VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Genetické programování v platformově nezávislém jazyce

Genetic Programming Based on a Platform Independent Language

2015 Zdeněk Gold

Tuto stránku nahradíte v tištěné verzi práce oficiálním zadáním Vaší diplomové či bakalářské práce.

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.*

Zde vložte text dohodnutého omezení přístupu k Vaší práci, chránící například firemní know-how. Zde vložte text dohodnutého omezení přístupu k Vaší práce, chránící například firemní know-how. A zavazujete se, že

1. o práci nikomu neřeknete,
2. po obhajobě na ni zapomenete a
3. budete popírat její existenci.

A ještě jeden důležitý odstavec. A ještě jeden důležitý odstavec. A ještě jeden důležitý odstavec. A ještě jeden důležitý odstavec. A ještě jeden důležitý odstavec. A ještě jeden důležitý odstavec. Konec textu dohodnutého omezení přístupu k Vaší práci.

V Ostravě 16. dubna 2009 .....

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

V Ostravě 16. dubna 2009 .....

Rád bych na tomto místě poděkoval všem, kteří mi s prací pomohli, protože bez nich by tato práce nevznikla.

Abstrakt

Klíčová slova: evoluční algoritmy, genetické programování, syntaktický strom, křížení, mutace, selekce, program, jedinec

Abstract

This is English abstract. This is English abstract. This is English abstract. This is English abstract. This is English abstract. This is English abstract.

Keywords: evolutionary, genetic, genetic programming

Seznam použitých zkratek a symbolů

GP	–	Genetické programování
GA	–	Genetický algoritmus

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Tento text je ukázkou sazby diplomové práce v LaTeXu pomocí třídy dokumentů `diploma`. Pochopitelně text není skutečnou diplomovou prací, ale jen ukázkou použití implementovaných maker v praxi. V kapitole **??** jsou ukázky použití různých maker a prostředí. V kapitole 11 bude „jako závěr“. Zároveň tato kapitola slouží jako ukázka generování křížových odkazů v LaTeXu.

2 Předmluva

Evoluční algoritmy vznikly jako řešení a jeden ze způsobů optimalizace v matematických a vědních kruzích. Celkem dlouho byl způsob optimalizace řešen dnes již klasickým matmatických aparátem, který je založen na infinitezimálním počtu, na variačních metodách aplikovaných ve funkcionálním prostoru nebo numerických metodách.

Tímto způsobem však lze nalézt optimální řešení pro problémy jednodušší charakter a pro ty složitější umožňuje nalézt pouze suboptimální řešení.

3 Evoluční algoritmy

Před tím, než se pustíme do popisu toho, co jsou to evoluční algoritmy a na jakých principech fungují, se seznámíme s tím, co jsou to „evoluční výpočetní techniky“.

3.1 Potřeba evolučních algoritmů

Před tím, než se pustíme do popisu toho, co jsou to evoluční algoritmy a na jakých principech fungují, se seznámíme s tím, co jsou to „evoluční výpočetní techniky“.

3.2 Charakteristika

Před tím, než se pustíme do popisu toho, co jsou to evoluční algoritmy a na jakých principech fungují, se seznámíme s tím, co jsou to „evoluční výpočetní techniky“.

3.2.1 Jedinec

Podle klasické Darwinové a Mendelovy teorie evoluce, je uznáváno dogma, podle něhož se jednotlivé druhy vyvíjejí tak, že jsou z rodičů plozeni potomci, kteří podléhají při svém vzniku mutacím. Rodiče a potomci nevhodní pro aktuální životní prostředí vymírají cyklicky po tzv. generacích, čímž uvolňují místo novým rodičům a jejich potomkům.

3.2.2 Populace

Podle klasické Darwinové a Mendelovy teorie evoluce, je uznáváno dogma, podle něhož se jednotlivé druhy vyvíjejí tak, že jsou z rodičů plozeni potomci, kteří podléhají při svém vzniku mutacím. Rodiče a potomci nevhodní pro aktuální životní prostředí vymírají cyklicky po tzv. generacích, čímž uvolňují místo novým rodičům a jejich potomkům.

Typickým rysem evolučních algoritmů je, že jsou založeny na práci s populací jedinců. Populace může být znázorněna jako matice NxM, kde sloupce představují jednotlivé jedince. Každý jedinec představuje aktuální řešení danéího problému. S každým jedincem je navíc spojena hodnota účelové funkce, která říká, jak vhodný je jedinec pro další vývoj populace.

Hlavní činnosti evolučních algoritmů je cyklické vytváření nových populací, tedy náhrada starách populací novými. To vše pomocí přesně definovaných matematických pravidel.

K vytvoření populace je třeba nadefinovat tzv. vzor, podle kterého se generuje celá počáteční populace. Ve vzoru jsou pro každý parametr konkrétního jedince definovány tři konstanty, a to typ proměnné a hranice intervalu, v němž může pohybovat hodnota parametru. Volba hranice je velmi důležitý krok, protože při jejich nevhodném zvolení se může stát, že budou nalezena řešení , která nebudou možné fyzikálně realizovat nebo nebudou mít opodstatnění. Další neméně důležtý význam hranic souvisí se samotným evolučním procesem. Může se stát, že daný problém bude reprezentován plochou, která

bude nabývat lokálních extrémů stále větších hodnot se vzrůstající vzdáleností od počátku. To způsobí, že evoluce bude nacházet stále nová řešení až do nekonečna. Je to způsobeno tím, že evoluční proces směřuje do stále hlubších a vzdálenějších extrémů.

Populace je na základě vzorového jedince vygenerována podle vzorce.

$\Theta(1 + \alpha)$.

Tento vztah zajišťuje, že všchny parametry jedinců budou náhodně vygenerovány uvnitř povolených hranic prostoru možných řešení.

Zobrazení o tom, jak kvalitně proběhla evoluce, se provádí pomocí tzv. historie vývoje hodnoty účelové funkce ve formě jednoduchého grafu. Na něm je vykreslena závislost vývoje účelové funkce na aktuálním počtu jejich ohodnocení. Jde o sekvenci nejhorších a nejlepších řešení z jednotlivých populací. Výhodnější je však zobrazení závislosti hodnoty účelové funkce na aktuálním počtu jejich ohodnocení. To proto, že u evolučních cyklů se provádí u jednotlivých algoritmů různý počet ohodnocení účelové funkce. U prvního případu může být pomalejší konvergence hodnoty účelové funkce zobrazena jako rychlejší a naopak. Skutečná informace o kvalitě evoluce je pak zkreslená. U druhého způsobu můžeme objektivně porovnávat různé typy algoritmů bez ohledu na jejich vnitřní struktuře. Kromě vývoje nejlepšího jedince je pak vhodné zobrazovat vývoj i nejhoršího jedince z populace, a to do jednoho grafu.

