují lepší trasy mezi městy, matematická podstata problému dosud nebyla objasněna [10].

4.2 Algoritmy řešící problém TSP

4.2.1 Hladový algoritmus

Hladový algoritmus je nejjednodušším způsobem pro řešení úlohy TSP. Princip tohoto algoritmu spočívá v tom, že je hledán nejbližší soused k městu, které bylo právě navštíveno. Na začátku se náhodně vybere město a, které bylo navštíveno, a od tohoto města je pak hledáno nejbližší nenavštívené město b. Dále se pak pokračuje stejným způsobem, dokud nejsou všechna města navštívena. Tímto způsobem bylo každé město navštíveno právě jedenkrát a bylo zjištěno přijatelné řešení. Avšak takovéto řešení se může podstatně lišit od optimálního řešení. Může nastat situace, kdy tato cesta může mít zbytečně drahé úseky, i když existuje i levnější varianta [7].

Na obrázku 4.1 je zobrazena situace, kdy hladový algoritmus využije při hledání cesty trasu začínající ve městě A a dále bude pokračovat přes body B - C, kde však tato trasa je velmi drahá. Celková velikost trasy A - B - C - D - A tedy

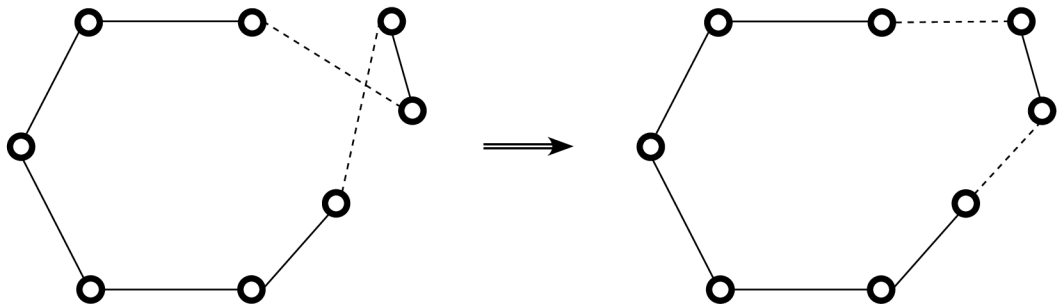


Obr. 4.1: Ukázka náhodného cestujícího [7]

bude 3+11+2+5 = 21. Přitom optimální řešení představuje trasa A - C - D - B - A s celkovou velikostí 4+2+8+3 = 17.

4.2.2 Heuristický algoritmus

Heuristický algoritmus je založen na lokálním prohledávání. Nejjednodušším algoritmem je tzv. algoritmus *2-výměny*. Nejdříve je provedena náhodná permutace všech měst a pojmenuje se jako trasa T. Ostatní sousední cesty trasy T se definují jako množina všech tras, které se mohou získat tak, že dojde k výměně dvou nesousedních uzlů v trase T. Tato operace se nazývá *2-výměny* a je zobrazena na obrázku 4.2 [11].



Obr. 4.2: Trasa T před a po použití algoritmu 2-výměny [7]

Nastane-li případ, kdy je nalezena trasa T', která dosahuje lepšího řešení než původní trasa T, pak tuto trasu nahradí. Pokud v okolí trasy T není žádná, která

by dosahovala lepšího řešení, trasa T je označena jako *2-optimální* a algoritmus se ukončí [11].

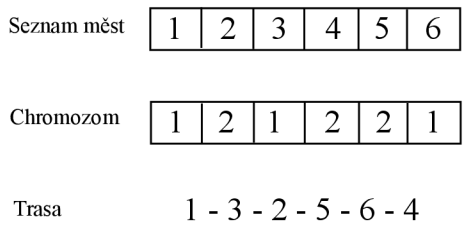
Heuristický algoritmus *2-výměny* lze jednoduše zobecnit na *k-výměny* tak, že se vybere k uzlů, které jsou vyměněny za lepší řešení trasy T. Pokud k představuje malé číslo, je celá oblast řešení nalezena rychle. V takovémto případě roste pravděpodobnost zjištění pouze suboptimálního řešení. Na druhou stranu, je-li velikost k velké číslo, zvyšuje se exponenciálně počet zjištěních sousedních řešení. V tomto případě se v praxi používá algoritmus $k > 3$ [13], [11].

4.3 Genetický algoritmus pro řešení TSP

Genetický algoritmus má oproti předchozím metodám značnou výhodu s jednoduchým a přirozeným ohodnocením funkce. Genetické algoritmy se pro problém TSP postupně zdokonalovaly a vznikaly tři způsoby reprezentace, kde je trasa popisována pomocí vektoru a délka se shoduje s počtem měst [7].

4.3.1 Ordinální reprezentace

Tato reprezentace je již většinou historického záznamu. Jedná se o nejstarší z metod pro řešení náhodného cestujícího pomocí GA. Při této reprezentaci je použito jednobodové křížení které má výhodu v tom, že z rodičů vznikají takoví jedinci s řešením, kde není zapotřebí použít jakoukoliv opravnou funkci. Všechny trasy jsou v tomto případě zakódovány do seznamu měst, kde každé město má přiřazeno své pevné pořadí [12], [7].



Obr. 4.3: Ordinální reprezentace [7]

Výsledný chromozom který vznikne se následně pomocí seznamu interpretuje gen po genu zleva doprava. Při ohodnocení n-tého genu, který má svoji hodnotu h se následně vybere h-tý prvek ze seznamu měst a je přidán k vytvářené cestě.

Pro konkrétní případ který je uveden na obrázku 4.3, bude první gen s hodnotou 1 určovat, že první prvek v seznamu měst bude začínat městem s číslem 1. Toto město se pak ze seznamu odstraní. Druhý gen v pořadí, který má hodnotu 2 vybere pak ze seznamu město, které je na druhém pořadí. Protože město s číslem 1 již bylo odstraněno, nyní se na druhé pozici nachází město s číslem 3. Takto se pokračuje stále dokola, dokud nevznikne celá okružní trasa, která pak bude vypadat: 1-3-2-5-6-4-1 [7].

