Univerzita Hradec Králové
Fakulta informatiky a managementu

katedra informatiky a kvantitativních metod

Evoluční algoritmy v biomedicínském výzkumu

DIPLOMOVÁ PRÁCE

Autor: Lukáš Rezner

Studijní obor: Aplikovaná informatika

Vedoucí práce: Ing. Karel Mls, Ph.D.

Hradec Králové září 2016

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a uvedl jsem všechny použité prameny a literaturu.

V Hradci Králové dne 18. srpna 2016 Lukáš Rezner

Poděkování

Rád bych zde poděkoval vedoucímu diplomové práce panu Ing. Karlovi Mlsovi, Ph.D. za odborné vedení práce, podnětné rady a čas, který mi věnoval. Mé poděkováni patři též Ing. Agátě Milanov, Ph.D. za zprostředkovaní spolupráce na praktickém projektu v oblasti biomedicíny.

Anotace

Diplomová práce se zabývá problematikou hledání řešení rozsáhlých problémů, kde standardní matematicko-statistické metody selhávají, pomocí evolučních přístupů. Konkrétním cílem práce je navrhnout a implementovat knihovnu evolučních algoritmů. Praktickým problémem, na který byla navržená knihovna aplikována, byla data získaná při vývoji nových léčiv v rámci projektu biomedicínského výzkumu na FIM.

V teoretické části je čtenář nejprve seznámen s pojmem optimalizační problém. Následuje samotná problematika evolučních algoritmů, její historie a základní pojmy.

Praktická část se následně soustředí na samotnou implementaci knihovny evolučních algoritmů. Zahrnuje popis jednotlivých tříd knihovny, doporučení, jakým způsobem konfigurovat knihovnu, a v neposlední řadě i výsledky testování funkčnosti knihovny.

Klíčová slova: evoluce, optimalizace, evoluční algoritmus, biomedicína, biomedicínský výzkum, knihovna, Java

Annotation

The diploma thesis deals with problematics of finding the solution of large problems where standard mathematical-statistics methods fails, with the use of evolutionary approach. The goal of this thesis is to design and implement the library of evolutionary algorithms. Library was designed for data which were delivered during the development of new medicines in the project of biomedical research at FIM.

In the theoretical part of thesis a reader will be informed about the concept of optimization problem. Followed by the problems of evolutionary algorithms, its history and basic concepts.

The practical part of thesis is focused on the implementation of library of evolutionary algorithms. It includes a description of each library class and recommendations on how to configure the library. Finally the thesis contains results of testing of the library.

Key words: evolution, optimalization, evolutionary algorithm, biomedicine, biomedical research, library, Java

Obsah

1 Úvod **1**
- 1.1 Cíl a metodika práce . . . 1
- 1.2 Struktura práce . . . 2

2 Teoretická část **3**
- 2.1 Optimalizační algoritmy . . . 3
  - 2.1.1 Dělení optimalizačních algoritmů . . . 3
  - 2.1.2 Využití heuristiky v optimalizaci . . . 5
- 2.2 Evoluce . . . 6
- 2.3 Biologická Evoluce . . . 6
  - 2.3.1 Darwinismus . . . 6
- 2.4 Evoluce jako algoritmus . . . 7
  - 2.4.1 Historie . . . 7
  - 2.4.2 Základní principy EA . . . 8
  - 2.4.3 Společné rysy evolučních algoritmů . . . 11
  - 2.4.4 Základní pojmy . . . 12
  - 2.4.5 Popis obecného evolučního algoritmu . . . 15
  - 2.4.6 Operátory . . . 16
  - 2.4.7 Selekce . . . 18

3 Praktická část **23**
- 3.1 Použité technologie . . . 23
- 3.2 Popis knihovny . . . 23
  - 3.2.1 Struktura knihovny . . . 24
  - 3.2.2 Popis důležitých tříd . . . 26
  - 3.2.3 Vlastní implementace služeb - crossover . . . 32
  - 3.2.4 Vlastní implementace služeb - selection . . . 35
  - 3.2.5 Vlastní implementace služeb - fitness funkce . . . 39
- 3.3 Hlavní proces . . . 39
  - 3.3.1 Rozhraní Evoluce . . . 39
  - 3.3.2 Implementace rozhraní evoluce . . . 40
- 3.4 Hlavní spouštěcí třída . . . 45
- 3.5 Konfigurace . . . 47
- 3.6 Testování knihovny . . . 49

4 Shrnutí výsledků 51

5 Závěry a doporučení 52

Seznam obrázků 53

Seznam grafů 53

Seznam tabulek 54

Seznam ukázek kódu 55

Literatura 58

Přílohy I

A Výsledky testů I

B Obsah DVD VIII

B.1 Spustitelný skript VIII

1. Úvod

Problémů, které nelze vyřešit deterministickou metodou a určit tak jedinečný výsledek je celá řada. Setkáváme se s nimi každodenně, jak v oblastech vědeckého, profesního, ale i všedního života. Chceme se dobrat výsledku, který se alespoň částečně přibližuje ideálu. Hledáme tedy nejlepší možné řešení. Problémy tohoto druhu se nazývají optimalizační.

Evoluční algoritmy jsou jednou z oblastí, která tuto problematiku řeší velmi dobře. Jejich výhoda spočívá v řešení problémů, které jsou charakteristické vysokou dimenzí.

Evoluční algoritmy se inspirují v přírodě, kde fungují procesy ověřené miliony let. Tyto ověřené procesy jsou využívány k vyřešení optimalizačních problémů.

Diplomová práce se bude zabývat zajímavým spojením evolučních algoritmů s biomedicínským výzkumem, kde se vyskytují optimalizační problémy charakteristické právě svou vysokou dimenzí.

1.1. Cíl a metodika práce

Cílem diplomové práce je implementovat knihovnu evolučních algoritmů v jazyce Java. Knihovna by měla být implementována na míru tak, aby usnadnila biomedicínský výzkum.

Výsledná knihovna evolučních algoritmů, jejíž implementaci má za cíl tato diplomová práce, bude použita v projektu „SPEV - Využití metod umělé inteligence v bioinformatice“. Tento projekt je realizován Univerzitou Hradec Králové a pomáhá řešit optimalizační problém biomedicínského výzkumu společně s neuronovou sítí. V tomto případě bude neuronová síť představovat fitness funkci evolučního algoritmu. Tím bude dosažena zajímavá kombinace dvou metod pro řešení optimalizačních problémů.

Pro vytvoření diplomové práce byla využita „metodika pro vypracování bakalářských a diplomových prací“.

V praktické části diplomové práce (implementace knihovny evolučních algoritmů) byla použita „metodika vývoje software - spirálový přístup“. Tato metodika je založena na systematickém opakování následujících kroků: analýza, hodnocení, vývoj a plánování.

1.2. Struktura práce

Tato diplomová práce je rozdělena na dvě hlavní části, teoretickou část a praktickou část. V teoretické části práce bude vysvětlen pojem optimalizační algoritmus. Dále budou objasněny pojmy jako evoluce a biologická evoluce. Nakonec bude popsáno, jak se dá evoluce využívat v informatice.

V praktické části bude podrobně popsána knihovna evolučních algoritmů. Popíšeme implementování jednotlivých metod křížení a selekce, konfigurace knihovny a v poslední řadě celý hlavní proces algoritmu.

2. Teoretická část

V této kapitole práce bude představena evoluce v přírodě a budou diskutovány možnosti jejího využití v počítačovém světě.

2.1. Optimalizační algoritmy

Optimalizační algoritmy slouží k nalezení vhodného (optimálního) řešení problému zejména v případech, kdy není znám matematický popis řešení problému. Tyto metody jsou v principu velmi jednoduché, jsou založené na různých způsobech prohledávání prostoru řešení a prakticky je nelze využít bez počítače.

Jedním ze společných prvků optimalizačních algoritmů je nutná znalost prostoru, resp. jeho ohraničení. Je potřeba znát, kde je přípustné optimální řešení. Dalším společným znakem těchto algoritmů je nutná znalost hodnotící funkce, tzv. fitness funkce. Obecně lze tuto funkci popsat:

$$f = f(x_1, x_2, x_3, ..., x_n),$$

kde $x_1, x_2, ..., x_n$ jsou optimalizované neznámé hledaného problému.

Velká část těchto algoritmů je více či méně postavena na základech (principech) procesů, které jsou využívány v přírodě, kde je ověřena správná funkce. Do této oblasti optimalizačních algoritmů patří také evoluční algoritmy. [1]

2.1.1. Dělení optimalizačních algoritmů

Optimalizační algoritmy lze dělit podle mnoha různých kritérií. Například podle principu činnosti, složitosti, kterou disponují atp. Základní rozdělení podle chovaní lze dělit takto:

- Enumerativní
- Deterministické
- Stochastické

- Smíšené

Enumerativní

Tyto algoritmy fungují na principu výpočtu všech možných kombinací daného problému. Tento typ algoritmů se hodí v případě, kdy vstupní hodnoty argumentů nabývají pouze diskrétního charakteru a nabývají malého počtu hodnot. Kdyby byl tento typ algoritmu použit na reálné vyřešení nějakého problému, pak by pravděpodobně potřeboval nekonečné množství času pro úspěšné ukončení. V praxi je tedy tento algoritmus nepoužitelný.

Deterministické

Tento typ optimalizačních algoritmů je postaven pouze na metodách klasické matematiky. Specifické algoritmy tohoto typu obvykle potřebují specifikovat omezující předpoklady, pomocí kterých pak algoritmus umožní podávat efektivní výsledky. Mezi ony požadavky patří například:

- Problém je lineární
- Problém je konvexní
- Prohledávaný prostor je malý
- Prohledávaný prostor je souvislý
- Problém je definován v analytickém tvaru

Algoritmus tohoto typu má jediné správné řešení.

Stochastické

Tento typ optimalizačních algoritmů je postaven na náhodném prohledávání. Principem je v podstatě čistě náhodné prohledávání hodnot prozkoumávaného prostoru. Výsledkem je vždy nejlepší řešení, které bylo nalezeno za celý proces tohoto algoritmu.

Pro tyto algoritmy platí, že:

- Bývají pomalé
- Vhodné použít tam, kde je malý prostor možných řešení
- Vhodné pro hrubý odhad

Smíšené

Smíšené optimalizační algoritmy, jak název napovídá, slučují výhody deterministických a stochastických algoritmů. Překvapivě tato kombinace dosahuje dobrých výsledků. Do této kategorie patří i celá skupina evolučních algoritmů.

Mezi znaky, pomocí kterých lze tento typ algoritmů charakterizovat, patří:

- Robustnost. Znamená to, že nezávisle na počátečních podmínkách většinou dokáží velmi rychle nalézt kvalitní řešení.
- Výkonné a efektivní. Dokáží nalézt dobré řešení za malého počtu opakování ohodnocení účelové funkce.
- Mají minimální požadavky na předběžné informace.
- Nepotřebují ke své činnosti analytický popis problému.
- Zvládnou během jednoho spuštění nalézt více globálních extrémů.

[2]

2.1.2. Využití heuristiky v optimalizaci

Skutečnost, že v některých optimalizačních problémech není možné nalézt pro správné řešení deterministický algoritmus, vede k využití algoritmů stochastických, jak bylo popsáno výše. Stochastické algoritmy sice negarantují nalezení výsledku v konečném počtu kroků, ale dokáží v přijatelném čase nalézt řešení alespoň sub-optimální.

Stochastické algoritmy využívají heuristiky k prohledávání prostoru možných řešení. Slovo heuristika je odvozeno z řeckého slova *heuriskein*, které v překladu znamená hledat a objevovat. Rozumíme tím tedy postup, ve kterém se využívá náhoda, intuice, analogie a zkušenost. Rozdíl mezi heuristikou a deterministickým přístupem spočívá v tom, že na rozdíl od deterministického přístupu heuristika nezaručuje, že nalezne řešení problému.

Heuristiky jsou běžnou součástí reálného života. Jako heuristiku lze definovat například hledání hub v lese, pokus o výhru ve sportovním utkání nebo o složení zkoušky ve škole. Inspirace využití heuristik v algoritmech je často odvozena ze znalosti přírodních a sociálních procesů.