Vývoj populace musí být vždy konvergentní k lepším hodnotám, což znamená, že nemůže nikdy vykazovat divergenci. V daném algoritmu funguje tzv. „elitismus“, který slouží jako jakýsi jednosměrný filtr, jenž propouští do nové populace pouze ta řešení, která jsou lepší či stejně dobrá jako ta ze staré populace.

3.2.3 Mutace

Podle klasické Darwinové a Mendelovy teorie evoluce, je uznáváno dogma, podle něhož se jednotlivé druhy vyvíjejí tak, že jsou z rodičů plozeni potomci, kteří podléhají při svém vzniku mutacím. Rodiče a potomci nevhodní pro aktuální životní prostředí vymírají cyklicky po tzv. generacích, čímž uvolňují místo novým rodičům a jejich potomkům.

3.2.4 Křížení

Podle klasické Darwinové a Mendelovy teorie evoluce, je uznáváno dogma, podle něhož se jednotlivé druhy vyvíjejí tak, že jsou z rodičů plozeni potomci, kteří podléhají při svém vzniku mutacím. Rodiče a potomci nevhodní pro aktuální životní prostředí vymírají cyklicky po tzv. generacích, čímž uvolňují místo novým rodičům a jejich potomkům.

3.2.5 Elitismus

Podle klasické Darwinové a Mendelovy teorie evoluce, je uznáváno dogma, podle něhož se jednotlivé druhy vyvíjejí tak, že jsou z rodičů plozeni potomci, kteří podléhají při svém vzniku mutacím. Rodiče a potomci nevhodní pro aktuální životní prostředí vymírají cyklicky po tzv. generacích, čímž uvolňují místo novým rodičům a jejich potomkům.

3.3 Vybrané algoritmy

Před tím, než se pustíme do popisu toho, co jsou to evoluční algoritmy a na jakých principech fungují, se seznámíme s tím, co jsou to „evoluční výpočetní techniky“.

3.3.1 Genetický algoritmus (GA)

Podle klasické Darwinové a Mendelovy teorie evoluce, je uznáváno dogma, podle něhož se jednotlivé druhy vyvíjejí tak, že jsou z rodičů plozeni potomci, kteří podléhají při svém vzniku mutacím. Rodiče a potomci nevhodní pro aktuální životní prostředí vymírají cyklicky po tzv. generacích, čímž uvolňují místo novým rodičům a jejich potom

3.3.2 Particle Swarm

Podle klasické Darwinové a Mendelovy teorie evoluce, je uznáváno dogma, podle něhož se jednotlivé druhy vyvíjejí tak, že jsou z rodičů plozeni potomci, kteří podléhají při svém vzniku mutacím. Rodiče a potomci nevhodní pro aktuální životní prostředí vymírají cyklicky po tzv. generacích, čímž uvolňují místo novým rodičům a jejich potom

3.3.3 SOMA

Podle klasické Darwinové a Mendelovy teorie evoluce, je uznáváno dogma, podle něhož se jednotlivé druhy vyvíjejí tak, že jsou z rodičů plozeni potomci, kteří podléhají při svém vzniku mutacím. Rodiče a potomci nevhodní pro aktuální životní prostředí vymírají cyklicky po tzv. generacích, čímž uvolňují místo novým rodičům a jejich potom

4 Genetické programování (GP)

Všechny dříve popsané algoritmy vznikly a jsou většinou užitečné v případech, kdy hledáme určitou konfiguraci pro matematický model problému tak, abychom dosáhly určitých mezních hodnot. Popusťme však ještě uzdu své fantazii a představme si situaci, kdy budeme chtít řešit problém, u kterého si nejsme jistí, jaký má být správný postup v jeho řešení. V zásadě budeme chtít vytvořit algoritmus, který dokáže generovat jiné algoritmy (či programy), který budou daný problém řešit.

Představme si situaci, kdy budeme chtít na základě nashromážděných dat vytvořit systém, který bude tyto data celkem dobře číst a popisovat je. Můžeme si to osvětlit na příkladě s bankou, která vede záznamy o úvěrech spolu s informacemi o věřitelích. Může se jednat o velikosti rodinného rozpočtu, počtu členů v rodině, počtu pracujících atd. S těmito parametry bychom potom chtěli vytvořit program, který by predikoval, zda je klient vhodným kandidátem na úvěr nebo ne.

GP je svým způsobem rozšířením GA s takovou obměnou, která dovoluje využít místo řetězcové reprezentace s pevnou délkou zvolit hirearchickou strukturu, ve které snáze reprezentujeme počítačové programy. Řešením pro nějaký model problému už není pouze konfigurace, ale celý algoritmus (program).

Otcem GP je standfordský informatik John Koza [1, 2]. Díky konceptu, který vytvořil, lze využít při tvorbě programů stejných evolučních operátorů, jaké obsahuje GA (křížení, mutace).

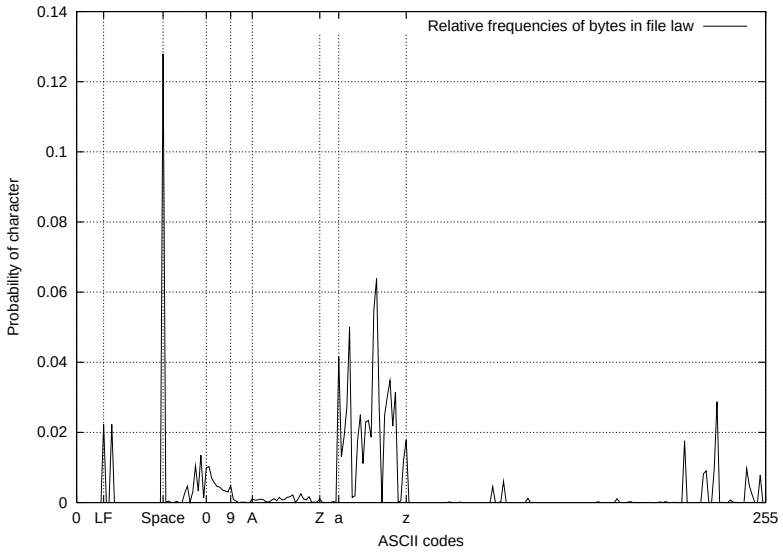
První problém, který po zrození myšlenky využít GA pro tvorbu a šlechtění počítačových programů je samotná reprezentace počítačového programu. Tato reprezentace musí být dostatečně obecná pro popsání v různých programovacích jazycích a zároveň zachovávat smyslupnost, syntaktickou správnost a spustitelnost nově vyšlechtěných programů při použití evolučních operátorů mutace a křížení.