Tato metoda, kde se používá pouze ordinální reprezentace a jednobodové křížení, není příliš vhodná pro řešení obchodního cestující. Proto v této metodě nelze dosáhnout dobrých výsledků a proto bylo takovéto řešení zamítnuto [7].

4.3.2 Sousedská reprezentace

Sousedská reprezentace je určena k usnadnění manipulace hran. Operátor křížení vytváří na základě této reprezentace potomstvo, které dědí většinu z těchto hran z mateřských vektorů. V tomto případě město j obsadí pozici i na vektoru tehdy, pokud existuje hrana na trase z města i do města j. Například: vektor 3-8-5-2-6-4-1-4 reprezentuje trasu 1-3-5-6-4-2-8-7. Město 3 zaujímá pozici 1 ve vektoru, protože hrany 1 a 3 jsou na trase. Stejně tak, město 8 zaujímá pozici 2, protože hrany 2 a 8 jsou na trase. Sousedská reprezentace se však pro řešení problému TSP neukázala jako příliš efektní.[12].

4.3.3 Reprezentace cesty

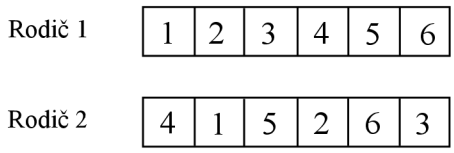
Na rozdíl od ordinální reprezentace je reprezentace cesty přirozený způsob jak zakódovat TSP trasy. Operátor křížení na základě této reprezentace obvykle vytváří potomky, které dědí buď relativní nebo absolutní pozici měst z mateřského chromozomu [12].

Operátor křížení s částečným přiřazením

Operátor křížení s částečným přiřazením PMX nejdříve náhodně vybere dva body řezu z obou rodičů za účelem vytvořit potomstvo. Nejdříve se mezi oběma rodiči vymění podřetězec a ostatní pozice zůstanou prázdné. Následně dojde k postupnému doplnění obou rodičů původními hodnotami tak, aby žádná z hodnot nebyla v chromozomu obsažena vícekrát. V posledním kroku se použije přepisovací předpis, kdy se vzájemně mezi sebou vymění dvě hodnoty z každého chromozomu aby opět nedošlo k situaci, kdy se vyskytne více než jeden prvek v chromozomu [12], [7].

Operátor křížení s rekombinací hran

Operátor křížení s rekombinací hran ERX je označován za nejúspěšnější operátor křížení pro řešení náhodného cestujícího. Při tomto operátoru křížení vzniká ze dvou rodičů pouze jeden potomek. Potomek obsahuje každý úsek cesty po obou z rodičů. Při vytváření potomka se používá tzv. hranová tabulka, která ke každému městu připíše seznam jeho sousedů z každého rodiče [7]. Počáteční chromozomy obou rodičů představuje obrázek 4.4.



Obr. 4.4: Počáteční chromozomy [7]

Ze vstupních hodnot obou rodičů bude mít hranová tabulka 4.1 následující tvar.

Tab. 4.1: Hranová tabulka [7]

Město	Sousedé	Město	Sousedé
1	2 4 5	4	1 3 5
2	1 3 5 6	5	1 2 4 6
3	2 4 6	6	2 3 5

Poté se postupně začne vytvářet cesta. Princip funguje tak, že je vybráno některé město z nejmenším počtem sousedů a toto město je pak z hranové tabulky odstraněno. V tomto případě se jedná o město s číslem 1. Hranová tabulka 4.2 pak bude mít podobu:

Tab. 4.2: Hranová tabulka po prvním kroku [7]

Město	Sousedé	Město	Sousedé
2	3 5 6	5	2 4 6
3	2 4 6	6	2 3 5
4	3 5		

V dalším kroku se ze sousedních měst čísla 1 (jedná se tedy o 2, 4 a 5), které bylo v předchozím kroku odstraněno, opět vybere další město s nejmenším počtem sousedů. V tomto případě bude vybráno číslo 4 a hranová tabulka 4.3 se upraví do tvaru:

Tab. 4.3: Hranová tabulka po druhém kroku [7]

Město	Sousedé	Město	Sousedé
2	3 5 6	5	2 6
3	2 6	6	2 3 5

Tento postup se stále opakuje, dokud nevznikne nový potomek. Pro uvedený příklad bude mít potomek tvar 1-4-3-2-5-6.

Použitím operátoru křížení s rekombinací hran lze docílit nejlepších výsledků. Potomek v tomto případě přebírá od svých rodičů dobré úseky konečné trasy [7].

5 SYMBOLICKÁ REGRESE

Podle literatury [6] je symbolická regrese jednou ze základních aplikací v GP a používá se pro řešení mnoha typů problémů, kde není známa závislost mezi proměnnými. Úkolem symbolické regrese je hledání takové funkce, zapsanou pomocí syntaktického stromu již s předepsanými operacemi, která dokáže nejlépe aproximovat data z trénovací množiny. Syntaktický strom byl již popsán v kapitole genetického programování. Při použití syntaktického stromu v symbolické regresi platí, že:

1. Koncové uzly tvoří nezávislé proměnné (x, y, z, ...) nebo celočíselné nezáporné konstanty (0, 1, 2, ...)

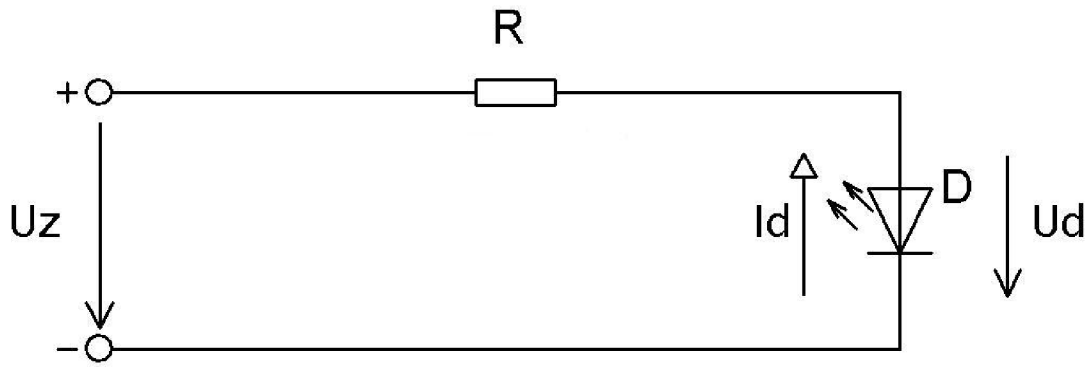
2. Funkcionální uzly tvoří jednoduché operace, které jsou unární, binární, ternární, ...