Stochastické algoritmy evolučního typu jsou v poslední době oblíbené zejména pro řešení optimalizačních problémů. Heuristikou se dají tedy považovat i evoluční algoritmy, které určitým způsobem modifikují populaci tak, aby se její vlastnosti zlepšovaly. V některých případech evolučních algoritmů se dokonce setkáváme s případem, že řešení se blíží globálnímu optimu. [3]

2.2. Evoluce

Termín evoluce je poměrně obecný pojem. V obecném chápání je evoluce změna, nicméně se nejedná pouze o změnu organizmů v čase. Předmět této změny popisuje například DouglasJ. Futuyma: „V širším slova smyslu, evoluce je pouze změna, a proto je všudypřítomná; galaxie, jazyky, politické systémy, to všechno se vyvíjí". Takto definovaný pojem je hodně obecný a nedává velký smysl. Je tedy zřejmé, že musí být lépe specifikován.

Změny u živých organismů, které podléhají klasické evoluční teorii, je vhodnější zařadit pod specifický pojem *biologická evoluce*, která je jednou z mnoha podmnožin evoluce. Práce se dále bude zabývat hlavně touto biologickou evolucí. Nicméně pro ujasnění pojmu je toto rozdělení potřeba zmínit, že existují i další druhy evolucí.[4]

2.3. Biologická Evoluce

V této kapitole bude popsán termín „biologická evoluce". Zaměříme se na jeho definici.

Jako ve spoustě dalších odvětví například v robotice, se i zde informatika inspiruje v přírodě a v přírodních vědách. Tisícem let je prověřeno, že daný proces funguje.[5]

Dále se práce zaměří na pojem „evoluce" (přesněji biologická evoluce). Znamená samovolný, dlouhodobě probíhající proces, ve kterém se v průběhu rozvíjí pozemský život. Teorii, která spojuje myšlenky postupné evoluce druhu s přirozeným výběrem, představil Charles Darwin. Kromě biologické evoluce se velmi dobře uplatňuje evoluce kulturní.

Teorii, která spojuje myšlenky postupné evoluce druhu s přirozeným výběrem, představil Charles Darwin.

Stručně řečeno evolucí živočišných druhů se rozumí postupný vývoj života od zcela prvního výskytu na Zemi k vývoji různých forem života. Společné rysy ve struktuře genetického kódu všech dosud žijících živočichů včetně lidi dokazuje, že evoluce, jak je popsána, funguje a existuje společný prvotní původ. Ukazuje se, že společné sady genů jsou selektovány deterministickým vlivem prostředí a podobnost organismů odráží prostředí, ve kterém se vyvíjely.[5]

2.3.1. Darwinismus

Darwinismus představuje evoluční teorii vytvořenou Charlesem Darwinem a Alfredem Wallacem. Princip této teorie spočívá v tom, že evoluce probíhá za pomocí drobných nepatrných změn na základě selekce vycházející z úspěšnosti rozmnožování jedince. Vychází z toho, že každý organismus je originál a neexistují dva shodné. Dále, že mohou dědit vlastnosti od vlastních rodičů a že je vyšší pravděpodobnost, že se

bude více množit organismus, který má více vhodných vlastnosti, které předá svým potomkům.

Darwinismus se stal první evoluční teorií, která se obecně dostala do podvědomí vědců, zároveň byla poslední ucelenou teorií. Postupem času, jak se znalosti v rámci těchto oborů rozšiřovaly, se začalo ukazovat, že takto pojatá základní myšlenka naprosto nedostačuje pro vědeckou práci, proto byl postupně darwinismus modifikován a obohacován o další poznatky, postuláty a vytvářely se snahy o syntézu s novým vědeckým poznáním. Této nově vznikající syntéze se říká neodarwinismus. [5]

2.4. Evoluce jako algoritmus

Tato kapitola se bude zabývat „evolučním algoritmem". Evoluční algoritmy jsou jedna z mnoha snah využít principy, mechanismy a hlavně vzory, které jsou pozorovatelné v přírodě.

2.4.1. Historie

Prvopočátky využívání přírodních principů evoluce k řešení optimalizačních úloh sahají do období 50. až 60. let dvacátého století. V této době se vyvinula oblast informatiky, kterou souhrnně nazýváme *evoluční algoritmy*. Tuto oblast využívající principů evoluce dále můžeme dělit na čtyři základní směry:

1. Evoluční strategie
2. Genetické programování
3. Evoluční programování
4. Genetické algoritmy

Evoluční strategie

Primárně slouží k optimalizaci reálných parametrů průmyslových zařízení. Prvně byla představena dílem Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution, publikované v roce 1973. [6]

Evoluční strategie využívá hlavně operátory kódování pomocí reálných čísel. Dále mutace, křížení a selekci.

Genetické programování

Genetické programování je zaměřeno na využívání metod biologické evoluce, při vytváření programů určených pro řešení určité úlohy. Řešením je počítačový program a

míra úspěšnosti tohoto řešení je dána fitness funkcí. Za tvůrce této metodologie je považován Joh R. Koza (kniha Genetic programming: on the programming of computers by means of natural selection). [7]

Evoluční programování

Evoluční programování spočívá v tom, že se aplikují náhodné mutace na konečné automaty, které reprezentují finální řešení . Tato technika pochází z roku 1966, kdy ji ve svém díle Artificial Intelligence through Simulated Evolution prezentovali autoři L.J. Fogel, A. J.Owens a M. J.Walsch. [8]

Genetické algoritmy

Genetické algoritmy jsou v dnešní době asi nejvíce užívanou technikou. Prvně byly představeny v publikaci Adaptation in Natural and Artificial Systems, kterou v roce 1975 publikoval John H. Holland. Kromě samotného popisu algoritmu se Holland věnoval také pokusům o teoretické zdůvodnění jeho funkčnosti. [9]

To, jak chápeme genetické algoritmy dnes z velké části staví na základu Hollandova principu. Do této kategorie můžeme přiřadit také genetické programování.

Díky nástupu počítačů byl získán silný nástroj, se kterým se snažíme napodobovat postupy ověřené přírodou. Dříve než evoluční algoritmy byla rozšířena oblast neuronových sítí a strojového učení až později přišla řada i na evoluční algoritmy, jejíž příkladem jsou genetické algoritmy.

Samotný termín genetické algoritmy (GA) vznikl v roce 1975, kdy na popud Johna Hollanda byla označena množina algoritmů, které měly určité shodné rysy. Genetické algoritmy byly zpočátku využívány k simulaci biologické evoluce. Později však pronikly do optimalizačních úloh, kde se ukázaly jako velmi silné. Na Hollandových základech stavěli jeho následovníci. Konec 80. a začátek 90. let však tato oblast zaznamenala mírný úpadek. Nyní se opět těmto algoritmům dostává velká pozornost. [10]

2.4.2. Základní principy EA

Skutečnosti, že dnes můžeme využívat evoluční algoritmy, vděčíme přírodovědci Charlesu Darwinovi. Před více než 150 lety vydal knihu, která se jmenuje *O původu druhů*, kde poprvé uveřejnil svou evoluční teorii, založenou na přírodním výběru. [11]

Evoluční algoritmy využívají heuristiky (jak bylo zmíněno v kapitole 2.1.2) a společně s nimi i přírodní principy, mezi které patří například výběr, křížení a mutace. [12]

Nejvhodnější využití evolučních algoritmů je pro řešení velkých komplexních optimalizačních problémů s velkou šancí uvíznutí v lokálním optimu. Proto se hodí použití evolučních algoritmů, které mají menší pravděpodobnost uvíznutí v lokálním minimu než tradiční gradientní metody. Evoluční algoritmy jsou mnohem robustnější než jiné prohledávací algoritmy.

Evoluční algoritmy jsou ve své podstatě speciálním případem prohledávajících algoritmů, které obecně slouží k vyhledání problému pomocí vyhledání tohoto řešení. Základ této skupiny algoritmů je pohyb v prostoru prohledáváním, kde konkrétně u evolučních algoritmů představuje každý jeden bod tohoto prostoru jedno ucelené řešení (kandidát řešení) a na základě vstupních parametrů a kritérií na výsledek se teprve rozhodne, zda konečné. Tento prostor bývá někdy označován jako prostor kandidátů.

Samotné prohledávání prostoru se rozumí generování nových kandidátu (generací) a následné testování jedinců. Při řešení konkrétních úloh je třeba generovat obrovské množství těchto jedinců k testu, dokud se nenajde optimální řešení.

Obecně prohledávací algoritmy mají málo požadavků na řešení úlohy, proto díky nim lze řešit velké množství problémů. Úloha, aby mohla být řešena pomocí prohledávajícího algoritmu, vyžaduje pouze to, aby řešení splňovalo dvě podmínky. Podle nich musí být řešení:

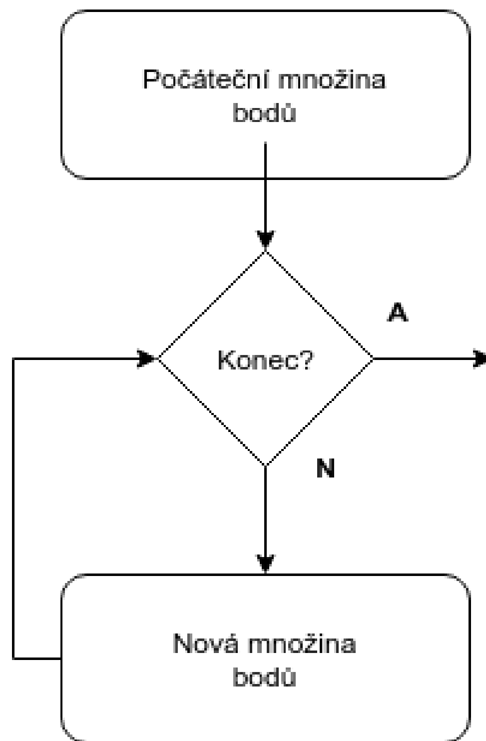
- rozložitelné
- hodnotitelné

U hledaného řešení se předpokládá, že bude mít tvar rozložený na jednotlivé složky. Takové řešení, které je rozdělené na složky, muže být vyjádřeno jako soubor atributů s jejich hodnotami. Výhoda je, že se může jednat o atributy různých typů (binární, nominální, reálné, atp.).

Každý bod v hledaném prostoru představuje jedno možné řešení (to je zmíněno výše), ale vhodných řešení je jen omezené množství. Proto je třeba určit tzv. míru vhodnosti, která představuje kvalitu daného kandidáta na řešení. Pokud každý z kandidátů bude mít tento údaj určující jeho kvalitu, lze jednotlivé kandidáty mezi sebou porovnávat a získat tak nejlepšího kandidáta (optimální řešení). Platí, že čím je míra vhodnosti vyšší, tím je daná kombinace hodnot atributů lepším řešením. Prohledávací, konkrétně evoluční, algoritmus hledá vlastně maximální míru vhodnosti a tu potom označí jako optimální řešení daného problému.

Základní struktura prohledávacího algoritmu je znázorněna na obrázku 1

Algoritmus začíná prohledáváním množiny kandidátu na řešení konkrétní úlohy. Jestliže se mezi nimi nenachází kandidát, který by splňoval podmínky na řešení úlohy, pak pokračuje k výběru jiné množiny kandidátů. Následně se celý proces opakuje, do-



Obrázek 1.: Struktura prohledávacího algoritmu.

kud se v množině kandidátu na řešení úlohy nenachází, který by splňoval podmínku, která by vedla k optimálnímu řešení úlohy.

Pod pojmem evoluční algoritmus se skrývá poměrně široká třída algoritmů. Jak bylo řečeno výše, jsou speciálním případem prohledávacích algoritmů. Také se dají definovat jako populační algoritmy. To znamená, že pracují s populací a využívají heuristiky, které dokáží modifikovat populaci takovým způsobem, aby se její vlastnosti zlepšovaly. Při volbě vhodné konfigurace evolučního algoritmu je dokázáno, že nejlepší jedinci populace se skutečně přibližují ke globálnímu optimu.

Výběr nových jedinců do populace má stochastický charakter, ale hledání už náhodné není. Pravděpodobnost konkrétního výběru je ovlivněna. Jedná se o kombinaci náhodného prohledávání prostoru a využívaní již známých informací o prostoru. To nám umožňuje ovlivňovat výběr do oblasti, kde je to pro nalezení optimálního řešení výhodné.