Vhodnou reprezentaci programů nalezl Koza v jazyce LISP, který reprezentuje programy jako S-výrazy, což je prakticky syntaktický strom, kterým reprezentují svoje programy překladače. Syntaktický strom je tvořeny dvěma možnými typy uzlů. Jsou to buď neterminály (tvoří funkce) a terminály (proměnné a konstanty). Při definici problémů se specifikuje množina neterminálů $\Pi = \{f_1, f_2, ..., f_n\}$ a množina terminálů $\Gamma = \{t_1, t_2, ..., t_n\}$. Příkladem syntaktického stromu, který znázorňuje výraz pro kombinaci k-té třídy z n prvků je na obrázku.

Funkce f_i z množiny Π tvořící uzly syntaktického stromu jejichž argumenty reprezentují hrany vedoucí do hloubky o jedno větší. Terminály zastupují proměnné či konstanty umístěné v listových uzlech ukončují růst stromu a samotné individuum.

4.1 Omezení GP

Některé programové konstrukce, jako je cyklus či rekurze je díky syntaktického stromu nerealizovatelná. Proto Koza definuje [1] dvojí požadavky na množinu funkcí a terminálů. Je to požadavek uzavřenosti (closure) a postačitelnosti (sufficiency) obou množin. Uzavřenost množiny je splněna tehdy, pokud může libovolná funkce příjmout jako argument libovoulnou funkci z množiny funkcí či terminál z množiny terminálních sym-



Obrázek 1: Nějaký graf

bolů. Uzavřenost zamezí tvorbu syntakticky nesprávných programů. Postačitelnst nám naproti tomu umožňuje, abychom byli schopní k danému problému nalézt odpovídající program (funkci), která by jej řešila. Díky tomu jsme schopni říci, že k danému problému můžeme vyjádřit řešení daného problému. Představme si problém nalezení potravy umělým mravencem v mřížkové soustavěm kterou můžeme popsat množinou funkcí tvořenou příkazy $\{KROKVPRED, OTOCVLEVO, OTOCVPRAVO\}$ a množinou terminálů s jediným příkazem $\{ZASTAV\}$. I bez dokázaní si můžeme jasně říci, že jsme schopní díky těmto příkazům dostat mravence do jakéhokoliv místa a zastavit.

Protože výsledná funkce je tvořena nejen proměnnými, ale také konstantami, je třeba si říci, jak je při tvorbě jedince reprezentovat. Protože můžeme chtít použít GP pro ekonomické modelování, měl by mít algoritmus možnost generovat vhodné konstanty v daném rozsahu a daného typu podle řešené úlohy. Koza navrhuje množinu terminálů o náhodnou konstantu C tak, že se během běhu programu při ohodnocení individuí doplní o náhodnou hodnotu z příslučné množiny. Toto číslo se potom využije jako kterýkoliv jiný terminál.

4.2 Množina funkcí a terminálů

Aby byl algoritmus GP skutečně úspěšný a efektivní při hledání vhodného řešení, musíme velmi pečlivě definovat množinu použitých funkcí a také terminálních symbolů. Chybí-li v množině funkcí nějaký klíčový krok, můžeme se nám tvorba správného programu zesložitit či v extrémním případě nemusíme dojít k řešení vůbec. Naproti tomu, pokud budeme mít množinu funkcí příliš bohatou, rozšíří se prostor možných řešení a tedy i časová složitost pro prohledání celého prostoru. Může se stát, že i přesto, že je vý-

sledná funkce jednoduchá, bude přesto časově zdlouhavé, abychom toto řešení nalezli. Je to jako hledat jehlu v kupce sena. Vhodné je omezit funkce na co nejmenší počet.

4.3 Reprezentace stromu

Implementací syntaktického stromu je několik. Jednou z možností jak strom v počítačích reprezentovat je s použitím uzlů obsahující sezname potomků. Jde o přirozené vnímání stromové struktury, která je však při samotné implementaci dosti obtížně manipulovatelná.

Druhou možností je seznam, který představuje strom v prefixovém tvaru zápisu. Pořadí prvků v seznamu odpovídá pořadí navštívení každého uzlu ve stromě, při procházení stromu do hloubky. Zajištění zpětné rekonstrukce stromu nám umožňuje informace o tom, zda prvek patří do množiny terminálních symbolů nebo naopak do množiny funkcí a arita každé funkce. Příkladem takovéto stromové reprezentace je na obrázku níže.

4.4 Generování počáteční populace

Při prvotním generování populace se budeme zabývat, jakým způsobem lze náhodně vygenerovat několik individuí. Existují dvě metody řídící růst stromu. Úplná - všechny listové uzly stromu mají stejnou hloubku, která je rovna maximální velikosti stromu. Metoda růstová naproti tomu dovoluje rozmanitější stromy. Pro generování následného potomka se náhodně rozhoduje, zda je použitý termínál nebo uzel. I zde je limitní maximální hloubka stromu.

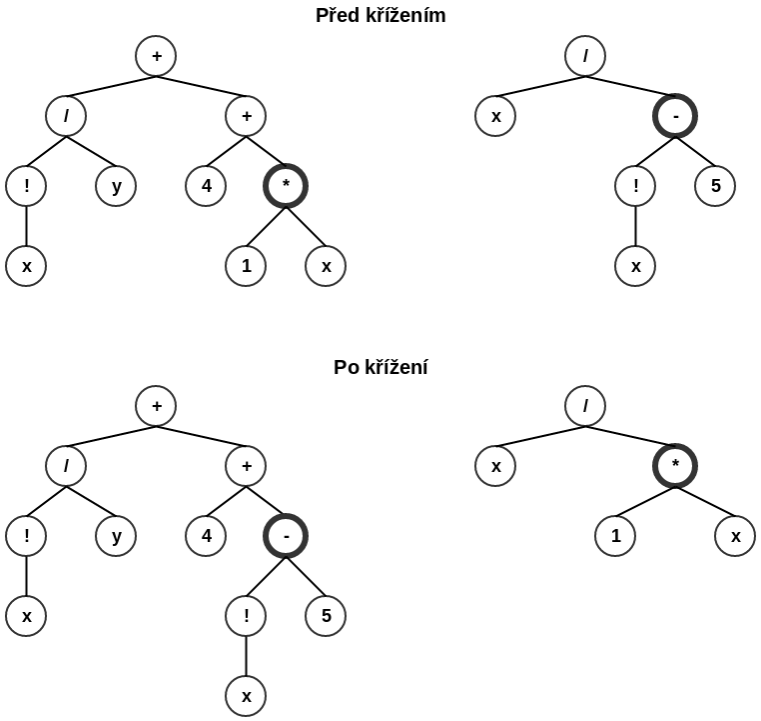
Koza ve své knize [1] doporučuje kombinaci obou metod, což v praxi znamená, že pro polovinu populace se použije růstová metoda a pro druhou polovinu úplná. Zároveň rovnoměrným rozložením hloubky stromů mezi všechny jedince, snížíme pravděpodobnost výskytu stejnývh řešení, která jsou v populaci nevyhovující.