5.1 Názorný příklad symbolické regrese

Jako ilustrativní příklad pro symbolickou regresi byl použit jednoduchý elektrický obvod [1], kde je počítán předřadný odpor k LED diodě. K výsledku nalezení tohoto odporu R slouží vzorec:

$$R = \frac{U_Z - U_D}{I_D} \ [\Omega],$$

kde R je rezistor, I_D značí proud protékající LED diodou, U_Z je napětí na zdroji a U_D představuje napětí na LED diodě. Schéma tohoto elektrického obvodu je znázorněno na obrázku 5.1.



Obr. 5.1: Schéma elektrického obvodu předřadníku pro LED diodu

[1] Elektrický obvod předřadníku byl použit ze stránky http://elektrolab.wz.cz/?nizkenapeti=25

Aby mohl být na tomto příkladě ukázán princip symbolické regrese, bude nyní předpoklad, že vzorce pro výpočet předřadného odporu a Ohmova zákona nejsou dosud známy. Při hledání řešení pomocí genetického programování budou použity naměřené hodnoty pro tento obvod z tabulky 5.1 po dvanácti provedených měření. Hodnoty z tabulky představují trénovací množinu.

Tab. 5.1: Tabulka naměřených hodnot

Číslo měření	U_Z $[V]$	U_D $[V]$	I_D $[A]$	R $[\Omega]$
1	14,3	2,9	0,050	228
2	12,0	3,2	0,020	440
3	11,5	2,5	0,018	500
4	11,7	3,0	0,025	348
5	11,2	2,8	0,020	420
6	11,1	2,4	0,030	290
7	10,0	3,1	0,015	460
8	10,8	2,7	0,025	324
9	10,2	1,9	0,010	830
10	9,1	2,5	0,012	550
11	8,7	1,2	0,015	500
12	8,0	2,6	0,018	300

Při všech měření docházelo k různým kombinacím hodnot všech veličin. První tři sloupce veličin v tabulce představují hodnoty zvolené pro měření, poslední dva pak naměřené hodnoty. Dle [7] bude cílem nalézt výslednou funkci v symbolickém tvaru, která nejlépe aproximuje vztah pro výpočet hodnoty rezistoru R.

5.2 Řešení pomocí genetického programování

Na začátku se nadefinují množiny terminálů a množiny funkcí. Tyto množiny budou obsahovat prvky, které se následně použijí ve stromové struktuře genetického programování. Množina terminálů bude obsahovat prvky $T = \{U_Z, I_D, U_D\}$ a množina funkcí $F = \{+, -, *, /\}$. Dále tyto množiny musí splňovat podmínky postačitelnosti a uzavřenosti [7]. Tyto podmínky již byly popsány dříve.

V dalším kroku se určí ohodnocení (*fitness*) funkce. K tomuto ohodnocení se použijí výsledky všech 12ti měření, které představují trénovací množinu funkce. Ohodnocení se získá pomocí zjištěného rozdílu mezi hodnotou vypočítanou programem

a naměřenou hodnotou odporu R postupně u všech měření. Provede se součet absolutních hodnot těchto rozdílů, který se nazývá celková chyba chromozomu. Tento součet bude určovat kvalitu. Pokud je rozdíl hodnot velký, jedná se o špatný program. Blíží-li se tato hodnota rozdílu k nule, pak program relativně dobře aproximuje hledaný výraz. Chybu *n-tého* chromozomu lze vyjádřit rovnicí 5.1:

$$e\left(n\right) = \sum_{k=1}^{12} \left|e\left(n,k\right) - R\left(k\right)\right|, \tag{5.1}$$

kde e(n, k) je vypočtená hodnota celkového odporu *n-tým* chromozomem na základě *k-tého* řádku z tabulky hodnot a R(k) značí hodnotu celkového odporu, která byla zjištěna měřením v *k-tém* čísle měření. V posledním kroku se nastaví parametry procesu, kterými jsou velikost populace, pravděpodobnost křížení a pravděpodobnost mutace [7].

6 PRAKTICKÁ ČÁST

V této kapitole budou popsána podrobná řešení dvou optimalizačních úloh. První část obsahuje řešení optimalizační úloha problému obchodního cestujícího pomocí genetického algoritmu. Ve druhé části je pak nalezeno optimální řešení příkladu na symbolickou regresi za pomoci genetického programování. U obou metod řešení jsou použity funkce a operátory, které byly popsány již v dřívějších kapitolách. Řešení těchto dvou příkladů je zapsáno programovacím jazykem JAVA v programu Eclipse.

6.1 Řešení problému TSP

Problém obchodního cestujícího je jednou z nejznámějších optimalizačních úloh. Důkladně byl tento problém již popsán v teoretické části. Tato úloha slouží pro nalezení nejkratší vzdálenosti mezi několika námi zvolenými městy v dané oblasti. Během této úlohy může být každé město navštíveno pouze jednou a začátek a konec musí být ve stejném městě.

6.1.1 Popis programu obchodního cestujícího

Celé praktické řešení zapsáno v programovacím jazyce JAVA obsahuje v našem případě celkem 8 tříd. Pro pořádek si tyto třídy můžeme rozdělit na:

- entitní třídy: *Mesta, Chromozom*

- manažerské třídy: *Krizeni, Mutace, Populace, Selekce, Republika*

- pomocné třídy: *NahodneCislo, ChromozomFitnessComparator*

- spustitelná třída: *Main*

První vytvořenou třídou v programu je třída ***Mesta***. Do této třídy se přiřazují mnou vytvořená města s jejich souřadnicemi na ose x a y a názvem.