Je však nutné, aby tato kombinace byla rovnoměrná a vždy byla zachována určitá náhoda při výběru. Jinak by se mohlo stát, že se hledání bude soustředit pouze na určitou oblast. Kdyby se tak dělo, mohlo by to mít následek takový, že by se ztratila variabilita jedinců. To znamená, že jedinci v jedné populaci by byli navzájem podobní. Při vytváření nové populace by byli jedinci vybíráni z jedné oblasti, což by mohlo vést k závěru,

že algoritmus skončí v lokálním optimu a výsledek tak nebude správný. Na druhou stranu, pokud je naopak nastavena přílišná náhoda při výběru, čas k ukončení algoritmu by mohl jít k nekonečnu. Návrh (konfigurace) evolučního algoritmu je vlastně nalezení kompromisu mezi snahou vybírat z určité oblasti a zachováním náhody při výběru. [13]

Evoluční algoritmy se používají jak pro optimalizaci diskrétních, tak i pro optimalizaci reálných proměnných. Je o nich známo, že fungují pomaleji než jiné heuristické algoritmy a pokud dopředu neznáme globální extrém, kterého chceme dosáhnout, nikdy nevíme, zda jsme již dosáhli cíleného stavu, kdy algoritmus můžeme zastavit. Mají však i značné výhody:

- Jsou velmi obecně definovatelné a proto se dají nastavit na různou škálu problémů.
- Dokáží se dostat z lokálního extrému.

Pro evoluční proces by pak měla platit ideálně rovnováha dvou cílů:

- Co nejdříve nalézt nejbližší (lokální) optimum v okolí výchozího bodu.
- Co nejdůkladněji prohledat celý prostor všech možných řešení.

Jednotlivé evoluční metody (konfigurace) se pak liší právě tím, jaký mají poměr těchto dvou cílů. Neexistuje jasně daná konfigurace, která by fungovala na určitý typ problému. Musí se tedy zkoušet, jaké nastavení poměru těchto dvou cílů je pro daný problém optimální. Evoluční algoritmy mají také velké množství vstupních parametrů a modifikací. Zatím se tedy pro výběr metody a volbu parametrů používá metoda „pokus omyl“.[14]

2.4.3. Společné rysy evolučních algoritmů

Jak bylo zmíněno mnohokrát, evoluční algoritmy se snaží za pomocí evolučních procesů nalézt řešení náročných optimalizačních problémů. Mezi základní společné rysy evolučních algoritmů patří například:

- Hybridnost
- Rychlost
- Schopnost nalézt řešení ve velké množině dat
- Schopnost dát více optimálních výsledků

Hybridnost

Evoluční algoritmy dokáží pracovat s naprosto nesourodou množinou možných stavů. Lze jako množinu stavu určit například boolean, reálné čísla, celá čísla a také různé číselné soustavy. Zajímavostí je, že lze jako množinu stavů také určit ordinální hodnoty (výčet prvků). Taková množina pak může vypadat nějak takto: $5; 24.36; \pi; false; mlo$

Rychlost

Evoluční algoritmy disponují menší výpočetní náročností oproti jiným optimalizačním metodám, jako jsou například enumerativní metody. Lze konstatovat, že evoluční algoritmy naleznou řešení mnohem rychleji než klasické metody.

Schopnost nalézt řešení ve velké množině dat

Některé velmi složité funkce, u kterých je velmi obtížné, někdy až nemožné odhadnout, kterým směrem se nachází globální minimum, případně maximum. Evoluční algoritmy díky své kombinaci náhodného výběru společně s výběrem cíleným, mají vyšší šanci toto složitě odhadnutelné globální minimum, případně maximum nalézt.

Schopnost dát více optimálních výsledků

Ve většině případů jsou evoluční algoritmy nastaveny tak, aby poskytovaly jedno nejlepší řešení. Díky jednoduchosti evolučního algoritmu na implementaci lze tento algoritmus upravit tak, aby vracel více nejlepších výsledků. Uživatel si pak sám může určit, který z výsledků mu více vyhovuje.

2.4.4. Základní pojmy

V této kapitole budou dovysvětleny, či zopakovány základní pojmy z oblasti optimalizačních algoritmů respektive evolučních algoritmů.

Účelová funkce - fitness funkce

Termínem účelová funkce označujeme funkci, jejíž optimalizací nalezneme optimální nastavení proměnných. Lze si tuto funkci představit jako geometrický problém, jehož nejnižší bod, respektive nejvyšší bod v N + 1 rozměrném prostoru chceme nalézt. N udává počet vstupních parametrů, které jsou určeny k optimalizaci. Správné nastavení této funkce je pro nalezení optimálního řešení velice důležité.

Hodnota účelové funkce - fitness funkce

Každého jedince v populaci si lze představit jako bod v prostoru. Každému takovému bodu v prostoru (jedinci) je přiřazena hodnota fitness. Tato hodnota představuje kvalitu konkrétního jedince v populaci, respektive bodu v prostoru. Tato hodnota nám umožňuje porovnávat jednotlivé jedince a určit tak vítěze. Je tedy zřejmé, že kvalita řešení je úměrná ke správné formulaci fitness funkce. Pokud fitness funkce nebude správně navržená, pak ani výsledek, který algoritmus zhodnotí jako optimální, nemusí být správný.

Prostor řešení

V oblasti optimalizačních algoritmů nazýváme prostorem řešení množinu všech možných řešení. Zpravidla se snažíme v tomto prostoru nalézt to nejlepší řešení, přičemž každý bod tohoto prostoru reprezentuje jedno řešení s určitou hodnotou fitness. Vždy je jisté, že se v prostoru nalézá hledané optimum. Prohledávaný prostor nemusí být spojitý.

Populace

Jak již bylo zmíněno mnohokrát, pro evoluční algoritmy je typická práce s populací. Populace se skládá z jednotlivých jedinců, kde každý jedinec představuje jedno řešení. Populace je tedy skupina možných kombinací vstupních argumentů. Cílem je nalézt optimální nastavení těchto argumentů.

Ke každému jedinci z populace je přiřazena hodnota, která představuje jeho kvalitu. Evoluční algoritmy iteračně vytvářejí nové populace, které vycházejí z předchozí populace.

Nový jedinec z každé populace je tvořen podle vzorového jedince. Takovým jedincům se říká spicient. Spicient bývá definován typem proměnné (integer, boolean, real, atd.) a jeho spodní a horní hranicí. Hranice jsou důležitými kritérii pro omezení prohledávaného prostoru. Například záporná plocha je nerealizovatelná.

No Free Lunch Teorém

Tento teorém je platný obecně pro všechny typy optimalizačních algoritmů. Tento teorém říká, že neexistuje algoritmus, který by dokázal vyřešit jakýkoliv problém lépe než kterýkoliv jiný algoritmus, jinými slovy neexistuje algoritmus, který by byl nejlepší na všechny možné typy problémů. [15]

Další pojmy

Pro zajímavost zde bude v tabulce 1 na straně 14 znázorněné porovnání významů v biologii a optimalizaci.

Pojem	Oblast	
	Optimalizace	Biologie
gen	úsek reprezentace jednoho řešení kódující jeden hledaný parametr	základní jednotka genetiky
chromozom	soubor genů kódující všechny hledané parametry	struktura v jádře eukaryotických buněk, která je nositelem genetické informace jedinců
populace	množina všech aktuálních kandidátů na optimální řešení problému	skupina jedinců stejného druhu obývající stejné území, kteří jsou schopni mezi sebou se množit
vhodnost (fitness)	kvalita jednoho určitého řešení	udává relativní genový příspěvek jedince do další generace
genofond	soubor všech zakódovaných parametrů v rámci všech řešení populace	soubor všech genů v populaci
selekce	operátor výběru nové populace řešení do nové iterace	přirozený výběr
mutace	operátor náhodné změny řešení	náhodná změna genetické informace
rekombinace (křížení)	operátor křížení dvou či více řešení	výměna částí chromozomů během prvního meiotického dělení buněk
adaptace	operátor, který za běhu algoritmu upravuje hodnoty řídících parametrů	vyznačuje znak, jež je výsledkem adaptabilního procesu

Tabulka 1.: Porovnání významu pojmů z oblasti biologie a optimalizace [16]

2.4.5. Popis obecného evolučního algoritmu

Než budou popsány detailně jednotlivé kroky algoritmu, bude zde uveden slovní popis algoritmu pro získání lepšího pochopení, jak algoritmus funguje.

Obecný genetický algoritmus může vypadat takto:

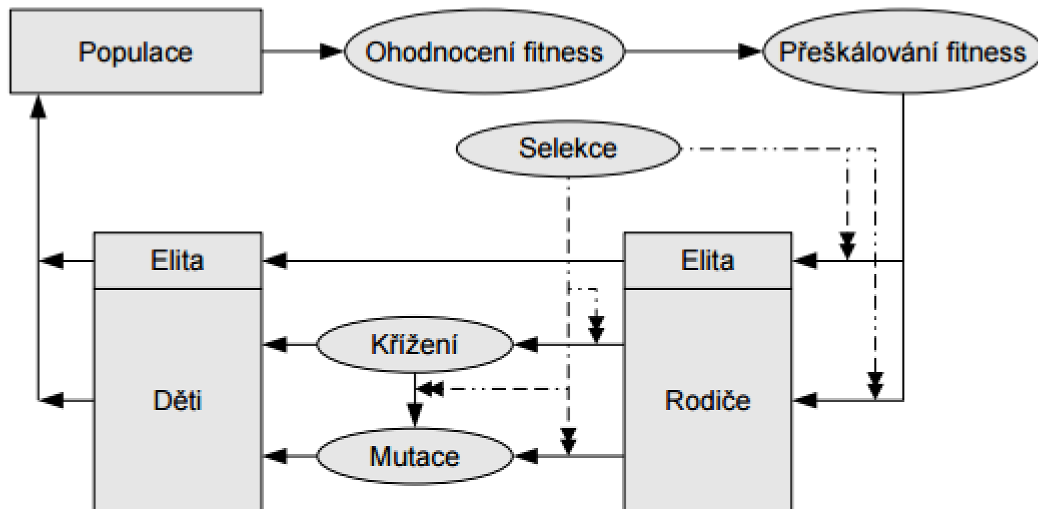
1. **Inicializace** - Vytvořit první náhodnou populaci a jedince.
2. **Začátek cyklu** - Ocenit každého jedince v nové populaci (stanovit kvalitu jedince pomocí účelové funkce).
3. Vytvořit nové jedince procesem reprodukce; to znamená, že budou použity operátory selekce, mutace a křížení.
4. **Rozhodnutí** - rozhodnutí, zda je nalezeno optimální řešení, pokud ne algoritmus bude pokračovat.
5. Vymaže staré nekvalitní jedince, aby se vytvořilo místo pro nové lepší jedince.
6. Ocenit nově vzniklé jedince a umístit do populace.
7. Rozhodnout, zda je splněna ukončovací podmínka, tak vybrat nejlepšího jedince jako řešení problému, jinak opakovat od kroku 3.

Evoluční algoritmus začíná vytvořením nové náhodné populace. Obecně bývá všech N jedinců vytvořeno s náhodným chromozomem. Slovem náhodně se zde myslí, že číselná posloupnost chromozomu je zde vytvořena pomocí pseudonáhodného generátoru náhodných čísel.

Tento přístup úplného náhodného vytvoření jedinců někdy nemusí být úplně dobrá volba, neboť takto nastavený počáteční stav může způsobit ztrátu času. Může se stát, že se vygenerují jedinci, kteří pro daný problém nedávají žádný smysl. Jako dobrý příklad lze uvést problém obchodního cestujícího, kde je vhodné zabránit duplicitě v chromozomu.

Pokud necháme generování první generace zcela náhodné, doporučuje se do fitness funkce začlenit tzv. potrestání (Penally Function). Obecné genetické algoritmy využívají k simulaci procesu reprodukce operace selekce, křížení a mutace. Evoluční strategie obvykle vytváří nové potomky pouze pomocí mutace jednoho z rodičů. Nemusí používat operátor křížení. U genetického programování mohou jednotlivé geny kódovat proměnné i funkce, zatímco geny v genetických algoritmech kódují pouze parametry účelové funkce. [17]

Celý tento slovní popis algoritmu je dobře znázorněn na obrázku 2.



Obrázek 2.: Schéma obecného evolučního algoritmu.

2.4.6. Operátory

Genetické operátory bývají aplikovány na jedince (chromozómy), či na dvojice jedinců. Aplikace těchto generátoru je vlastně základ fungování algoritmu. Použitím genetických algoritmů dochází ke změnám složení populace. Mezi nejdůležitější operátory jednoznačně patří:

- Křížení
- Mutace
- Inverze
- Selekce
- Reprodukce

Tyto operátory lze chápat také jako jednotlivé fáze reprodukce.