Vše si můžeme ukázat na příkladě populace s $N = 500$ jedinců, kde je stanovena maximální hloubka $h_max = 6$. Při generování základní populace budeme postupovat tak, že 100 jedinců bude generováno s $h_max = 2$ (50 s růstovou metodou a 50 s úplnou metodou),následně pro dalších 100 řešení generujeme obdobné stromy hloubký o 1 vyšší $h_max = 3$ a tak dále, až do hloubky $h_max = 6$.

4.5 Operátory

V GP budeme používat stejné operace jako u GA. Operátor křížení pracuje s dvojícemi stromů. U každého stromu z dvojíce zvolí náhodně uzel pro křížení a v dalším kroku zamění podstromy pod zvolenými uzly mezi sebou. Více osvětlíme příkladem na obrázku.

Operátor mutace také náhodně vybere uzel stromu a celý jeho podstrom odstraní. Na tomto místě se poté náhodně vygeneruje chybějící část stromu do maximální hloubky předepsané parametrem.



Obrázek 2: Příklad operátoru křížení

4.6 Popis algoritmu

Před tím, než se pustíme do popisu toho, co jsou to evoluční algoritmy a na jakých principech fungují, se seznámíme s tím, co jsou to „evoluční výpočetní techniky“.

Příklad 4.1
Postup evolučního algoritmu

1. Vymezení parametrů evoluce – jako je stanovení kritéria ukončení (počet cyklu, vhodnost řešení,...), stanovení účelové funkce, případně tzv. vhodnosti. Účelovou funkcí se rozumí obvykle matematický model, jehož minimalizace/maximalizace vede k řešení

2. Generování prvopočáteční populace (obecně matice M x N, kde M je počet parametrů jedince a M je počet jedinců v populaci). Jedincem se rozumí vektor čísel s takovým počtem složek, kolik je optimalizovaných parametrů. Složky jsou nastaveny nahodile a každý jedinec představuje jedno možné řešení

3. Všichni jedinci se ohodnotí přes definovanou účelovou funkci a každému z nich se přiřadí: a) buď přímá hodnota vrácená účelovou funkcí, nebo b) vhodnost, což je upravená hodnota účelové funkce

4. Nastává výběr rodičů podle jejich kvality

5. Křížením rodičů se tvoří potomci. Proces křížení je u každého algoritmu odlišný.

6. Každý potomek je zmutován

7. Každý jedinec se ohodnotí stejně jako v kroku 3.

8. Vyberou se nejlepší jedinci

9. Vybraní jedinci zaplní novou populaci

10. Stará populace je zapomenuta a na její místo nastupuje populace nová. Dále se pokračuje krokem 4.

Evoluční algoritmy nejsou populární jen proto, že jsou moderní a odlišné od klasických, ale hlavně pro fakt, že v případě vhodného aplikování jsou schopny nahradit člověka ■

5 Implementace GP

Před tím, než se pustíme do popisu toho, co jsou to evoluční algoritmy a na jakých principech fungují, se seznámíme s tím, co jsou to „evoluční výpočetní techniky“.

5.1 UML model

Před tím, než se pustíme do popisu toho, co jsou to evoluční algoritmy a na jakých principech fungují, se seznámíme s tím, co jsou to „evoluční výpočetní techniky“.

5.2 Aplikace EvolTools

Před tím, než se pustíme do popisu toho, co jsou to evoluční algoritmy a na jakých principech fungují, se seznámíme s tím, co jsou to „evoluční výpočetní techniky“.

6 Testování na vybraných problémech

Před tím, než se pustíme do popisu toho, co jsou to evoluční algoritmy a na jakých principech fungují, se seznámíme s tím, co jsou to „evoluční výpočetní techniky“.

6.1 Problémy

Před tím, než se pustíme do popisu toho, co jsou to evoluční algoritmy a na jakých principech fungují, se seznámíme s tím, co jsou to „evoluční výpočetní techniky“.

6.1.1 Problémy

Před tím, než se pustíme do popisu toho, co jsou to evoluční algoritmy a na jakých principech fungují, se seznámíme s tím, co jsou to „evoluční výpočetní techniky“.

6.1.2 Sudoku

Před tím, než se pustíme do popisu toho, co jsou to evoluční algoritmy a na jakých principech fungují, se seznámíme s tím, co jsou to „evoluční výpočetní techniky“.

6.1.3 Aproximace modelu

Před tím, než se pustíme do popisu toho, co jsou to evoluční algoritmy a na jakých principech fungují, se seznámíme s tím, co jsou to „evoluční výpočetní techniky“.

6.2 Výsledky

Před tím, než se pustíme do popisu toho, co jsou to evoluční algoritmy a na jakých principech fungují, se seznámíme s tím, co jsou to „evoluční výpočetní techniky“.

7 Závěr

Před tím, než se pustíme do popisu toho, co jsou to evoluční algoritmy a na jakých principech fungují, se seznámíme s tím, co jsou to „evoluční výpočetní techniky“.

Poznámka 7.1 Následující definice a věty nedávají dohromady příliš smysl. Jsou tu jen pro ukázku.

7.1 Úvod do teorie optimalizačních algoritmů

Optimalizační algoritmy jsou mocným nástrojem pro řešení mnoha problémů inženýrské praxe. Obvykle se používají tam, kde je řešení daného problému analytickou cestou nevhodné či nereálné. Většina problémů inženýrské praxe může být definována jako optimalizační úlohy, např. nalezení optimální trajektorie robota či optimální tloušťka tlakové nádoby. Řešení takových problémů obvykle vyžaduje práci s argumenty optimalizovaných funkcí.

Algoritmy této třídy mají svůj specifický název, a to „evoluční algoritmy“. Řeší problémy tak elegantně, že se staly velmi oblíbené a používané v mnoha inženýrských oborech. Z hlediska nejobecnějšího členění patří evoluční algoritmy k algoritmům heuristickým. Heuristické algoritmy můžeme rozdělit na deterministické a stochastické. Algoritmy druhé skupiny se liší v tom, že některé jejich kroky využívají náhodné operace, a to znamená, že výsledné řešení, které jimi získáme, se v jednotlivýh běžích programu mohou lišit.