Do následující třídy ***Republika*** se uloží seznam všech měst, které jsme si vytvořili. Republika bude představovat tzv. oblast ve které se všechna města nacházejí. Maximální počet měst, která tato oblast může obsahovat, jsem nastavil na hodnotu 100. Tento parametr lze libovolně upravit podle potřeby daného uživatele. Pro naše účely bude tento počet zcela postačující, protože budeme hledat optimální řešení mezi 25ti městy. Následně tato třída obsahuje funkci pro výpočet vzdálenosti mezi městy. Výpočet vzdálenosti byl řešen pomocí Pythagorovy věty.

V dalším kroku budou vytvořená vstupní města zapsána do chromozomu a každý chromozom dostane své ohodnocení fitness. To se provádí ve třídě ***Chromozom***. Nejdříve se do chromozomu zapíší města v náhodném pořadí a následně dochází k výpočtu vzdáleností mezi sousedními městy. Pokud tedy máme v chromozomu pořadí měst Brno - Praha - Ostrava - Liberec - Pardubice, vypočítají se vzdálenosti mezi dvojicemi měst Brno - Praha, Praha - Ostrava, Ostrava - Liberec, atd. Poté co se projede takto celý chromozom, dochází k výpočtu vzdálenosti mezi posledním a prvním městem v chromozomu. V našem případě tuto dvojici tvoří Pardubice - Brno. Po výpočtu vzdáleností mezi všemi dvojicemi se vytvoří suma všech vzdáleností. Protože ohodnocení fitness musí být od 0 do 1, pak výsledná fitness bude vypočítána jako převrácená hodnota sumy vzdáleností.

Další vytvořenou třídou je ***Selekce***. V této třídě budou vybrány chromozomy které budou následně tvořit novou populaci, ostatní chromozomy pak podstoupí proces křížení a popř. mutace. K praktickému řešení jsme použili selekci *ruletovou*. Při ruletovém výběru byl použit následující postup. Nejdříve byly sečteny hodnoty fitness všech vytvořených chromozomů. Tento součet tvořil 100 % kola rulety. Poté byla kazdému chromozomu přiděla taková část výseče rulety, jak velká byla hodnota fitness daného chromozomu z celkového součtu. Podle teorie platí, že čím větší výseč je danému chromozomu přidělena, tím je větší pravděpodobnost, že bude vybrán do nové populace. Tento výběr je však zcela náhodný a ne vždy je vybrán nejlepší jedinec. Za účelem náhodného výběru z kola rulety byla vytvořena pomocná třída ***NahodneCislo***, kde je vygenerováno náhodné číslo od 0 do 100 a tak vybrána hodnota spadající do některé z výseče rulety.

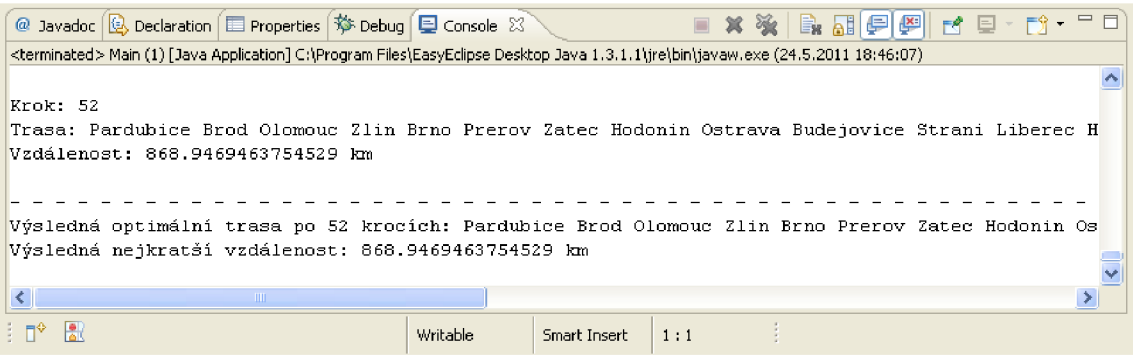
Poté co byly vybrány chromozomy které vstupují přímo do nové populace, ostatní chromozomy čeká proces křížení. V tomto procesu ze dvou rodičů vytvoří dva noví potomci. To je řešeno pomocí třídy ***Krizeni***. Existuje několik druhů křížení, v mém případě jsem zvolil *permutační křížení*. Nejdříve se náhodně vybere počet míst křížení a pak přesné pozice na chromozomech. Poté dojde k výměně jednotlivých měst mezi dvěma chromozomy. Aby však nenastala situace kdy bude jeden chromozom obsahovat po výměně některé město dvakrát, byla vytvořena funkce *opravDuplicity* která zkontroluje nově vytvořené chromozomy, a pokud jsou některá města v daném chromozomu obsažena dvakrát, tato funkce je vymění zpět do původních chromozomů.

U chromozomu může kromě křížení nastat také operace mutace. Pravděpodobnost mutace se však provádí jen s malou pravděpodobností. Tento operátor je prakticky řešen ve třídě ***Mutace***. Operátor mutace je odlišný od operátoru křížení. V mutaci se opět náhodně vyberou dvě pozice měst, které jsou pak vzájemně vyměněny. To se však děje pouze v jednom chromozomu.