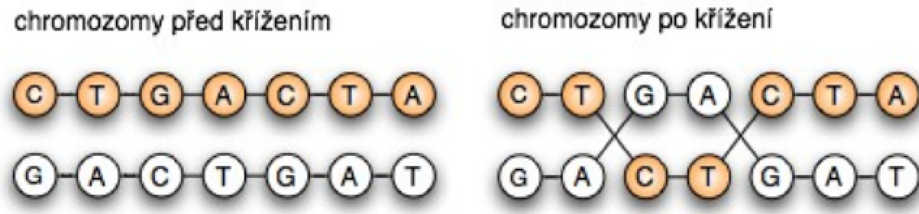
Křížení

Pokud máme vybrány dva jedince (rodičovské chromozomy)

$$a = (a_1, ..., a_m)$$

a

$$b = (b_1, ..., b_m),$$



Obrázek 3.: Proces dvoubodového křížení jedince.

potom tzv. jednobodové křížení je takové, které náhodně zvolí jako bod křížení přirozené číslo

$$k \in [1, ..., m-1]$$

a vytvoří dva nové potomky ve tvaru:

$$c = (a_1, ..., a_k, b_{k+1}, ..., b_m)$$
$$d = (b_1, ..., b_k, a_{k+1}, ..., a_m)$$

Jednobodové křížení je úplně tou základní variantou pro křížení jedinců. Obecně však lze definovat

$$l$$

bodové křížení, které náhodně vybere

$$k, ..., k_l \in [1, ..., m-1]$$

a za každou z těchto pozic vzájemně zaměňuje podřetězce rodičovských chromozomů. Lze tedy říci, že vždy vznikne

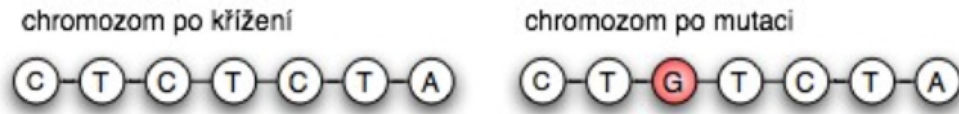
$$l * 2$$

nových potomků.

Křížení se považuje za jednu ze základních věcí genetických algoritmů, které zaručují efektivnost. Křížení vlastně zaručuje rychlou kombinaci výhodných vlastností rodičů tak, že potomek má velkou pravděpodobnost, že kvalitou bude převyšovat své rodiče. Toho výsledku se však docílí pouze v případě, pokud se dobře zvolí oba rodiče z populace (selekce).

Většinou samotný jev křížení bývá omezen pravděpodobností, zda se stane. V ostatních případech křížení nenastane a potomci jsou pak kopií svých rodičů.

Jako příklad je na obrázku 3 znázorněné dvoubodové křížení.



Obrázek 4.: Proces mutace jedince.

Mutace

Mutace je unární operátor, který se aplikuje s určitou pravděpodobností na každý gen jedince (chromozomu). Geny, které jsou vybrány k mutaci, jsou pozměněny na jiný náhodně vybraný symbol z kódové abecedy.

Hlavním důvodem mutace je poskytnout možnost obohatit populaci o nové geny, které v ní nejsou. Například pokud žádný z jedinců v populaci neobsahuje na i-té pozici symbol s kódové abecedy, samotným křížením není šance, aby jedinec se symbolem s na i-té pozici získal nový gen, který v populaci není. Díky operaci mutace je tedy šance, že jedinec je schopen získat gen, který v populaci není. Za pomocí samotného křížení by nebyla šance získat takový gen. Přitom právě tyto geny mohou hrát velkou roli při nalezení kvalitního řešení. Na druhou stranu mutace vědomě narušuje genetickou informaci, je třeba ji aplikovat jen s velmi malou pravděpodobností.

Jako příklad je na obrázku 4 znázorněná mutace jedince.

[10]

2.4.7. Selekce

Selekce je další klíčový prvek v celém evolučním algoritmu. Dodnes volba metody selekce patří k předmětům zkoumání. Jejím hlavním úkolem je zajistit výběr správných kandidátů, kteří se stanou rodiči. Pro každý konkrétní problém, který by měl algoritmus řešit, je nutné zvolit správnou metodu selekce. Existuje celá řada používaných metod selekce.

Je-li dána populace P_t, která má n jedinců. Tyto jedince značíme postupně $i = 1, ..., n$. Je třeba v tuto chvíli zavést pojem očekávaná hodnota (EV). Tento pojem udává očekávaný počet výběrů i-tého jedince v konkrétní generaci t. Častým požadavkem bývá, aby EV byla neklesající funkcí fitness, kterou budeme značit f_i. Nazýváme to selekční tlak, který je důležitý, aby řešení algoritmu konvergovalo k optimálnímu řešení. Míra selekčního tlaku je závislá na výběru konkrétní metody. Mezi základní metody selekce patří:

- Ruleta

- Seřaďovací metoda

- Lineární klasifikace
- Exponenciální klasifikace
- Sigma škálování
- Boltzmannův výběr
- Turnaj

Ruleta

Tato metoda selekce spočívá v principu, že každému jedinci z populace přiřadí pravděpodobnost, že bude vybrán na základě poměru jeho kvality a kvality zbytku populace. Jedinec i je vybrán s pravděpodobností

$$p_i = \frac{f_i}{\sum_{j=1}^{n} f_i}$$

Očekávanou hodnotu výběru i-tého jedince pak získáme jako podíl jeho fitness a průměru fitness celé populace:

$$EV(i) = \frac{n * f_i}{\sum_{j=1}^{n} f_i}$$

Největší nevýhoda této metody je tendence k předčasné konvergenci. Je tím myšleno, že na začátku evoluce máme náhodně vygenerovanou populaci s různou kvalitou jedinců a pouze malý počet jedinců má kvalitu vysokou. Tito jedinci pak dostanou velkou pravděpodobnost, že budou vybráni. Výsledkem je, že se tito jedinci velmi rychle rozmnoží a ostatní části prostoru zůstanou neprozkoumány.

Seřaďovací metoda

Principem této metody je, že se všichni jedinci seřadí podle jejich kvality vzestupně tak, že platí $f_1 \leq f_2 \leq ... \leq f_n$. Očekávaná hodnota pak záleží pouze na pořadí v této množině chromozomů. Nejprve zvolíme náhodně $H = EV(n) > 0$. Následně pro každého jedince stanovíme EV podle předpisu:

$$EV(i) = D + (H - D)\frac{i-1}{n-1},$$

kde $D = EV(1)$. Tato metoda si dává za cíl zabránit rychlé konvergenci algoritmu k výslednému řešení, které nastává v ruletě. Nevýhoda může být například, že při této metodě se ztrácí informace o tom, kteří jedinci jsou výrazně lepší než ostatní.

Lineární klasifikace

Tato metoda vyžaduje, aby všichni jedinci byli seřazeni od nejhoršího jedince k nejlepšímu. Pokud bude platit tato podmínka, potom pravděpodobnost výběru je dána vztahem:

$$p_i = \frac{1}{N}(n^- + (n^+ + n^-)\frac{i-1}{N-1}); i \in [1, 2, ..., N]$$

,

kde $\frac{n^-}{N}$ (resp. $\frac{n^+}{N}$) je pravděpodobnost, že bude vybrán nejhorší (resp. nejlepší) jedinec.

Protože se pravděpodobnost váže k indexu jedince, a nikoliv k jeho fitness, může se stát, že dva jedinci se stejnou fitness obdrží různou pravděpodobnost.

Exponenciální klasifikace

Od lineární klasifikace se liší pouze tím, že pravděpodobnost v populaci není rozdělena lineárně ale s exponenciální závislostí. Stejně jako u lineární klasifikace, index N představuje index nejlepšího jedince v populaci. Pravděpodobnost výběru i-tého jedince je dána:

$$p_i = \frac{c^{N-1}}{\sum_{j=1}^{N} c^{N-j}}; i \in [1, 2, ..., N]$$

Základ exponentu c je parametr metody, který se volí v rozsahu $0 < c < 1$.

Tato metoda se považuje za jednu z nejlépe fungujících vůbec. Změnou parametru c je možno docílit vhodného selekčního tlaku při nízké ztrátě variability.

Sigma škálovaní

Další z metod, která se snaží zabránit nedostatkům ruletové metody. Očekávaná hodnota je v této metodě v podobě funkce fitness f_i, s výběrovým průměrem $\acute{f} = \frac{1}{N}\sum_{j=1}^{n} f_i$ a výběrové směrodatné odchylky $\sigma = \sqrt{\frac{1}{n-1}\sum_{j=1}^{n}(f_j - \acute{f})^2}$.

$$EV(i) = \begin{cases} 1 + \frac{f_i - \acute{f}}{2\sigma} & pro \sigma \neq 0 \\ 1 & pro \sigma = 0 \end{cases}$$

Výhodou této metody je, že na začátku evoluce, kdy je směrodatná odchylka vysoká, nedostanou nejlepší jedinci vysokou pravděpodobnost výběru a tím nevstoupí do procesu reprodukce. Jakmile směrodatná odchylka klesne, začínají vyhrávat lepší jedinci. Selekční tlak u této metody zůstává přibližně stejný po celou dobu běhu algoritmu.

Boltzmannův výběr

Tato metoda je podobná technice simulovaného žíhaní. Pracuje s parametrem t, který značí teplotu systému, ta může během procesu klesat. Očekávaná hodnota je vyjádřena vztahem:

$$EV(i) = \frac{n * e^{\frac{f_i}{t}}}{\sum_{j=1}^{n} f_j}$$

Tato metoda je jednou z metod, která zdokonaluje metodu Sigma škálovaní. Jde o to, že se snaží měnit selekční tlak v průběhu algoritmu tak, aby ze začátku byla šance na výběr pro každého jedince shodná a populace zůstala po nějakou dobu rozmanitá. Později, kdy se najde ta správná oblast, zvýší se selekční tlak a tím se urychlí nalezení optimálně kvalitního řešení. Vliv selekčního tlaku je ideální měnit po celý průběh evoluce. Teplota t je klesající funkcí času algoritmu. Selekční tlak s klesající teplotou roste.

Turnaj

Tato metoda selekce je velmi jednoduchá a vhodná také k paralelní implementaci. Celý princip spočívá v tom, že vždy mezi sebou soutěží dva jedinci a horší z nich je vyřazen. Lepší z nich je zařazen do reprodukce. Tato metoda je poměrně dost věrná metodám biologické evoluce, kdy samci spolu soupeří a vyhrává silnější.

Jiná varianta této metody počítá s parametrem $r > 0.5$. Zároveň jsou vybrány dva jedinci do turnaje. Následně se zvolí náhodné číslo $k \in [0, 1]$. Pokud je $k < r$, vybereme silnějšího jedince a v opačném případě slabšího jedince.

Hlavní výhodou této metody je, že se nemusí při implementaci procházet celá populace a počítat průměrnou kvalitu či jiné charakteristiky a tím šetřit procesorový čas. Selekční tlak u této metody je přibližně stejný selekčnímu tlaku seřaďovací metody.

Z pravidla do turnaje nemusí být zvoleni pouze dva jedinci. Často lze počet účastníků turnaje měnit. Tím lze určovat kvalitu výsledné selekce jedinců. Pokud bude do turnaje vybráno více jedinců je větší pravděpodobnost, že budou vybráni lepší jedinci. Naopak, pokud turnaj bude pouze mezi dvěma jedinci, pak je stejná pravděpodobnost, že se do turnaje dostane špatný jedinec, tak jako dobrý jedinec.

Metody, které byly popsány výše lze různě modifikovat a případně zlepšovat. Mezi takové vylepšení jednoznačně patří tzv. **Sociální injekce** a **Elitismus**. [10]

Sociální injekce

Sociální injekce (často také sociální tlak) je metoda oživení populace. Jedná se o to, že při vytváření nové populace jedinců je při použití této metody sociální injekce vždy vytvořeno několik nových jedinců, kteří nejsou vytvořeni křížením, jako ostatní jedinci, ale pseudo-náhodným generováním jejich chromozomu. Tito jedinci vnesou do populace, která se udává jedním směrem větší náhodu při tvorbě následujících populací.