Stochastické heuristické metody poskytují pouze obecný rámec a vlastní operace algoritmu je třeba zvolit (operace křížení, mutace, ...). Protože tyto metody se často inspirují přírodními procesy, jsou také nazývány evoluční algoritmy. Podle jejich strategie je lze rozdělit do dvou tříd

7.2 Společné rysy

Evoluční algoritmy mají několik společných rysů:

1. Jednoduchost, protože tyto algoritmy lze naprogramovat obvykle velmi jednoduše.
2. Hybridnost čísel, se kterými algoritmus pracuje. Bez jakýchkoliv problémů lze kombinovat čísla typu integer, real, případně jen vybrané množiny čísel.
3. Používání dekadických čísel. Jedinec se nemusí převádět do binárního kódu, který je běžně používán u genetických algoritmů. U binárního zápisu mohou totiž mutace způsobit skokovou změnu čisla, což nemusí být dobrý dopad na průběh evoluce. Tyto nerovnoměrnosti se sice dají odstranit použitím Grayova kódování, nicméně práce s reálnými čísly je stále výhodnější.
4. Rychlost. Díky své relativní jednoduchosti, zvláště při porovnání s klasickými metodami, lze říci, že požadované řešení naleznou mnohem rychleji.

5. Chopnost nalézt extrém i u funkcí, které jsou v grafickém slova smyslu ploché a extrém je jen dírou v této rovině. Lze to označit jako hledání jehly v kupce sena. Ovšem u těchto problémů je účinnost jakýchkoliv algoritmů, včetně evolučních, velmi nízká.

6. Schopnost dát vícenásobné řešení.Výsledkem evoluce je nejlepší jedinec – jedno řešení. Pokud je ovšem v globálních extrémů více, pak lze očekávat, že budou rovněž evolučním procesem nalazeny. Jinými slovy, evoluční algoritmy jsou vhodné pro hledání extrémů funkcí trpících takovými patologiemi, jako např. šum, vsoký počet dimenzí, „multimodalita“

7.3 Optimalizace a účelová funkce

7.3.1 Vybrané pojmy z optimalizace

Úlohy vedoucí k výpočtu extrémů funkce jsou obvykle úlohy prakticé činnosti člověka a vyžadují analytické a mnohem častěji numerické výpočty extrémů funkcí více proměnných. Existují však komplikace, které ztížují optimalizaci někerých problémů, mající tyto zdroje:

- Prostor možných řešení je příliš velký. To může být například u numerativního přístupu (procházení všech možností) dosti velkým problémem
- Samotný problém je natolik složitý, že jeho matematický model použitý při optimalizaci vrací výsledky, které sice odpovídají příslušnému modelu, ale nesouhlasí se skutečně řešeným problémem. Model je vůči realitě nepřesný.
- Účelová funkce použitá k měření kvality aktuálního nalezeného řešení může podléhat šumu nebo se může měnit v čase. Proto nestačí pouze jedno řešení, ale množina řešení reprezentující vývoj toho nejlepšího.
- Množina řešení podléhá striktnímu omezení, že nalezení optimálního řešení je extrémně složitý problém.

Historický původ optimalizačních úloh sahá do antiky. Z té doby známe úlohy, jako je Didonina úloha, která měla od krále přislíbenou zemi, kterou by ohraničila volskou kůží. Didonina kůži nařezala na tenké pásky a ohraničila území, které se stalo základem Kartága.

Impulz pro rozvoj optimalizačních metod po druhé světové válce přinesly úlohy z oblasti ekonomie, které lze shrnout pod název problému optimalizace výrobních programů. Typicky se v nich předpokládá, že podnik má technické možnosti pro výrobu n druhů výrobků a hledá výrobní program x. Matematicky toto můžeme zapsat jako $h(k)$) je $\Theta(1 + \alpha)$. , kde udává počet výrobků tohoto druhu. Takový výrobní program přináší podniku užitek . O této funkci mluvíme jako o funkci účelové. Podnik však musí pro konkrétní program respektovat řadu omezení (výrobní zdroje) a rovněž omezení odbytu výrobků.

Jinou skupinou optimalizačních úloh pak může být problém úloh z dopravního hospodáství. V těchto problémech jde o to, aby byl navržen plán přepravy zboží s míst výroby do míst spotřeby. Respektují se omezení na množství přepravovaného zboží, požadavyk spotřebitelů a příp. další. Výsledkem by pak měl být plán na omezení nákladů na přepravu a nebo čas přepravy.

7.4 Evoluční algoritmy

Seznam několika evolučních algoritmů

7.4.1 Stochastic Hill Climbing

Stochastický horolezecký algoritmus je verze tzv. horolezeckého algoritmu obohaceného o stochastickou složku. Patří mezi gradientní metody, tzn. že prohledávajíprostor možných řešení ve směru největšího spádu. Díky své gradientní povaze velmi často uvázne v lokálním extrému. Vždy se vychází z náhodného bodu v prostoru možných řešení. Pro momentálně navržené řešení se pomocí konečného souboru transformací navrhne určité okolí a daná funkce se minimalizuje jen v tomto okolí. Získané lokální řešení se pak použije jako střed pro výpočet nového okolí.

7.4.2 Simulované žíhání

Je vylepšená verze horolezeckého algoritmu, u něhož vylepšení spočívá v tom, že je do horolezeckého algoritmu zavedena tzv. krátkodobá paměť, jejimž úkolem je pamatovat si ty transformace, které sloužily pro vypočítání aktuálního středu. To má v konečném důsledku ten efekt, že se algoritmus nezacyklí, díky zakázanému použití těchto transformací.

Na rozdíl od algoritmu horolezeckého, Tabu Search tak často neuvízne v lokálním extrému.

7.4.3 Evoluční strategie

Evoluční strategie patří mezi první úspěšné stochastické algoritmy v historii. Byl navržen počátkem šedesátých let Rechenbergem a Schwefelem. Vychází z principů přirozeného výběru podobně jako genetické algoritmy. Na rozdíl od jiných stochastických algoritmů pracuje sevoluční strategie přímo s reálnými hodnotami. Jejím jádrem je práce s řešením ve formě vektoru x, které je mutováno pomocí vektoru náhodných čísel.

7.4.4 Optimalizace mravenčí kolonií

Optimalizace mravenčí kolonií je algoritmus, jehož činnost napodobuje chování mravenců v kolonii. Princip je následovný: Nechť existuje zdroj mravenců (mraveniště) a cíl jejich snažení (potrava). Když jsou mravenci vypuštěni, tak po nějaké době dojde k tomu, že všichni mravenci se pohybují po kratší (optimální) cestě mezi zdrojem a cílem. Tento efekt, kdy mravenci najdou optimální cestu je dán tím, že si svou cestu značkují

feromonem. Pokud dorazí prní mravenec k rozcestí dvou cest, které vedou ke stejnému cíli, pak je jeh rozhodnutí, po které cestě se vydá, náhodné. Ti, kteří najdou potravu, začnou cestu značkovat a při návratu jsou díky těmto značkám při rozhodování ovlivnění ve prostěch této cesty. Při návratu ji označkují podruhé, což opět zvyšuje pravděpodobnost rozhodnutí dalších mravenců v její prospěch. Tyto principy jsou použity v ACO algoritmu.