Důležitou třídou v celém programu je třída ***Populace***. V této třídě se nastavují atributy, které ovlivňují výsledné řešení. Jednou z atribut je nastavení velikosti populace a dále lze nastavit procentuální pravděpodobnosti operátorů mutace a křížení. V této třídě také volíme počet nejlepších potomků, kteří putují ze selekce přímo do nové populace a počet kroků, kolikrát se má provést ověření, že nalezená trasa má nejkratší vzdálenost a představuje tak nalezené optimální řešení. V této třídě také vzniká nová populace, do které se vloží chromozomy měst ze selekce, křížení a mutace. Pomocí doplňkové třídy ***ChromozomFitnessComparator*** seřadíme chromozomy v nové populaci podle jejich fitness od nejlepšího. Následně je vypsána trasa nejlepšího chromozomu do konzole. Abychom se ujistili, že námi nalezená trasa představuje konečné řešení, provede se celý proces genetického algoritmu do té doby, dokud výsledná trasa nebude vícekrát ověřena. V tomto případě je počet ověření nastaven na 50 kroků. Pokud v 50ti posledních krocích nebude nalezena již kratší vzdálenost, dojde k ukončení procesu. Jinak dojde k přepsání aktuální trasy trasou lepší, čítač se vynuluje a běží nových 50 kroků. Po ukončení procesu je konečná výsledná vzdálenost vypsána spolu s počtem provedených kroků a celkovou vzdáleností do konzolového okna programu Eclipse.

Poslední třída, která bude popsána je spustitelná třída ***Main***. V této třídě jsou zadána všechny města mezi kterými hledáme optimální řešení problému obchodního cestujícího. Každému městu je přiřazena pozice na ose x, y a název města. Pro simulaci problému jsem vytvořil 25 měst.

Po spuštění programu obchodního cestujícího se v konzoli vypíšou potřebné údaje o právě provedeném výpočtu. Jsou zde vypsány údaje kolik kroků bylo provedeno pro nalezení nejkratší trasy, dále lze zjistit celkovou vzdálenost mezi všemi městy a pořadí v jakém byla jednotlivá města navštívena. Ukázka výpisu konzolového okna je zobrazen na obrázku 6.1.



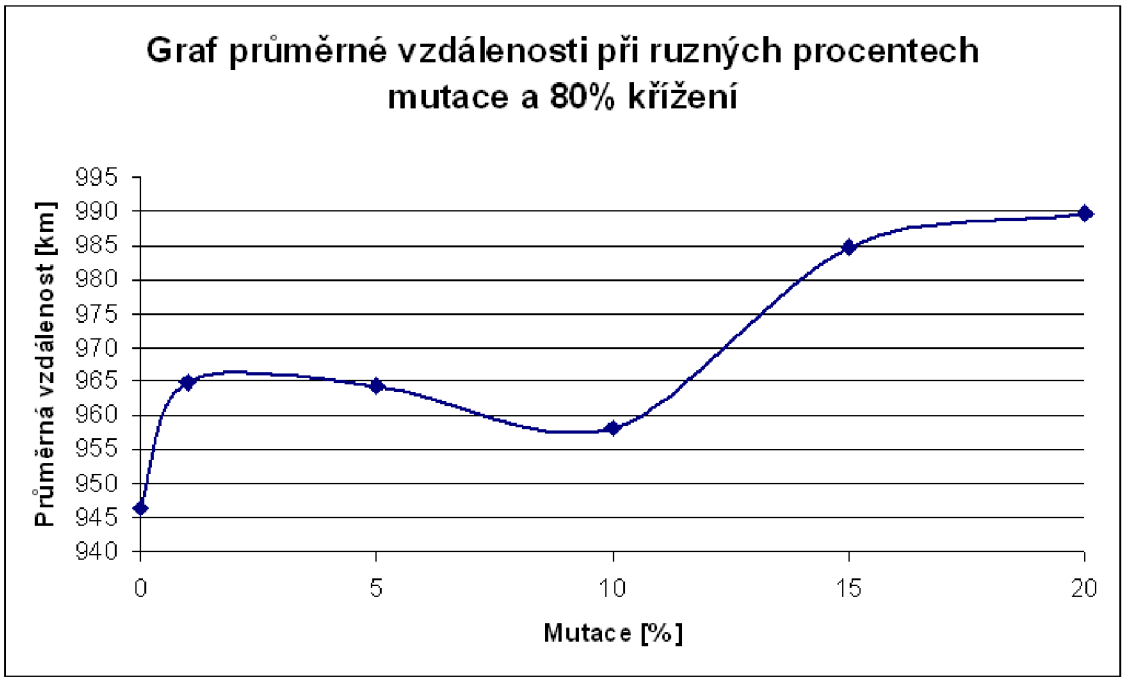
Obr. 6.1: Výsledný výpis do konzole programu Eclipse

6.1.2 Měření v TSP

V problému obchodního cestujícího se pokoušíme nalézt co nejkratší možnou vzdálenost mezi několika městy. Tato výsledná vzdálenost bývá ovlivněna nastavením jednotlivých parametrů při hledání řešení.

Jedním z těchto parametrů je počet chromozomů které jsou v selekci vybrány a postupují přímo do výsledné populace. Dalším z parametrů je procento pravděpodobnosti křížení. Udává se, že při vytváření nové populace se křížení používá s 80 % pravděpodobností. Posledním parametrem který má vliv na výsledném řešení je pravděpodobnost mutace.

Abych ověřil jaký vliv má mutace na vznik nové populace, rozhodl jsem se porovnat výsledné vzdálenosti při různém nastavení procentuální pravděpodobnosti mutace. Počet chromozomů, které jsou vybrány přímo do nové populace, jsem u všech měření nastavil na hodnotu 1. Do nové populace bude tedy vcházet přímo pouze jeden chromozom. Pravděpodobnost křížení jsem opět ponechal stejné pro všechna měření. Křížení bylo provedeno s pravděpodobností 80 %. Rozsah pravděpodobnosti mutace byl zvolen od 0 % do 20 % s krokem 5 %. Pro každou hodnotu mutace bylo následně provedeno 10 měření. Protože výsledná vzdálenost se po každém měření liší, byly po provedení všech 10ti měření tyto výsledky pro každou hodnotu mutace zprůměrovány. Nejkratší nalezené vzdálenosti odpovídají nejlepšímu nalezenému jedinci v populaci. Takto se postupovalo i pro další hodnoty mutace.