Elitismus

Pokud není použit Elitismus, často se stává, že nejsilnější jedinci nepřejdou do další generace, protože například nebyli vybráni pro křížení a nebo jejich genetický materiál byl během křížení či mutace zničen. Elitismus zajistí to, že se nejlepší jedinci automaticky dostanou do nové generace a teprve potom začne samotná selekce. Pro splnění monotónnosti populace stačí přenést do nové populace jen jednoho nejlepšího jedince. [10]

3. Praktická část

Jako praktická část této práce byla vyvinuta knihovna, která bude v této kapitole podrobně popsána. Knihovna slouží pro využití evolučních algoritmů v biomedicíně. Celá knihovna je implementována v jazyce Java. Jedná se tedy o ryze objektové programování, které splňuje tyto vlastnosti:

- Zapouzdření
- Dědičnost
- Polymorfismus

To umožňuje velkou volnost a znovupoužitelnost při samotné implementaci knihovny.

3.1. Použité technologie

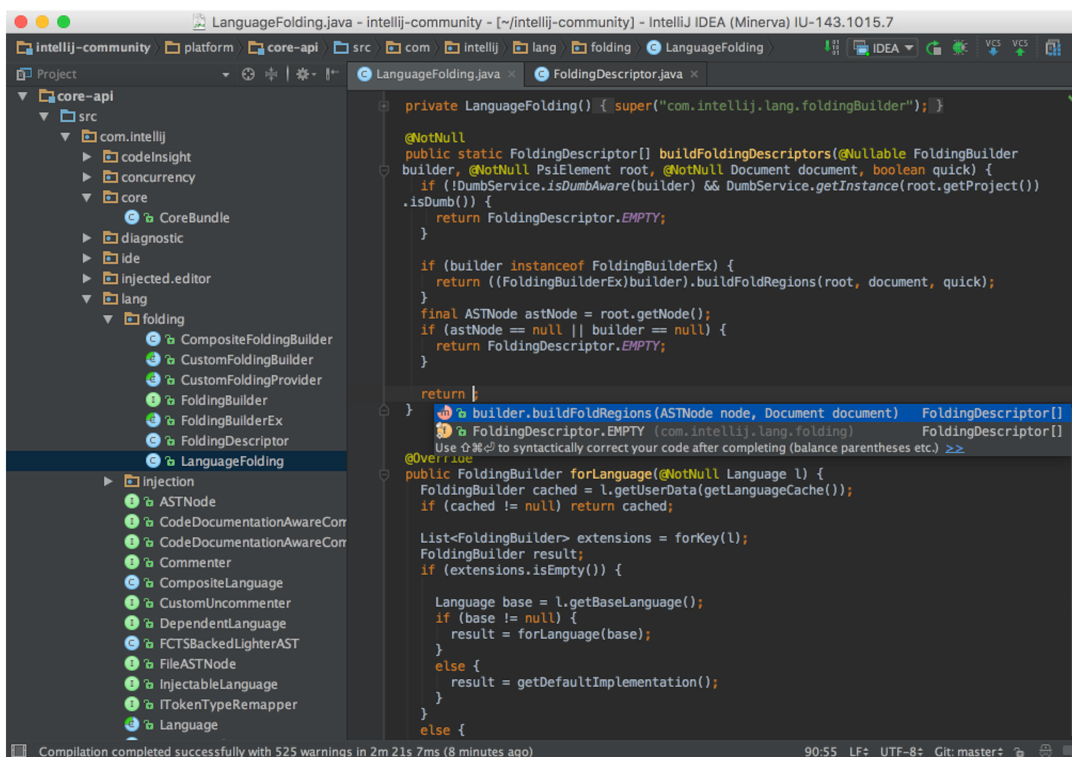
Technologie, které byly použity k implementaci této knihovny evolučních algoritmů není mnoho. Hlavní technologií je **Java verze 8**, která je aktuálně nejnovější. Dále je pro podporu závislostí na jiných knihovnách použitý **Apache Maven**.

3.2. Popis knihovny

Jak bylo uvedeno v úvodu kapitoly, jedná se o knihovnu, která umožňuje využít evoluční algoritmus v oblasti biomedicíny. Celá knihovna je implementovaná v jazyce Java za pomocí vývojového nástroje *IntelliJ IDEA*, který nabízí mnoho užitečných nástrojů a usnadňují vývoj. Ukázka tohoto vývojového nástroje je na obrázku 5.

Samotná knihovna nedisponuje žádným grafickým uživatelským prostředím. Na veřejnost vystupuje pouze s veřejným API[1]. Pro umožnění testování je implementovaná spouštěcí třída, která umožňuje celou knihovnu spustit jako samostatný program v příkazové řádce.

[1] API (Aplikační programové rozhraní) se označuje v informatice sbírka funkcí, které může programátor použít, ale nemusí znát implementaci. [18]



Obrázek 5.: Ukázka vývojového nástroje Intellij IDEA.

Knihovna je implementována tak, aby se dala externě konfigurovat. K nastavení samotného evoluční algoritmu je využit tzv. „properties soubor". Jedná se o *key-value úložiště*, to umožňuje snadno a rychle nastavit aplikaci.

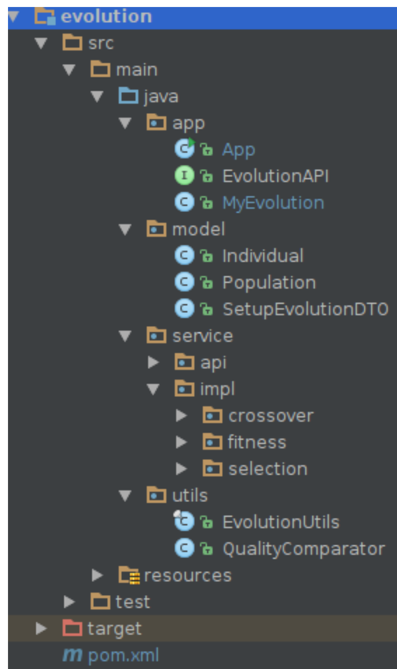
Princip *key-value úložišť* je, že program očekává pod určitým klíčem hodnotu, kterou uživatel nastavil. Lze tak před každým spuštěním aplikace přenastavit její hodnoty, bez nutnosti znovu kompilovat celou knihovnu. Seznam všech hodnot, které lze nastavit, je popsán níže v kapitole 3.5.

V případě spuštění knihovny jako samostatné aplikace pomocí příkazové řádky, je vyžadován konfigurační soubor, který obsahuje všechny nastavitelné hodnoty. Tento soubor je očekáván ve stejné složce jako samotná knihovna. Pokud soubor chybí nebo je chybný, knihovna automaticky zvolí výchozí hodnoty pro parametry. Podrobněji o parametrech je popsáno v kapitole 3.5.

3.2.1. Struktura knihovny

V této kapitole bude popsána struktura jednotlivých tříd v celé knihovně. Jejich umístění v balíčcích.

Jak je vidět na obrázku 6, lze knihovnu rozdělit na tyto hlavní balíčky:



Obrázek 6.: Rozložení balíčku knihovny.

- app
- model
- service
  - crossover
  - fitness
  - selection
- util

Balíček - app

Balíček *app* obsahuje základní třídy, které komunikují s okolím. Jedná se o již zmíněné veřejné API, pomocí kterého lze tuto knihovnu spouštět z jiné aplikace. Dále třídu, která umožňuje pustit knihovnu jako samostatný program běžící v příkazové řádce.

Balíček - model

V tomto balíčku jsou umístěny hlavně třídy, se kterými evoluční algoritmus pracuje (Jedinec, Populace). Díky objektovému návrhu můžou být jedinec a populace považovány za objekty, které mají své vlastnosti a metody.

Balíček - service

Důležitý balíček kde jsou umístěny veškeré služby, které evoluční algoritmus využívá. Základem jsou rozhraní pro křížení, fitness funkci a výběr. Dále jsou v tomto balíčku umístěny různé implementace [2] výše uvedených rozhraní.

Balíček - util

Poslední balíček nabízí utility třídu [3] společně s komparátorem.

3.2.2. Popis důležitých tříd

V této kapitole budou podrobně rozebrány důležité třídy [4] knihovny evolučního algoritmu. Vybrány budou třídy, které mají význam pro tuto knihovnu.

Modelové třídy

V celé knihovně jsou použity pouze dvě třídy, se kterými je uvažováno jako s modelovými třídami. Vytváří tedy objekty ze tříd a používá pro vykonání celého evolučního algoritmu.

- Individual
- Population

Třída Jedinec - Individual

```
8  /**
9   * Member of Population
10  */
11 public class Individual {
12
13     private String chromosome;
14
15     private Double quality;
16
17     getry, setry ...
18 }
```

Ukázka kódu 1: Atributy třídy Jedinec

[2] Implementace rozhraní znamená, že třída je povinná nabízet všechny metody, které rozhraní definuje.
[3] Soubor statických metod, u kterých je pravděpodobné, že by mohli být užitečné ve více částech systému.[19]
[4] Třída představuje skupinu objektů, které představují stejné vlastnosti.[20]

Každý objekt vytvořený ze třídy Jedinec (instance třídy Jedinec) disponuje atributy *chromosome*, *quality*.

Chromosome představuje chromozom daného jedince v populaci. Je reprezentován jako binární číslo složené z „nul a jedniček", zapsané jako textový řetězec. Lze použít datový typ **String**. Použitím datového typu String dovoluje využití výhod práce s textem v jazyce Java. Lze například jednoduše určit kolik logických jedniček a logických nul chromozom má. Nabízí se využít například kolekci binárních čísel, ale to má jednu velkou nevýhodu. Při práci s kolekcí čísel by se muselo vždy procházet celou kolekcí a tím zpomalovat běh evolučního algoritmu.

Délka chromozomu je nastavitelná. Uživatel si může podle situace navýšit nebo snížit počet genů v chromozomu.

Quality Tento atribut určuje kvalitu daného jedince vzhledem k populaci. Je vyjádřena reálným číslem a platí, že čím vyšší tím, je jedinec kvalitnější a je pravděpodobnější, že bude vybrán pro křížení. Kvalita jedince je výsledkem fitness funkce.

Dále zde budou popsány metody, kterými třída Jedinec disponuje.

```
17    /**
18     * Generate random chromosome like String. (Don't use loop over
           all gens of chromosome)
19     * @param length - length of chromosome
20     * @param countOfPositiveGens - count of "1" in chromosome.
           Other items are "0"
21     */
22    public void generateChromosome(int length, int
          countOfPositiveGens) {
23        StringBuilder chromosome = new StringBuilder(length);
24
25        Set<Integer> generatedPositions = getRandomPositions(length,
              countOfPositiveGens);
26
27        for (int i = 0; i < length; i++) {
28            if (generatedPositions.contains(Integer.valueOf(i))) {
29                chromosome.append(1);
30            } else {
31                chromosome.append(0);
32            }
33        }
34        this.chromosome = chromosome.toString();
35    }
```

```
36 }
```

Ukázka kódu 2: Metoda generateChromosome

Metoda generateChromosome (ukázka kódu 2), kterou nabízí třída Jedinec slouží k vygenerování náhodného chromozomu. Využívá se při vytváření úplně první populace, kdy je potřeba, aby nově vzniklý jedinec měl náhodné geny. Vstupem do metody generateChromosome (ukázka kódu 2) jsou dva parametry. První udává, jak má být výsledný chromozom dlouhý (počet genů) a druhý parametr kolik genů má být tzv. booleanových jedniček (logická pravda). Výsledný chromozom je řetězec, který obsahuje jedničky a nuly, tedy datový typ boolean.

```
37     /**
38      * Helper method which return Set of positions from interval 0 -
           size of chromosome.
39      * @param length - length of String
40      * @param count - size of result Set
41      * @return - Set with positions
42      */
43     public Set<Integer> getRandomPositions(int length, int count) {
44         Set<Integer> generated = new LinkedHashSet<>(count);
45         while (generated.size() < count) {
46             Integer next =
                   EvolutionAPI.randomGenerator.nextInt(length);
47             if (generated.contains(next)) {
48                 next = EvolutionAPI.randomGenerator.nextInt(length);
49             }
50             generated.add(next);
51         }
52         return generated;
53     }
54 }
```

Ukázka kódu 3: Metoda getRandomPositions

Metoda getRandomPositions (ukázka kódu 3) vrací množinu náhodných pozic z chromozomu jedince. Disponuje dvěma vstupy. První vstup udává délku chromozomu a druhý říká, kolik prvků má obsahovat výsledná množina.

Třída Populace - Population

```
 8 /**
 9  * Population
10  */
```

```
11 public class Population {
12
13     private Double crossRate;
14     private Double mutationRate;
15     private Integer size;
16
17     private Integer lengthOfChromosome;
18     private Integer countOfPositiveGens;
19
20     private Integer socialPressure;
21     private boolean penalty;
22
23     private List<Individual> individuals = new ArrayList<>();
24     private Individual bestIndividual;
25     private Individual worstIndividual;
26
27     getry, setry ...
28 }
```

Ukázka kódu 4: Atributy třídy populace

Jak je vidět v ukázce kódu 4, třída *Populace* má mnoho atributů, které udávají vlastnosti populace.