Feromonová váha je aditivní, což umožňuje přidávat další feromony od dalších mravenců. V ACO algoritmu je zohledněn i fakt vypařování feromonů tak, že váhy jednotlivých spojů s časem slábnou. To zvyšuje robustnost algoritmu z hlediska nalezení globálních extrémů.

ACO byl úspěšně použitý na optimalizování problému Obchodního cestujícího nebo při návrhu telekomunikačnich sítí.

7.4.5 Imunitní systém

Metoda imunitního systému je, jak už název napovídá, algoritmus, který je založen na principech fungování imunitního systému v živých organismech. Nahlíží se na něj jako na multiagentní systém, kde jednotliví agenti mají svůj specifický úkol. Tito agenti mají různé pravomoci a schopnosti komunikovat s jinými agenty. Na základě této komunikace a určité svobody v rozhodování jednotlivých agentů vzniká hirearchická struktura schopná řešit komplikované problémy. Může například jít o použití antivirové ochrany u velkých rozsáhlých počítačových sítí.

7.4.6 Memetický algoritmus

Memetický algoritmus. Tento pojem představuje širokou třídu metaheuristických algoritmů. Klíčovou charakteristikou těchto algoritmů je použití různých aproximačních algoritmů, technik lokálního vyhledávání, speciálních rekombinačních operátorů apod. V podstatě mohou být memetické algoritmy charakterizovány jako strategie soutěže a spolupráce projevující atributy synergetiky. Jako příklad memetického algoritmu lze uvést hybridní kombinaci genetikých algoritmů a simulovaného žíhání či paralelní lokální prohledávání.

S úspěchem byly použity na řešení takových problému, jako problém obchodního cestujícího, učení neuronové sítě, plánovaní údržby, nelineární celočíselné

7.4.7 Rozptýlené prohledávání

Rozptýlené prohledávání se svou podstatou liší od standardních evolučních algoritmů a je dost podobný algoritmu Tabu Search. Je to vektorově orientovaný algoritmus, který má za úkol generovat nové vektory na základě pomocných heuristických technik. Při startu se vychází z řešení získaných pomocí vhodné heuristiké techniky. Poté jsou generována nová řešení na základě podmnožiny nejlepších řešení ze startu. Z těchto nově nalezených řešení se opět vybere množina těch nejlepších a celý proces se opakuje. Tento algoritmus

byl použit k řešení problémů, jako je řízení dopravy, učení neuronové sítě, optimalizace bez omezení, atd.

7.4.8 Rojení částic

Rojení častic je založeno na práci s populací jedinců, jejichž pozice v prostoru možných řešení je měněna pomocí tzv. rychlostního vektoru. Podle popisu nedochází v základní verzi mezi jedinci k vzájemnému ovlivňování. To je odstraněno ve verzi s tzv. sousedstvím. V rámci tohoto sousedství dochází k vzájemnému ovlivňování tak, že jedinci patřící do jednoho sousedství putují k nejhlubšímu extrému, který byl v tomto sousedství nalezen.

7.5 Optimalizační problémy

Zde budou popsány některé optimalizační problémy

7.5.1 Problém obchodního cestujícího

Problém obchodního cestujícího (TSP) je úloha z množiny NP-úplných problémů.

7.5.2 Problém sudoku

Jedná se o problém nalezení řešení k danému schámatu sudoku mřížky, pokud máme k dispozici pouze několik vyplněných čísel.

8 Genetické programování

Název "genetické programování"se zrodil již počátkem 90. let, kdy byl představen J. Kozou, jako algoritmus pro využití v problémeh jako je predikce, klasifikace, aproximace, tvorba programů. Vlastnostmi se podobá neuronovými sítěmi. Od jiných evolučních algoritmů vyčnívá svou značně velkou populací (čítající tisíce jedinců) a z toho důvodu se jeví, jako značně pomalý algoritmus. Výhodou oproti jiným algoritmům (např. genetický algoritmus), které mají většinou lineárlní strukturu, zde tvoři jedince ne-lineární chromosomy (např. stromy či grafy).

Pro to, abychom pochopili smysl tohoto algoritmu, položmě si tuto otázku: "Jak se může počítač naučit řešit problémy bez toho, abychom jej k tomu přímo nenaprogramovali? Jinými slovy, jak může počítač sám dělat to co potřebujeme aby dělal, bez toho abychom mu to přesně řekli?"Touto otázkou Arthur Samuel již v roce 1959 otevřel debatu kolem umělé inteligence.

Typickým příkladem pro genetické programování je model, který se snaží nalézt logický výraz, jehož výsledkem je buď ano či ne. Můžeme si to představit na modelu banky, která eviduje své zákazníky a jejich úvěry. U každého záznamu zákazníka známe cílový stav (dostal úvěr nebo nedostal)? Nás potom ude zajímat, podle jakých kritérií bychom mohli tento model kategorizovat (čili vytvořit podmínky pro rozhodnutí, kdo úvěr dostane a kdo ne).

Reprezentace možného modelu se opírá o stromovou struktura, která je univerzální pro popis složitější (komplexnější) funkce.

8.1 Fitness

Každého vytvořeného reprezentativního jedince je třeba ohodnotit v rámci celé populace čítající desídky jiných jedinců tak, aby bylo možné některé jedince (řešení) upřednostňovat při výběru pro křížení před jinými (horšími) řešeními. Hodnota takto přidělená jedinci je označována za jeho fitness (neboli vhodnost).

8.2 Selekce

Další funkcí, kterou genetické programování uplatňuje na populaci je selekce jedinců (řešeních), které figurují na vstupu fáze křížení. Algoritmus selekce nahlíží na hodnoty fitness jedinců a náhodně vybírá jedince z populace tak, aby byly vybírání jedinci s vyšším fitness s větší pravděpodobnosti, než ti s nižší fitness. Metod, jak výběr provést je hned několik. Jedná se například o výběr ruletovým kolem či jinou metodou.

8.3 Křížení

Největší podíl na tvorbě nových jedinců má metoda křížení rodičů. Stejně jako v biologických organismech se křížením označuje proces předávání genetického materiálu obou rodičů svým potomkům, jde i v tomto případě o vytváření jedince, skládajícího se z části svých rodičů.

8.4 Mutace

Operací mutace se rozumí proces, při němž se malinko pozmění část potomka tak, že jej nelze zoětně charakterizovat jako část prvního rodiče a část druhého rodiče, ale malá část je náhodně pozměněna. Důvodem tohoto snažení je to, aby jedinci v nové generaci nekonvergovali příliš brzo a to pro generování stále stejných řešení.