Obr. 6.2: Graf při různém nastavení pravděpodobnosti mutace a 80 % křížení

Z výsledného grafu na obrázku 6.2 lze zjistit, že nejkratší vzdálenosti jsou dosaženy při použití 0 % mutace. S přibývajícími hodnotami pravděpodobnosti mutace dostáváme výsledné trasy s delšími vzdálenostmi. Při použitím větší pravděpodobnosti mutace je tedy výsledná vzdálenost spíše negativně ovlivněna. V tabulce 6.1 jsou zobrazeny výsledné vzdálenosti při použití 0% mutace a 80 % křížení.

Tab. 6.1: Výsledky měření při 0 % mutace a 80 % křížení

Měření	Počáteční vzdálenost	Výsledná vzdálenost
1	1085,91 km	919,22 km
2	1046,85 km	943,35 km
3	1012,95 km	961,52 km
4	1035,92 km	954,40 km
5	1028,89 km	976,46 km
6	1057,13 km	1027,20 km
7	1060,25 km	***800,25 km***
8	1024,56 km	1023,30 km
9	1026,37 km	914,25 km
10	985,18 km	945,76 km
	Průměrná vzdálenost:	*946,57 km*

Přestože se tyto hodnoty v každém měření liší, vytvořený program funguje zcela správně. Tyto rozdílné výsledky jsou ovlivněny tím, že při každém měření vznikají zcela nové chromozomy měst které tvoří výslednou populaci.

6.2 Řešení symbolické regrese

Jak již bylo popsáno dříve, nyní se bude jednat o řešení optimalizační úlohy pomocí GP. Používá se v případech, kdy hledáme závislost mezi vstupními a výstupními proměnnými. Jako příklad který bude názorně vyřešen byl vybrán elektrický obvod pro výpočet předřadného odporu k diodě 5.1. Nyní budeme samozřejmě předpokládat, že tento výpočet neznáme. Výsledek pak bude zobrazen pomocí syntaktického stromu.

6.2.1 Popis programu pro výpočet symbolické regrese

Pro praktické řešení symbolické regrese byl použit již předem vytvořený *framework* podle [17]. Tento framework obsahuje třídy, které jsou využívány pro řešení genetického programování. Tyto třídy obsahují jednotlivá pravidla, kterými je genetické programování řízeno, dále třídy pro vytvoření terminálních a neterminálních uzlů stromu, a třídy obsahující operace mutace, křížení, klonování chromozomů, výběr jedinců do nové populace a ohodnocení fitness.

V následujícím kroku bylo potřeba nadefinovat třídy které budou obsahovat funkce, tedy neterminální uzly chromozomu. Tyto třídy jsou umístěny v balíčku (package) ***examples***. Pro tento příklad byly vytvořeny třídy aritmetické funkce sčítání, odčítání, násobení a dělení. Funkci sčítání obsahuje třída s názvem ***PlusAction***, funkci mínus třída ***MinusAction***, funkci násobení představuje třída ***TimesAction*** a funkci dělení třída ***DivisionAction***.

V dalším kroku byly vytvořeny třídy pro terminální uzly. Tyto terminální uzly budou představovat vstupní proměnné UD, UZ, ID. Zdrojové kódy jsou pro všechny proměnné stejné, liší se pouze *názvem třídy* pro jednotlivou proměnnou, *konstruktorem* a vracející metodou *getSymbol()*. Tyto třídy terminálních uzlů jsou opět umístěny v balíčku *examples*.

Poté co byly vytvořeny neterminální a terminální uzly, je nyní potřeba nadefinovat třídu pro ohodnocení fitness každého jedince, která bude odpovídat námi zvolenému příkladu. Tady naměřené výsledky ve všech 12ti měření tvoří trénovací množinu. Pro ohodnocení se nejdříve vypočítá rozdíl hodnoty zjištěné z naměřené tabulky a hodnoty vypočtené programem pro všechna měření. Následně se provede součet absolutních hodnot těchto rozdílů a tím se vypočítá chyba jedince. K tomuto výpočtu chyby byla podle literatury [7] použita rovnice 6.1:

$$e\left(i\right) = \sum_{j=1}^{12} \left|e\left(i,j\right) - R\left(j\right)\right|, \tag{6.1}$$

kde e(i, j) je vypočtená hodnota celkového odporu *i-tým* chromozomem na základě *i-tého* řádku z tabulky hodnot a R(j) značí hodnotu celkového odporu, která byla zjištěna měřením v *i-tém* čísle měření. Po provedení výpočtu chyby se přistoupí k samotnému ohodnocení. Protože ohodnocení fitness používá hodnoty od 0 do 1, je nutné toto ohodnocení vypočítat jako převrácenou hodnotu chyby. Tato fitness funkce je vytvořena ve třídě ***FitnessRegression*** v balíčku *examples*.

Jako poslední krok byla v balíčku *examples* vytvořena třída ***RegressionTest***. Tato třída kompletuje celý příklad a spouští celý proces. Tato třída obsahuje bezkontextovou gramatiku, která je vytvořena formou BNF (Backus-Naurova forma).

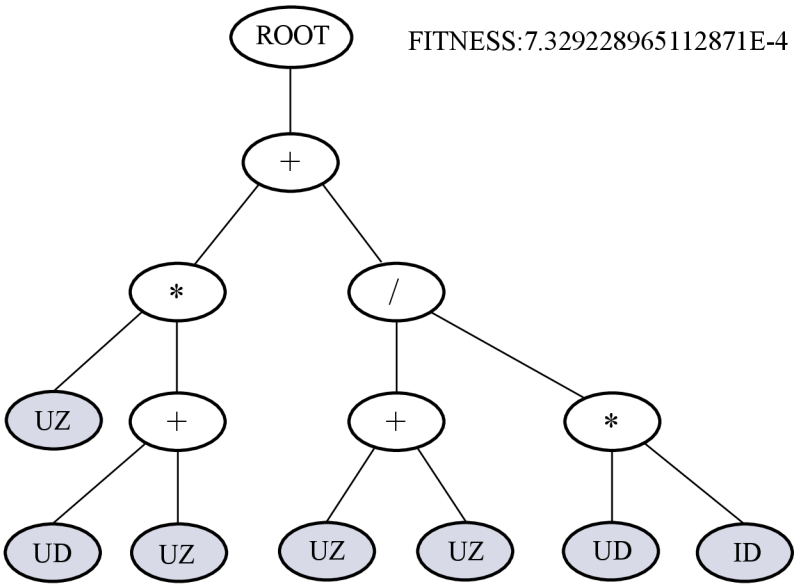
Genetické programovaní je v tomto případě zapsáno pomocí čtveřice $\{N, T, P, S\}$. N představuje množinu neterminálních uzlů (funkcí). T pak značí všechny terminální uzly (terminály). Symbol P značí konečnou množinu přepisovacích pravidel a S počáteční neterminály [18]. Pro tento zvolený příklad množina terminálů obsahuje proměnné $\{UD, UZ, ID\}$, množina funkcí (neterminálů) pak algebraické výrazy $\{+, -, *, /\}$. Dále je v této třídě uložena tabulka naměřených hodnot 5.1 se všemi vstupními a výstupními hodnotami. Výsledkem pak získáváme vygenerovaný kód nejlepšího chromozomu ze kterého pak pomocí programu `GraphViz` bude sestaven syntaktický strom. Výpis bude dále obsahovat ohodnocení fitness nejlepšího jedince v populaci.