CrossRate, MutationRate Tyto parametry udávají s jakou pravděpodobností dojde ke křížení, respektive k mutaci. Lze mít rozdílné nastavení pravděpodobnosti pro křížení a mutaci.

LengthOfChromosome, CountOfPositiveGens Informují o délce a charakteru chromozomu. Parametr *CountOfPositiveGens* udává, z kolika logických jedniček se bude skládat chromozom. Jedná se o maximální možný počet. Algoritmus náhodně vybere od jedné do nakonfigurované hranice počet jedniček.

SocialPressure Parametr *socialPressure* udává kolik nových členů má být vloženo do existující populace při sociální injekce. Pokud je nastavena nula, sociální injekce nebude provedena. Metoda sociální injekce byla popsána výše v kapitole Sociální injekce2.4.7.

Penalty Zajímavý atribut, který říká, zda má být populace trestána. Penalizace evolučního algoritmu je jedna z metod, jak zabránit evolučnímu algoritmu, aby se udával jedním směrem. Konkrétní řešení penalizace tohoto evolučního algoritmu je popsáno níže v kapitole 3.3.2.

V poslední řadě je v každé populaci uložen list všech jedinců, kteří do populace patří. Pro zpětné dohledání a možnost vytvářet statistiky, je zde uvedeno, který jedinec je nejlepší, respektive nejhorší.

```
110     public Individual createRandomIndividual() {
111        Individual individual = new Individual();
112           //generate random chromosome
113          int randomInt =
                EvolutionAPI.randomGenerator.nextInt(countOfPositiveGens
                -1) + 1;
114          individual.generateChromosome(lengthOfChromosome, randomInt);
115          //set random quality
116          individual.setQuality(100.0*Math.random());
117          return individual;
118     }
```

Ukázka kódu 5: Metoda createRandomIndividual

Jedna z metod, kterou nabízí třída *Populace*, ukazuje ukázka kódu 5. Tato metoda slouží hlavně při vytváření první populace, u které je vyžadováno, aby byla vytvořena z jedinců, kteří mají náhodné vlastnosti. Je vytvořen právě jeden nový jedinec, kterému se přiřadí náhodně vygenerovaný chromozom a náhodná kvalita. Není chtěné, aby kvalita nějak souvisela s vygenerovaným chromozomem. To by mohlo ovlivňovat vývoj populace jedním směrem.

Metoda je také využita při sociální injekci, kde jsou pro oživení, již existující populace, vloženi nový jedinci.

Služby - rozhraní [5]

Knihovna evolučních algoritmů pracuje s těmito základními rozhraními:

- CrossoverService
- FitnessFunctionService
- SelectionService

CrossoverService

```
10      public interface CrossoverService {
11
```

[5] Rozhraní (interface) je množina metod, kterými třída disponující rozhraním musí implementovat. Rozhraní pouze popisuje metody.[21]

```
12        /**
13         * Cross over via @param parent1 and @param parent2.
14         * @return children
15         */
16        Individual[] crossover(Individual parent1, Individual parent2);
17    }
```

Ukázka kódu 6: Rozhraní CrossoverService

Jak název napovídá, jedná se o rozhraní, které má na starosti křížení jedinců v evolučním algoritmu. V ukázce kódu 6 je patrné, že rozhraní má pouze jednu metodu. Vstupními parametry metody jsou jedinci - rodiče. Metoda provede křížení a výstupem je pole nových jedinců - potomků.

FitnessFunctionService

```
5    public interface FitnessFunctionService {
6
7        /**
8         * Calculate FITNESS function
9         * @param individual
10        * @return value of fitness for individual
11        */
12       Double calculateFitnessFunction(Individual individual);
13   }
```

Ukázka kódu 7: Rozhraní FitnessFunctionService

Rozhraní znázorněné v ukázce kódu 7 budou implementovat třídy, které slouží pro výpočet fitness funkce. Opět rozhraní nabízí pouze jednu metodu. Vstupním parametrem pro metodu je samotný jedinec, pro kterého má být fitness funkce vypočítána. Výsledkem této metody je kvalita daného jedince.

SelectionService

```
7    public interface SelectionService {
8
9        /**
10        * Random selection
11        */
12       Individual randomSelectIndividual(List<Individual>
             individuals, double sumOfFitnessFunction);
13 }
```

Ukázka kódu 8: Rozhraní SelectionService

Rozhraní s názvem *SelectionService* (ukázka kódu 8) se stará o výběr jedince z populace. Nabízená metoda vyžaduje pro vstup list všech jedinců z populace a součet hodnoty fitness funkce všech jejich jedinců. Výsledkem volání této metody je vybraný jedinec z dané populace vhodný ke křížení.

3.2.3. Vlastní implementace služeb - crossover

V knihovně evolučních algoritmů jsou dostupné (implementované) dvě různé metody křížení. Obě třídy implementují jedno rozhraní CrossoverService (ukázka kódu 6). To umožňuje konfigurovat algoritmus a nastavit před samotným spuštěním jakou metodu křížení má použít.

Implementace BinaryMaskCrossover

Princip této metody křížení lze popsat ve dvou krocích.

1. Náhodně vytvoří binární masku.
2. Aplikuje binární masku na vybrané jedince (rodiče).

Třída celkem obsahuje tři metody. Hlavní metoda třídy (ukázka kódu 9), která je veřejně dostupná pro okolí pochází z rozhraní CrossoverService (ukázka kódu 6). Nejprve metoda náhodně vytvoří pomocí metody generateRandomBinaryMask (ukázka kódu 10) binární masku. Takto vygenerovaná maska je aplikovaná pomocí metody applyMask (ukázka kódu 11) na oba rodiče, kteří do metody vstupují jako vstupní parametry. Vytvoří se dva nový jedinci s chromozomem, na který je použitá vygenerovaná binární maska pomocí binární operace XOR [6]. Výsledkem jsou dva jedinci s chromozomem rodičů, na které byla aplikována vygenerována binární maska.

```
13    @Override
14     public Individual[] crossover(Individual parent1, Individual
           parent2) {
15         boolean[] randomBinaryMask =
               generateRandomBinaryMask(parent1.getChromosome().length());
16
17         //create offspring
18         Individual[] children = new Individual[2];
19         children[0] =
               EvolutionUtils.createIndividual(applyMask(parent1,
               randomBinaryMask));
20         children[1]=
               EvolutionUtils.createIndividual(applyMask(parent2,
               randomBinaryMask));
```

[6] XOR - exkluzivní součet vrací binární 1 pokud se sčítanci liší. V ostatních případech vrací binární 0.[22]

```
21          return children;
22      }
```

Ukázka kódu 9: Metoda crossover třídy BinaryMaskCrossover

Úkolem metody generateRandomBinaryMask (ukázka kódu 10) je zajistit náhodné vygenerovaní binární masky. Vstupem do metody je délka výsledné masky. Binární maska je v podobě pole binárních čísel, kde délka pole je vstupním parametrem. Za použití pseudonáhodného generátoru čísel [7] je postupně vygenerované celé pole binárních čísel (maska).

```
42      private boolean[] generateRandomBinaryMask(int length) {
43          boolean[] mask = new boolean[length];
44          for (int i = 0; i < length; i++) {
45              mask[i] = EvolutionAPI.randomGenerator.nextBoolean();
46          }
47          return mask;
48      }
```

Ukázka kódu 10: Metoda generateRandomBinaryMask třídy BinaryMaskCrossover

Metoda applyMask (ukázka kódu 11) zajišťuje správné aplikovaní binární masky. Vstupem je jeden z rodičů a binární maska, která má být aplikovaná. Výstupem je nový chromozom, který bude patřit potomkovi. Postupně se zde prochází chromozom rodiče. Prochází se gen po genu a společně s binární maskou jsou vstupem do binární operace XOR. Výsledkem XOR operace je nový gen, který je vložen do výsledného chromozomu.

```
24      private String applyMask(Individual parent, boolean[] mask) {
25          String parentChromosome = parent.getChromosome();
26
27          StringBuilder childChromosome = new StringBuilder();
28          for (int i = 0; i < mask.length; i++) {
29              boolean gen = parentChromosome.charAt(i) == '1';
30              boolean maskGen = mask[i];
31
32              //XOR
33              if ((gen && maskGen) || (!gen && !maskGen)) {
34                  childChromosome.append(0);
35              } else {
36                  childChromosome.append(1);
37              }
```

[7] Deterministický počítač není schopný generovat čistě náhodná čísla. Čísla jsou předvídatelně generována, nicméně mají charakter náhodného šumu, což je postačující. [23]

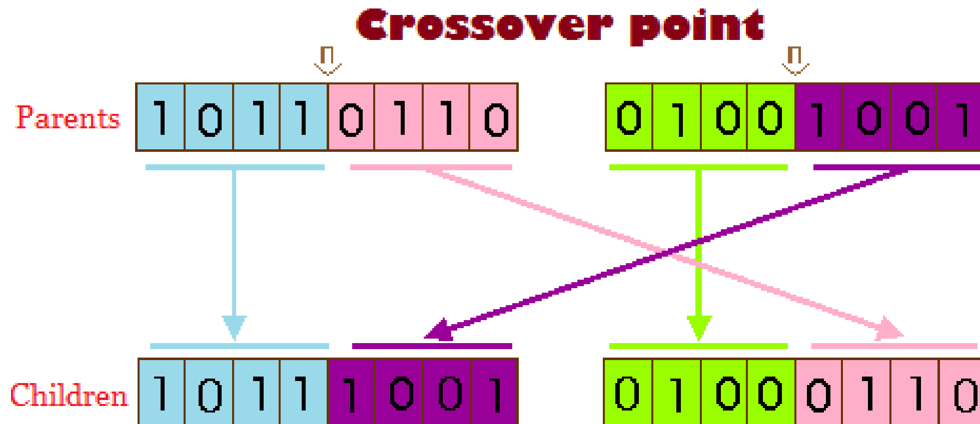
```
38         }
39         return childChromosome.toString();
40     }
```

Ukázka kódu 11: Metoda applyMask třídy BinaryMaskCrossover

Implementace OnePointCrossover

Tato implementace (metoda) křížení patří mezi ty jednodušší. Jde o základní metodu křížení, kdy se náhodně nalezne bod v chromozomu a rozdělením vzniknou právě dva jedinci. Pro lepší znázornění zde bude uvedena ukázka implementace 12

```
24     @Override
25     public Individual[] crossover(Individual parent1, Individual
           parent2) {
26         //parent's chromosomes
27         String parentChromosome1 = parent1.getChromosome();
28         String parentChromosome2 = parent2.getChromosome();
29 
30         int length = parentChromosome1.length();
31         //get random point crossover
32         Integer crossoverPoint =
               EvolutionAPI.randomGenerator.nextInt(length);
33 
34         String childChromosome1;
35         String childChromosome2;
36 
37         //create child's chromosomes
38         if (Math.random() < 0.5) {
39             childChromosome1 = parentChromosome1.substring(0,
                   crossoverPoint) +
                   parentChromosome2.substring(crossoverPoint, length);
40             childChromosome2 = parentChromosome2.substring(0,
                   crossoverPoint) +
                   parentChromosome1.substring(crossoverPoint, length);
41         } else {
42             childChromosome1 = parentChromosome2.substring(0,
                   crossoverPoint) +
                   parentChromosome1.substring(crossoverPoint, length);
43             childChromosome2 = parentChromosome1.substring(0,
                   crossoverPoint) +
                   parentChromosome2.substring(crossoverPoint, length);
44         }
```



Obrázek 7.: Znázornění jednobodového křížení. [24]

```
45
46          //create offspring
47          Individual child1 =
               EvolutionUtils.createIndividual(childChromosome1);
48          Individual child2 =
               EvolutionUtils.createIndividual(childChromosome2);
49
50          return new Individual[]{child1, child2};
51      }
```

Ukázka kódu 12: Metoda crossover třídy OnePointCrossover

Vstupem do metody jsou vybraní rodiče. Důležité jsou jejich chromozomy. Předpokladem této metody je, že délka obou chromozomů obou rodičů je shodná. Pomocí pseudonáhodného generátoru je vybrána náhodná pozice a je zvolen bod křížení.