9 Praktická část

Tohle je sekce zabývající se implemntaci aplikace

9.1 Existující aplikace

10 Testování a výsledky

Tohle je sekce zabývající se implemntaci aplikace

[4] Kvasnička V., Pospíchal J., Tiňo P., Evolučné algoritmy, STU Bratislava, ISBN 85-246-2000, 2000 [5] Zelinka I.: Analytic Programming by Means of Soma Algorithm. ICICIS'02, First International Conference on Intelligent Computing and Information Systems, Egypt, Cairo, 2002 [6] Zelinka Ivan, Evoluční výpočetní techniky - principy a alikace, BEN, Praha, 2008

11 Závěr

Tak doufám, že Vám tato ukázka k něčemu byla. Další informace najdete v publikacích [3, 4].

Zdeněk Gold

12 Reference

[1] Koza, J. R., *Genetic programming. On the Programming of Computers by Means of Natural Selection*, Cambridge, MA: MIT Press, 1992.

[2] Koza, J. R., *Genetic programming II. Automatic Discovery of Reusable Programs*, Cambridge, MA: MIT Press, 1994.

[3] Goossens, Michel, *The LaTeX companion*, New York: Addison, 1994.

[4] Lamport, Leslie, *A document preparation system: user's guide and reference manual*, New York: Addison-Wesley Pub. Co., 2015.

[5] Koza J.R, *Genetic Programming*, MIT Press, ISBN 0-262-11189-6, 1998

[6] Koza J.R, Bennet F.H., Andre D., Keane M., *Genetic Programming III*, Morgan Kaufnamm pub., ISBN 1-55860-543-6, 1999

[7] Lampinen Jouni, Zelinka, Ivan, *New Ideas in Optimization and Mechanical Engineering Design Optimization by Differential Evolution. Volume 1*, London: McGraw-Hill, 1999. 20 p. ISBN 007-709506-5

Definice 12.1 *Binární strom je struktura definovaná nad konečnou množinou uzlů, která:*

- *neobsahuje žádný uzel,*
- *je složena ze tří disjunktních množin uzlů: kořene, binárního stromu zvaného levý podstrom a binárního stromu tzv. pravého podstromu.*

Pak by se taky mohla hodit nějaká věta a k ní důkaz.

Věta 12.1 *Průměrná časová složitost neúspěšného vyhledání v hashovací tabulce se separátním zřetězením je $\Theta(1+\alpha)$, za předpokladu jednoduchého uniformního hashování.*

Důkaz. Za předpokladu jednoduchého uniformního hashování se každý klíč k hashuje se stejnou pravděpodobností do libovolného z m slotů tabulky. Průměrný čas neúspěšného hledání klíče k je proto průměrný čas prohledání jednoho z m seznamů. Průměrná délka každého takového seznamu je rovna faktoru naplnění $\alpha = n/m$. Tudíž lze očekávat, že budeme nuceni prozkoumat α prvků. Z toho plyne, že celkový čas pro neúspěšné hledání (plus navíc konstantní čas pro výpočet $h(k)$) je $\Theta(1+\alpha)$. ∎

Příklad 12.1
Mějme napsat funkci, která spočítá uzly ve stromu. Předpokládejme, že binární strom je definován způsobem uvedeným v definici 12.1 na straně 28. Naše úloha se výrazně zjednoduší uvědomíme-li si její rekurzivní charakter a předpokládáme, že aktuální uzel je R.

- Je-li R prázdný strom (tj. $R = NULL$), pak počet jeho uzlů je pochopitelně nula. Tím máme problém vyřešen.

- V opačném případě víme, že ve stromu určitě jeden uzel existuje (R) a počty uzlů v levém a pravém podstromu se dají určit obdobným způsobem rekurzivně. To znamená, že počet uzlů ve stromu s kořenem R je $1+\mathrm{pocet\_uzlu}(A)+\mathrm{pocet\_uzlu}(B)$

Počty uzlů pro jednotlivé podstromy se předávají jako výsledky volání funkcí prostřednictvím zásobníku programu, nejsou tudíž potřeba žádné pomocné proměnné. ■

Poznámka 12.1 Program z příkladu 12.1 pochopitelně chybí, ale můžete se podívat třeba na program uvedený ve výpisu 1.

12.1 Výpisy programů

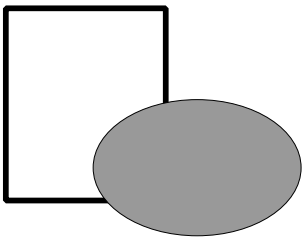
Tato diplomová práce má nastaven výchozí jazyk Java, jak je vidět z výpisu 1. Výpis kódu 1 zároveň demonstruje možnost přímého vložení zdrojového kódu programu do textu práce. Druhou možností je načtení zdrojového kódu programu z externího souboru, viz výpis 2. Pokud potřebujeme změnit programovací jazyk pro konkrétní výpis kódu, můžeme jeho to provést přímo v záhlaví prostředí `lstlisting`. Výpis 3 je v jazyku Pascal. Všimněte si zvýraznění klíčových slov.

Poznámka 12.2 Pro správnou sazbu je třeba pro odsazování používat tabulátory, nikoliv mezery.

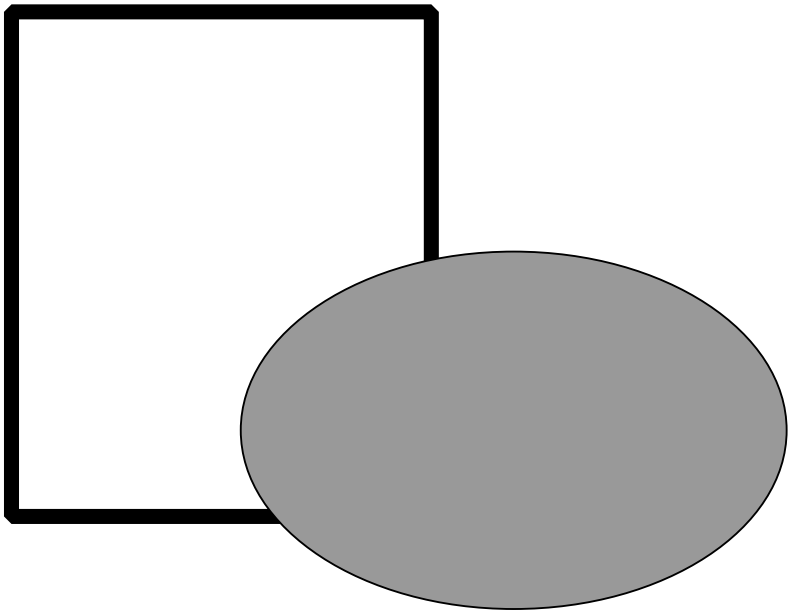
```
public class MyClass
{
    public int MyMethod(int a, int b)
    {
        while (a != b)
        {
            if (a < b)
                b -= a;
            else
                a -= b;
        }
    }
}
```