Stejně jako tomu bylo u řešení problému obchodního cestujícího, i v tomto případě lze nastavit různé parametry procesu genetického programování. Tyto parametry se nacházejí v konfiguračním souboru *config_default.ini*. V tomto souboru je možné nastavit velikost populace (population.size), počet provedených evolucí (evolutions), procentuální poměr křížení a mutace (crossover_rate, mutation_rate). Pro řešení mnou zvoleného problému byla velikost populace nastavena na hodnotu 20, počet evolucí na hodnotu 40, procento křížení 90 % a mutace 5 %. Také je potřeba nastavit maximální hloubku stromu (max_tree_height), v tomto případě jsem volil hodnotu 6, která bude dostačující.

Při výsledku hledání optimálního řešení byla zjištěna výsledná hodnota fitness $7{,}2992 * 10^{-4}$. Tato hodnota představuje velmi dobrý výsledek. Spolu s hodnotou fitness byl vygenerován i kód ze kterého bude následně získán výsledný syntaktický strom. Pro vykreslení syntaktického stromu se použije program **GraphViz** do kterého bude tento kód vložen.

Po provedení všech předchozích kroků byl získán výsledný syntaktický strom zvoleného příkladu symbolické regrese. Tento strom je zobrazen na obrázku 6.3.
Z tohoto obrázku lze zjistit, že výsledný syntaktický strom má hloubku 3 a odpovídá výrazu 6.2:

$$R = U_Z \cdot (U_D + U_Z) + \frac{(U_Z + U_Z)}{(U_D \cdot I_D)} \tag{6.2}$$



Obr. 6.3: Výsledný syntaktický strom

6.2.2 Měření symbolické regrese

Jako poslední krok jsem provedl několik měření pro symbolickou regresi abych zjistil v jaké rozsahu hodnot se budou pohybovat výsledky ohodnocení fitness. U všech měření jsem ponechal nastavení parametrů stejné. Mutace probíhá s pravděpodobností 5 % a křížení 90 %. Hloubka stromu je nastavena na hodnotu 6.

První nalezený výsledek byl již popsán výše. V dalším měření bylo nalezeno optimální řešení s ohodnocením fitness $15{,}7086 * 10^{-4}$ odpovídající matematickému výrazu 6.3:

$$R = \frac{U_Z}{I_D} + I_D \tag{6.3}$$

Ve třetím měření nalezené optimální řešení mělo hodnotu fitness $7{,}2992 * 10^{-4}$. Po přepsání ze syntaktického stromu měl nalezený výraz tvar 6.4:

$$R = (U_Z + (U_Z - I_D)) \cdot (((U_Z - I_D) + U_D) \cdot U_D) \tag{6.4}$$

Následně došlo k algebraické úpravě všech nalezených výrazů. I když se hodnota fitness blížila ideální nulové hodnotě, přesto se po této úpravě nepodařilo najít správnou hledanou funkci. Ani následné další pokusy a změny nastavení křížení a mutace nevedly ke správným výsledkům. V každém výsledku tyto výrazy postrádaly některou z potřebných proměnných nebo algebraických výrazů.

7 ZÁVĚR

Cílem této bakalářské práce bylo seznámit se s evolučními algoritmy a prostudovat teorii evolučních optimalizačních technik, především genetických algoritmů a genetického programování. Následně ze získaných znalostí popsat a prakticky vytvořit programy, z nichž jeden bude použit pro optimalizaci řešení pomocí genetických algoritmů a druhý pro řešení optimalizační úlohy pomocí genetického programování.

V teoretické části jsem nejdříve všeobecně shrnul a popsal význam a vznik evolučních algoritmů a části, ze kterým se skládají. Následně pro řešení optimalizačních problémů bylo potřeba se zaměřit na genetický algoritmus a genetické programování. Důkladně byla rozebrána jejich podstata, všechny důležité části ze kterých se genetické algoritmy a genetické programování skládají, kódování a v neposlední řadě také operátory, které se při nalezení optimálních řešení používají.

Praktické řešení bylo rozděleno na dvě části. V první části jsem se zaměřil na klasickou optimalizační úlohu náhodného obchodního cestujícího, která využívá při nalezení optimálního řešení genetické algoritmy. Praktické řešení bylo implementováno do programovacího jazyku JAVA v programu Eclipse. Do programu byly zadány počáteční vstupní informace mezi jakým počtem měst bude hledána nejkratší trasa, jednotlivé souřadnice a názvy těchto měst. Za pomoci genetických operátorů selekce, mutace a křížení bylo hledáno nejoptimálnější spojení mezi jednotlivými městy, kde žádné město nesmí být navštíveno vícekrát, počátek a konec trasy musí být ve stejném městě. Pokud byly provedeny všechny kroky a byla nalezena nejkratší vzdálenost, v konzolovém okně programu Eclipse se vypsaly údaje o celkovém počtu provedených kroků vedoucí k výslednému řešení, celková nejkratší vzdálenost a pořadí v jakém byla města navštívena. Také bylo provedeno několik měření ke zjištění vhodného nastavení procentuálního poměru operátoru mutace a křížení.