Chromozomy rodičů jsou v tomto bodě rozděleny a promíchány. Výsledkem jsou dva noví jedinci s nakombinovanými chromozomy od obou rodičů. Pro lepší pochopení jednobodového křížení je zde vložen obrázek 7.

3.2.4. Vlastní implementace služeb - selection

V této kapitole budou podrobně rozebrány všechny metody výběru, které tato knihovna evolučních algoritmů nabízí. Každá jedna třída implementuje rozhraní SelectionService (ukázka kódu 8). To umožňuje uživatelem zvolenou metodu selekce použít pro běh evolučního algoritmu. Celkem tato knihovna nabízí tři metody výběru, které zde budou postupně popsány.

1. ProbabilityLineSelection

2. TournamentSelection

3. RouletteSelection

Implementace ProbabilityLineSelection

Tato zajímavá metoda selekce, kde její princip spočívá v tom, že před samotným výběrem je vytvořena přímka, kde jedinec, který je kvalitnější zabírá na přímce více místa. Takhle jsou rozloženy na přímce všichni jedinci z populace. Nakonec se náhodně vybere bod z přímky, a tak se zvolí jedinec, který danému bodu náleží.

Samotná implementace obsahuje dvě metody. První z nich je privátní[8] metoda generateQualityLine (ukázka kódu 13), která má na starosti vytvoření „přímky kvality“. Projde postupně všechny jedince z populace a vytváří přímku tak, že vždy vezme kvalitu jedince, kterou vloží nakonec a přímka se tak prodlužuje. Reprezentace přímky je spíše abstraktní. Ve skutečnosti je výsledkem mapa[9], kde klíčem je rozsah bodů na přímce a hodnotou je onen jedinec.

```
33        private Map<DoubleRange, Individual>
              generateQualityLine(List<Individual> individuals, double
              sumOfFitnessFunction) {
34        Map<DoubleRange, Individual> result = new HashMap<>();
35        double lastRange = sumOfFitnessFunction;
36        for (Individual individual : individuals) {
37            Double individualQuality = individual.getQuality();
38            DoubleRange range;
39            //for last the worst individual set end of range as "0"
40            if (individualQuality.equals(0d)) {
41                range = new DoubleRange(lastRange, 0d);
42            } else {
43                range = new DoubleRange(lastRange, lastRange -
                      individualQuality);
44            }
45
46            lastRange -= individualQuality;
47            result.put(range, individual);
48        }
49        return result;
50    }
```

Ukázka kódu 13: Metoda generateQualityLine třídy ProbabilityLineSelection

[8]Klíčové slovo private určuje, že metoda je viditelná pouze uvnitř třídy.

[9]Objekt, který spojuje libovolný klíč k libovolné hodnotě. Není povoleno, aby existoval duplicitní klíč. [25]

Druhá metoda randomSelectIndividual (ukázka kódu 14) je veřejně vystavená pro okolí. Vstupem do metody je list všech jedinců populace, ze které bude vybrán jeden. Druhým vstupem je součet kvality všech jedinců, který bude sloužit pro informaci, jak dlouhá přímka bude.

Nejprve je list jedinců setříděn podle kvality jedinců. Tedy nejlepší jedinec bude první a nejhorší poslední. S takto setříděným listem se zavolá metoda generateQualityLine (ukázka kódu 13), která sestaví přímku jedinců. Bod výběru bude vzat z pseudonáhodného generátoru. Nakonec metoda projde postupně celou přímku a vybere jedince, kterému náleží vylosovaný bod. Takový jedinec bude vybrán.

```
18    @Override
19    public Individual randomSelectIndividual(List<Individual>
          individuals, double sumOfFitnessFunction) {
20        //firstly sort by quality
21        Collections.sort(individuals,new QualityComparator());
22        //generate line
23        Map<DoubleRange, Individual> lineMap =
              generateQualityLine(individuals, sumOfFitnessFunction);
24        int selectPoint = EvolutionAPI.randomGenerator.nextInt((int)
              sumOfFitnessFunction - 1) + 1;
25        for (DoubleRange range : lineMap.keySet()) {
26            if (range.containsInteger(selectPoint)) {
27                return lineMap.get(range);
28            }
29        }
30        return null;
31    }
```

Ukázka kódu 14: Metoda randomSelectIndividual třídy ProbabilityLineSelection

Implementace TournamentSelection

Další zajímavá metoda selekce, která se staví k výběru, jako k souboji. Pro lepší ilustraci zde bude vložena ukázka samotné implementace metody randomSelectIndividual (ukázka kódu 15) třídy TournamentSelection.

```
17    @Override
18    public Individual randomSelectIndividual(List<Individual>
          individuals, double sumOfFitnessFunction) {
19
20        int numberOfCompetitors =
              EvolutionAPI.randomGenerator.nextInt(8) + 2;
21
```

```
22          List<Individual> competitors = new
                ArrayList<>(numberOfCompetitors);
23          for (int i = 1; i <= numberOfCompetitors; i++) {
24              int randomIndex =
                    EvolutionAPI.randomGenerator.nextInt(individuals.size()
                    - 1);
25
26              competitors.add(individuals.get(randomIndex));
27          }
28
29          Collections.sort(competitors, new QualityComparator());
30          return competitors.get(0);
31      }
```

Ukázka kódu 15: Metoda randomSelectIndividual třídy TournamentSelection

Princip, jak implementovat tuto metodu selekce je jednoduchý. V první řadě se náhodně určí z kolika účastníku se turnaj bude skládat. Ti jsou pak zcela náhodně vybrány do turnaje. Konkrétně v této implementaci je daný maximální počet 10 účastníků. Turnaj vyhrává nejkvalitnější jedinec.

Implementace RouletteSelection

Zřejmě nejznámější metoda selekce se nazývá ruleta. Tato implementace (ukázka kódu 16) ruletové metody funguje takovým způsobem, že nejprve se určí náhodný bod zastavení a postupně se sčítají kvality jednotlivých jedinců, dokud nepřesáhne náhodně vybraný bod. V ten moment se vybírá daný jedinec.

```
10      @Override
11      public Individual randomSelectIndividual(List<Individual>
            individuals, double sumOfFitnessFunction) {
12          double sum = sumOfFitnessFunction;
13          individuals.sort(new QualityComparator());
14
15          double fitnessPoint = Math.random() * sumOfFitnessFunction;
16
17          for (Individual individual : individuals) {
18              Double currentQuality = individual.getQuality();
19              sum -= currentQuality;
20              //for last the worst individual set end of range as "0"
21              if (currentQuality.equals(0d)) {
22                  sum = 0;
23              }
24
```

```
25              if (sum < fitnessPoint) {
26                  return individual;
27              }
28          }
29          return null;
30      }
```

Ukázka kódu 16: Metoda randomSelectIndividual třídy RouletteSelection

3.2.5. Vlastní implementace služeb - fitness funkce

Poslední kapitolou, která popisuje implementace služeb jsou implementace rozhraní FitnessFunctionService (ukázka kódu 7). V této knihovně evolučních algoritmů je implementována pouze jedna prostá fitness funkce určená primárně pro testovací účely.

Implementace MyFitnessFunction

Jak bylo uvedeno výše, toto je jediná implementace fitness funkce. Jedná se o jednoduchou fitness funkci. Její princip je takový, že kvalita jedince (reálné číslo) je určena podle počtu pozitivních genu v chromozomu jedince. Pro lepší znázornění je zde uvedena ukázka kódu 17.

```
12      @Override
13      public Double calculateFitnessFunction(Individual individual) {
14          String chromosome = individual.getChromosome();
15          int countOfPositiveGens =
              EvolutionUtils.getCountOfPositiveGens(chromosome);
16          return (double) countOfPositiveGens * Math.random();
17      }
```

Ukázka kódu 17: Metoda calculateFitnessFunction třídy MyFitnessFunction

3.3. Hlavní proces

Tato kapitola bude věnována nejdůležitějšímu procesu a to je samotná implementace evolučního algoritmu.

3.3.1. Rozhraní Evoluce

Veřejné rozhraní EvolutionAPI (ukázka kódu 18) této knihovny evolučních algoritmů nabízí pouze dvě základní metody.

1. createInitialPopulation
2. createNextPopulation

První metoda se z pravidla volá jako první a jejím úkolem je zajistit vytvoření první náhodné vstupní populace. Vstupem do této metody je DTO[10] s nastavením celého procesu. Výstupem metody je první náhodně vytvořená populace.

Druhá metoda slouží k vytvoření každé další populace. Vstupem do této metody je předchozí populace. Podle principů evoluce bude vytvořena nová „lepší" populace, která je výstupem této metody.

```
17      /**
18       * Tahle metoda se musi pustit prvni. Slouzi k vytvořeni první
            populace.
19       * Vytvori se nahodny jedinci
20       * @param setupProcess prepravka, kde jsou parametry k evoluci
21       */
22      Population createInitialPopulation(SetupEvolutionDTO
            setupProcess);
23
24      /**
25       * Metoda vytvori z predesle populace novou populaci. Provede
            selekci, krizeni, mutaci
26       * @param previousPopulation
27       * @return nova LEPSI populace
28       */
29      Population createNextPopulation(Population previousPopulation);
```

Ukázka kódu 18: Veřejné rozhraní Evolučního algoritmu

3.3.2. Implementace rozhraní evoluce

Knihovna evolučních algoritmů nabízí pouze jednu implementaci rozhraní evoluce EvolutionAPI (ukázka kódu 18). V této kapitole bude podrobně popsáno, jakým způsobem je vyřešena samotná implementace veřejného rozhraní evoluce.

Třída, která implementuje rozhraní evoluce se nazývá *MyEvolution*. Je vybavená konstruktorem [11], který má vstupní parametry služby výběru a křížení. Tím je zajištěna možnost konfiguračně měnit metodu použití těchto služeb.

[10] DTO (Data transfer object) - jednoduchý datový kontejner, který dokáže přenášet data mezi vrstvami a úrovněmi [26]

[11] Konstruktor - na první pohled vypadá, jako metoda která se jmenuje jako třída, ale konstruktor má úplně jiný úkol. Jediným úkolem je vytvořit instanci třídy. [27]

```
27      public MyEvolution(SelectionService selection, CrossoverService
            crossover) {
28          this.selectionService = selection;
29          this.crossoverService = crossover;
30      }
```

Ukázka kódu 19: Konstruktor třídy MyEvolution

Jak bylo popsáno výše, rozhraní nabízí dvě metody, u kterých je třeba dodržet pořadí volání. Jako první je nutné, aby byla zavolána metoda createInitialPopulation (ukázka kódu 20).

```
32      @Override
33      public Population createInitialPopulation(SetupEvolutionDTO
            setupProcess) {
34          // create init Population
35          Population initPopulation = new Population(setupProcess);
36          // create individuals with random chromosome
37          initPopulation.initPopulation();
38          return initPopulation;
39      }
```

Ukázka kódu 20: Metoda createInitialPopulation třídy MyEvolution

Jejím úkolem je vytvořit úplně první populaci, která má být vytvořena zcela náhodně. Nejprve je pomocí konstruktoru vytvořen samotný objekt populace. Dále je zavolána metoda, která postupně vytvoří za pomocí metody createRandomIndividual (ukázka kódu 5) nové náhodné jedince.