Výpis 1: Program v jazyce Java

```
public class MyClass
{
    public int MyMethod(int a, int b)
    {
        while (a != b)
        {
            if (a < b)
                b -= a;
            else
                a -= b;
        }
    }
```



Obrázek 3: Pokusný obrázek – absolutní velikost



Obrázek 4: Pokusný obrázek – relativní velikost

```
}
```

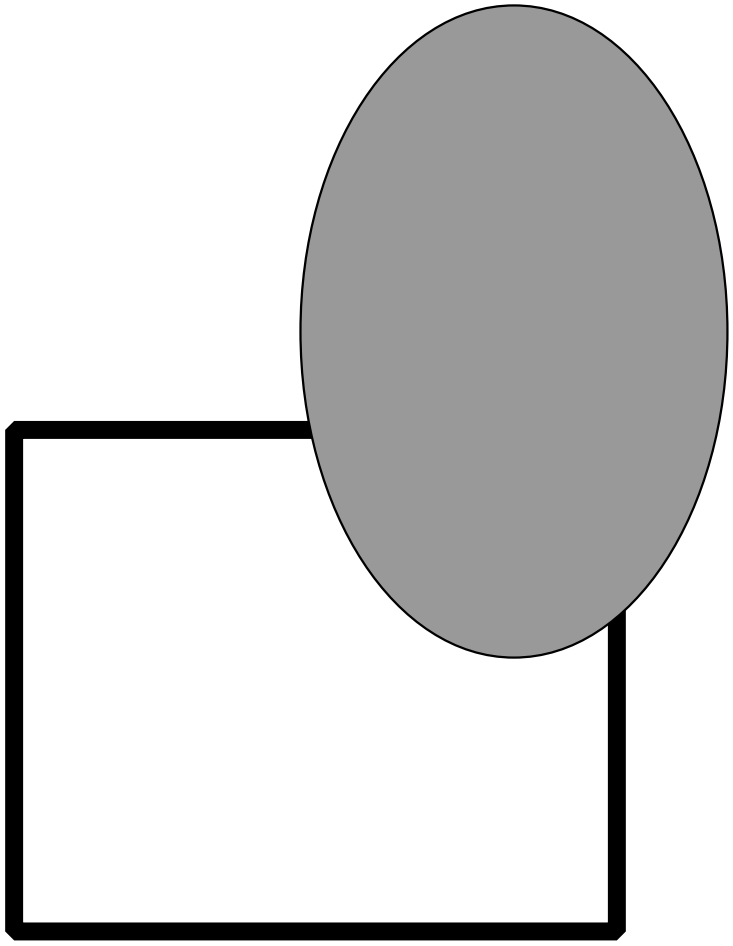
Výpis 2: Program v jazyce Java, načtený z externího souboru

```
procedure X(i : integer; var x : real);
begin
   x := i + 3;
end;
```

Výpis 3: Program v Pascalu

12.2 Obrázky a tabulky

A ještě si můžeme zkusit vysázet obrázek. Obrázek 3 má určenu absolutní velikost, zatímco obrázek 4 je určen relativně vůči šířce textu.



Obrázek 5: Pokusný obrázek – otočený naležato

q	$\delta(q,0)$	$\delta(q,1)$
q_0	q_1	q_0
q_1	q_1	q_2
q_2	q_1	q_0

Tabulka 1: Pokusná tabulka

A ještě zkusíme vysázet několik tabulek, ale jen kvůli seznamu tabulek v úvodu. Tabulka 1 představuje jednoduchou tabulku, která se svou šířkou pohodlně vejde do šířky textu. Velké tabulky, stejně jako obrázky, můžeme vysázet naležato. Ukázkou velké, komplikované tabulky[1] je tabulka 2.

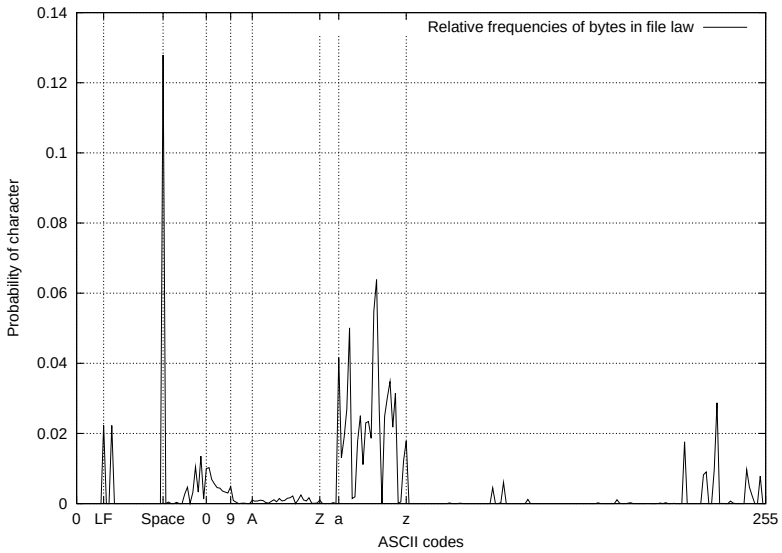
[1] Pokud, ale píšete práci česky, měly by být tabulky také česky – mě se jen nechtěla předělávat do češtiny.

File	bible.txt		world.txt		law.txt		latimes.txt	
Language	English		English		Czech		English	
Format	Plain text		Plain text		Plain text		SGML	
Size of file [bytes]	4047392		2473400		64573143		498360166	
Number of tokens	1532262	100%	684767	100%	19432898	100%	161254928	100%
Number of words	766131	50%	342383	50%	9716449	50%	70766067	43.885%
Number of nonwords	766131	50%	342384	50%	9716449	50%	80619289	49.995%
Number of controls							9869572	6.12%
Number of unique tokens	13791	100%	23564	100%	250570	100%	529482	100%
Number of unique words	13744	99.659%	23082	97.955%	246266	98.282%	524280	99.018%
Number of unique nonwords	47	0.341%	482	2.045%	4304	1.718%	3079	0.582%
Number of unique controls							2123	0.401%
Word average frequency	55.743		14.833		39.455		134.978	
Nonword average frequency	16300.66		710.34		2257.539		26183.595	
Control average frequency							4648.88	
Minimal length of word	1		1		1		1	
Maximal length of word	18		27		41		58	
Minimal length of nonword	1		1		1		1	
Maximal length of nonword	4		56		700		253	
Minimal length of control							3	
Maximal length of control							132	

Tabulka 2: Experimental Files — Detailed Statistics

A Grafy a měření

Tohle je příloha k práci. Většinou se sem dávají grafy, tabulky, které by vzhledem ke svému počtu překážely v textu diplomky.



Obrázek 6: Nějaký graf