Ve druhé části jsem si vybral elektrický obvod předřadného odporu k diodě, který představoval příklad na symbolickou regresi řešenou pomocí genetického programování. V tomto případě bylo považováno že vzorec pro výpočet elektrického obvodu není znám a k výslednému řešení se použily pouze údaje z tabulky naměřených hodnot. Opět bylo řešení příkladu prakticky řešeno v jazyce JAVA. V programu byly do terminálních uzlů vloženy hodnoty proměnných, do neterminálních uzlů použité algebraické výrazy a načtena tabulka naměřených hodnot. Ve výsledku byl vytvořen syntaktický strom řešení zadaného problému. Opět bylo provedeno více měření hledaného problému. Bylo však zjištěno, že i když hodnoty fitness určovaly dobrý výsledek, správné řešení syntaktického stromu programu nalezeno nebylo.

Tato práce měla za úkol seznámit se s evolučními algoritmy v moderní numerické matematice při nalezení řešení optimalizačních problémů a následném praktickém využití při simulování problému.

LITERATURA

[1] TEDA, J. *Genetické algoritmy a jejich aplikace v praxi.* [online]. 2004-2010, [cit. 2010-10-19]. Dostupné z WWW: http://programujte.com/?akce=clanek&cl=2005072601-geneticke-algoritma-jejich-aplikace-v-praxi. ISSN 1801-1586.

[2] LUNER, P. *Jemný úvod do genetických algoritmů.* [online].[cit. 2010-10-19]. Dostupné z WWW: http://cgg.mff.cuni.cz/ pepca/prg022/luner.html.

[3] KALÁTOVÁ, E; DOBIÁŠ, J. *Evoluční algoritmy.* [online].[cit. 2010-10-19]. Dostupné z WWW: http://www.kiv.zcu.cz/studies/predmety/uir/gen_alg2/E_alg.htm.

[4] OBITKO, M. *Introduction to Genetic Algorithms: Genetic Algorithm.* [online]. 1998, [cit. 2010-10-21]. Dostupné z WWW: http://www.obitko.com/tutorials/genetic-algorithms/ga-basic-description.php.

[5] SCHWARZ, J; SEKANINA, L. *Aplikované evoluční algoritmy EVO.* Brno: Vysoké učení technické, 2006.

[6] KVASNIČKA, V; POSPÍCHAL, J; TIŇO, P. *Evolučné algoritmy.* První vydání. Bratislava: Slovenská technická univerzita, 2000. 223 s. ISBN 80-227-1377-5.

[7] HYNEK, J. *Genetické algoritmy a genetické programování.* První vydání. Praha: Grada Publishing, a.s., 2008. 200 s. ISBN 978-80-247-2695-3.

[8] POSTERUS. *Evolučné a genetické algoritmy.* [online]. [cit. 2010-10-23]. Dostupné z WWW: http://www.posterus.sk/?p=3364.

[9] POSPÍCHAL, P. Diplomová práce *Akcelerace genetického algoritmu s využitím GPU.* Brno: Vysoké učení technické, 2008.

[10] KOPŘIVA, J. Diplomová práce *Srovnání algoritmů při řešení problému obchodního cestujícího.* Brno: Vysoké učení technické, 2009.

[11] PANUŠ, J. Disertační práce *Evoluční algoritmy v optimalizačních problémech veřejné správy.* Pardubice: Univerzita Pardubice, 2008.

[12] POTVIN, J. *Genetic algorithms for the traveling salesman problem.* Montréal, Québec: Univerzité de Montréal.

[13] MICHALEWICZ, Z; FOGEL, D. *How to Solve It: Modern Heuristics.* Springer, Berlin, 2000. ISBN 3-540-66061-5.

[14] WEISE, T. *Global Optimization Algorithms: Theory and Application.* [online]. Druhé vydání, 2009, Version: 2009-06-26, [cit. 2011-05-12]. Dostupné z WWW: http://www.it-weise.de/projects/book.pdf.

[15] MAŘÍK, V; ŠTĚPÁNKOVÁ, O; LAŽANSKÝ, J; a kol. *Umělá inteligence (4).* První vydání. Praha: Academia, 2003. 475 s. ISBN 80-200-1044-0.

[16] MACHÁČEK, M. Diplomová práce *Genetické programování v prostředí Mathematica.* Zlín: Univerzita Tomáše Bati ve Zlíně, 2010.

[17] KARÁSEK, J.; BURGET, R.; BENEŠ, R.; NAGY, Ľ. *Grammar Guided Genetic Programming for Automatic Image Filter Design. In The 10th International Conference on Knowledge in Telecommunication Technologies and Optics.* Ostrava: VSB - Technical University of Ostrava, 2010. s. 205-210. ISBN 978-80-248-2330-0.

[18] KASPŘÍKOVÁ, E. Diplomová práce*Analytické programování v C#.* Zlín: Univerzita Tomáše Bati ve Zlíně, 2010. s. 14

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

GA Genetický algoritmus – Genetic algorithm

EA Evoluční algoritmus – Evolution algorithm

GP Genetické programování – Genetic programming

TSP Problém obchodního cestujícího – Traveling Salesman Problem

PMX Operátor křížení s částečným přiřazením – Partially matched crossover

ERX Operátor křížení s rekombinací hran – Edge recombination crossover

BNF Backus-Naurova forma – Backus-Naur Form

SEZNAM PŘÍLOH

A Přílohy **54**
A.1 Přiložené DVD 54

A PŘÍLOHY

A.1 Přiložené DVD

Na přiloženém DVD jsou uloženy oba prakticky řešené programy v jazyce JAVA. Dále je zde software Eclipse ve kterém byly programy vytvořeny a software GraphViz pro vizuální zobrazení syntaktického stromu pro řešení příkladu symbolické regrese. DVD také obsahuje elektronickou podobu bakalářské práce a původní zdrojový kód v programu LaTeX.