Druhá metoda v pořadí je createNextPopulation (ukázka kódu 21). Tato metoda je spíše abstraktem nad samotným vytvořením následujících populací. Je třeba provést několik kroků před vytvořením populace.

```
41      @Override
42      public Population createNextPopulation(Population
            previousPopulation) {
43          //urcite nejlepsiho a nejhorsiho z populace
44          preProcessNextPopulation(previousPopulation);
45
46          // upravime hodnoty fitness pro lepsi selekci od 0 do
                bestFitness-minFitness
47          double sumFitness = 0.0;
48          for (Individual individual :
                previousPopulation.getIndividuals()) {
49              double quality= individual.getQuality() -
```

```
                previousPopulation.getWorstIndividual().getQuality();
50              individual.setQuality(quality);
51              sumFitness += quality;
52          }
53          // create better population than before and replace
                previously population
54          return createNewPopulation(previousPopulation, sumFitness);
55      }
```

Ukázka kódu 21: Metoda createNextPopulation třídy MyEvolution

1. Určit nejlépe hodnoceného jedince
2. Určit nejhůře hodnoceného jedince
3. Provést trest

Penalty

V této knihovně evolučních algoritmů je zaveden systém trestů jedinců. V metodě doPenalty (ukázka kódu 22) je potrestán takový jedinec, který má více pozitivních genů, než je nastavený limit pro populaci. Trest je realizován tak, že je jedinci snížena kvalita na polovinu. Takový jedinec bude mít menší šanci, že se dostane do výběru. Systém trestů lze v konfiguraci vypnout.

```
87      private boolean doPenalty(Individual individual, Integer
            countOfPositiveGens) {
88          if (countOfPositiveGens <
                EvolutionUtils.getCountOfPositiveGens(individual.getChromosome()))
                {
89              individual.setQuality(individual.getQuality() / 2);
90              return true;
91          }
92          return false;
93      }
```

Ukázka kódu 22: Metoda doPenalty třídy MyEvolution

Metoda createNewPopulation

Metoda createNewPopulation je pro celou knihovnu nejdůležitější, neboť se v ní skrývá základní struktura evolučního algoritmu jako takového. Základem je cyklus z ukázky kódu 23, který postupně provede několik kroků:

1. Výběr jedinců
2. Křížení
3. Mutace

```
107     for (int iIndividual = 0; iIndividual < oldPopulation.getSize();
            iIndividual += 2) {
108         // random select two individuals
109         Individual individual1 =
                selectionService.randomSelectIndividual(oldPopulation.getIndividuals
                sumFitness);
110         Individual individual2 =
                selectionService.randomSelectIndividual(oldPopulation.getIndividuals
                sumFitness);
111         // crossover - create offspring
112         if (Math.random() < oldPopulation.getCrossRate()) {
113             Individual[] children =
                    crossoverService.crossover(individual1,
                    individual2);
114             newPopulation.getIndividuals().add(iIndividual,
                    children[0]);
115             newPopulation.getIndividuals().add(iIndividual + 1,
                    children[1]);
116
117             // mutation
118             for (Individual child : children) {
119                 if (Math.random() <
                        newPopulation.getMutationRate()) {
120                     doMutation(child);
121                 }
122             }
123         } else { // if not crossover then ..
124             newPopulation.getIndividuals().add(iIndividual,
                    individual1);
125             newPopulation.getIndividuals().add(iIndividual + 1,
                    individual2);
126         }
127     }
```

Ukázka kódu 23: Cyklus metody createNewPopulation třídy MyEvolution

Výběr a křížení je provedeno pomocí výše popsaných služeb, ale mutace je v této knihovně dostupná pouze jako privátní metoda doMutation (ukázka kódu 24).

Nejprve se zvolí několik náhodně vybraných pozic v chromozomu, na kterých se vzápětí invertuje původní gen za opačnou hodnotu genu. Starý chromozom je vyměněn.

```
147     private void doMutation(Individual child) {
148         int length = child.getChromosome().length();
149         String originalChromosome = child.getChromosome();
150         //get random positions where will be mutation
151         Set<Integer> generatedPositionToChange =
                child.getRandomPositions(length,
                EvolutionAPI.randomGenerator.nextInt(length));
152         //mutation on random positions
153         for (Integer position : generatedPositionToChange) {
154             char gen = originalChromosome.charAt(position);
155             if (gen == '1') {
156                 child.setChromosome(EvolutionUtils.changeCharInPosition(position,
                        '0', originalChromosome));
157             } else {
158                 child.setChromosome(EvolutionUtils.changeCharInPosition(position,
                        '1', originalChromosome));
159             }
160         }
161     }
```

Ukázka kódu 24: Metoda mutace třídy MyEvolution

V neposlední řadě metody createNewPopulation je uplatněno pravidlo „nejlepší musí žít". To znamená, že nejlépe hodnocený jedinec se automaticky dostane do nové populace.

V úplném závěru metody createNewPopulation je uplatnění sociální injekce. Pokud je v konfiguraci povolena sociální injekce, pak se zavolá metoda doItSocialInjection (ukázka kódu 25), která podle konfigurace vytvoří nové jedince s pseudonáhodně vytvořeným chromozomem.

```
137     /**
138      * Put new random individual into exist population. Social
            injection
139      */
140     private void doItSocialInjection(Population population) {
141         for (int i = 0; i < population.getSocialPressure(); i++) {
142             Individual randomIndividual =
                    population.createRandomIndividual();
143             population.getIndividuals().add(randomIndividual);
144         }
```

```
145        }
```

Ukázka kódu 25: Metoda doItSocialInjection třídy MyEvolution

3.4. Hlavní spouštěcí třída

Poslední třídou, která zde bude představena, je třída spouštěcí. Jejím hlavním úkolem je zajistit, aby uživatel mohl tuto konzolovou aplikaci samostatně spustit. V jazyce JAVA se k tomuto využívá speciální metoda, která se nazývá *main* (ukázka kódu 26). Tato statická metoda má za úkol zavolat konstruktor třídy, ve které se nachází.

```
147     public static void main(String[] args) {
148         new App();
149     }
```

Ukázka kódu 26: Metoda main třídy App

Konstruktor třídy App

Konstruktor této metody se postará o tyto důležitý kroky knihovny evolučních algoritmů:

1. Načte konfiguraci z externího souboru. Pokud tento soubor neexistuje, nebo pokud je vyplněný špatně, pak je brána v úvahu výchozí konfigurace. Celá konfigurace a její možnosti jsou popsány v kapitole konfigurace 3.5.

2. Pokusí se vytvořit instance služeb. Jak bylo zmíněno výše, knihovna evolučních algoritmů umožňuje konfiguračně měnit metody selekce, křížení a fitness funkce. Názvy služeb, které se mají pro dané spuštění použít, jsou součástí konfigurace. Opět platí, že pokud nejsou k dispozici, knihovna použije výchozí služby.

3. Samotný proces vytváření generací.

Proces vytvoření generací

Konstruktor spouštěcí třídy *App* obsahuje důležitou část evolučního algoritmu. Jedná se o samotné vytváření jednotlivých generací. Jak je vidět v ukázce kódu 27. Nejprve je splněna podmínka vytvoření úplně první populace, která obsahuje pseudonáhodně vytvořené jedince. Dále pokračuje postupné vytváření nových populací na základě předešlé populace. Ukončovací podmínkou je počet generací, které se mají vytvořit. Tato informace je součástí konfigurace.

```
50      EvolutionAPI evolution = new MyEvolution(selectionService,
           crossoverService);
51
52      //first population
53      Population population =
           evolution.createInitialPopulation(setupProcess);
54
55      for (int generationCount = 0; generationCount <
           setupProcess.getCountGeneration(); generationCount++) {
56          calculateQualityPopulation(population);
57
58          population = evolution.createNextPopulation(population);
59
60          // log it
61          String outMessage = "Generation " + generationCount + ",
               Fitness: " + population.getBestIndividual().getQuality()
               + " Chromosome:
               "+population.getBestIndividual().getChromosome();
62          App.logger.info(outMessage);
63      }
```

Ukázka kódu 27: Proces vytváření generací třídy App

V každém jednom kroku vytváření nové generace se nejprve vypočte fitness funkce pro stávající populaci. Tím se zhodnotí kvalita každého jedince, který existuje. Výpočet fitness funkce se provede zavoláním metody calculateQualityPopulationMethod (ukázka kódu 28), která postupně projde každého jedince a zavoláním služby pro fitness funkci zjistí jeho kvalitu. Získaná kvalita je přiřazena jedincovi.

```
143     private void calculateQualityPopulation(Population population) {
144
145         for (Individual individual : population.getIndividuals()) {
146             double newQuality =
                   defaultFitnessFunctionService.calculateFitnessFunction(individual);
147             individual.setQuality(newQuality);
148         }
149     }
```

Ukázka kódu 28: Výpočet fitness funkce v populaci

Pokud je stávající populace jedinců ohodnocena, je připraveno vytvoření nové populace, která vychází z předešlé populace. Výsledkem je nová populace.

Logování

Důležitým posledním krokem vytváření jednotlivých generací, je interakce s uživatelem. Jelikož se jedná pouze o konzolovou aplikace (knihovnu), výstupem pro uživatele je záznam v textové podobě.

Každá nově vytvořená generace vytvoří jeden záznam v logu (řádek), který obsahuje důležité informace o generaci:

1. Pořadové číslo generace
2. Výsledek fitness funkce nejlepšího jedince
3. Chromozom nejlepšího jedince

```
1   2016-07-19 23:20:12 INFO App:61 - Generation 0, Fitness:
      0.9609857775805771 Chromosome: 00000000001000000000
2   2016-07-19 23:20:12 INFO App:61 - Generation 1, Fitness:
      2.0835727975478004 Chromosome: 00001000010000000100
3   2016-07-19 23:20:12 INFO App:61 - Generation 2, Fitness:
      4.567456442456937 Chromosome: 00001000010001000101
```

Ukázka kódu 29: Příklad logování informací pro uživatele

3.5. Konfigurace

V této kapitole bude popsána konfigurace knihovny evolučních algoritmů, kterou může uživatel upravit. Jedná se o externí „properties soubor“[12].

Podmínkou načtení této konfigurace je, aby se tento konfigurační soubor nacházel ve stejné složce jako je knihovna evolučních algoritmů. Není však nezbytné, aby tento soubor existoval. Pokud při startu knihovny neexistuje konfigurační soubor ve stejné složce, pak knihovna použije výchozí hodnoty.

V ukázce 30 lze vidět celou výchozí konfiguraci, která má celkem 11 hodnot. Tato konfigurace je použita pokud knihovna pří spuštění nenalezne soubor s konfigurací, nebo pokud je soubor s chybnými hodnotami. Je důležité, aby levá strana (klíč) byla shodná také v externím konfiguračním souboru.

```
1 evolution.param.cross_rate=1
2 evolution.param.mutation_rate=0.5
3 evolution.param.population_size=30
```

[12] Soubor, který slouží k uložení konfiguračních parametrů aplikace, které lze měnit bez nutnosti kompilace aplikace. Hodnoty jsou uloženy ve tvaru klíč - hodnota. [28]

```
4  evolution.param.generation_count=500
5  evolution.param.positive_gens=2
6  evolution.param.length_chromosome=20
7  evolution.param.social_pressure=0
8  evolution.param.penalty=false
9
10 evolution.service.fitness=service.impl.fitness.MyFitnessFunction
11 evolution.service.selection=service.impl.selection.RouletteSelection
12 evolution.service.crossover=service.impl.crossover.OnePointCrossover
```

Ukázka kódu 30: Ukázka výchozí konfigurace.

1. parametr - míra křížení

Tento parametr uvádí pravděpodobnost, s jakou se provede křížení mezi jedinci. Jedná se o procenta převedená na desetinné číslo. Lze tedy zadat číslo v rozmezí od 0 do 1.

2. parametr - míra mutace

Druhým parametrem je pravděpodobnost, že jedinec projde mutací. Jde opět o procenta vyjádřená v desetinném čísle. Tím, že se mutace neprovede vždy, ale jen v určitém procentu, je vnesená určitá náhoda do celého algoritmu.

3. parametr - počet jedinců v populaci

Parametr udává, kolik bude existovat jedinců v jedné populaci.

4. parametr - počet generací

Tento parametr udává kolik celkem generací bude vytvořeno. Jedná se zároveň o ukončovací podmínku celého algoritmu. Cyklus celého algoritmu se provede tolikrát, kolikrát je uvedeno v tomto parametru.

5. parametr - počet pozitivních genů jedince

Zajímavý parametr, který udává hranici pozitivních genů v chromozomu. Knihovna evolučních algoritmů při generování nového pseudonáhodného chromozomu bere tento parametr jako horní hranici. Určí tedy počet pozitivních genů v rozmezí od jedné do této horní meze.

6. parametr - délka chromozomu

Parametr udává délku binárního řetězce (chromozomu) jedince. Platí zde, že čím delší chromozom má jedinec, tím je zpracování celého algoritmu náročnější.

7. parametr - sociální tlak

Tímto parametrem aktivujeme tzv. sociální injekci. Metoda sociální injekce byla popsána výše v kapitole Sociální injekce 2.4.7. V podstatě se jedná o počet nově (náhodně) založených jedinců, kteří budou ve chvíli sociální injekce vložení do stávající populace. Dojde k určitému oživení populace, jejíž vývoj by se mohl udávat špatným směrem. Tato funkčnost lze deaktivovat způsobem, že se zvolí hodnota parametru 0.

8. parametr - trest

Jedná se to parametr typu boolean. To znamená, že může nabývat pouze hodnot „TRUE" a „FALSE". Pokud je tento parametr zapnutý, pak je každý jedinec trestán, pokud nesplní podmínky. Podrobněji je tento typ trestu popsán v kapitole Implementace rozhraní evoluce 3.3.2.

9 - 11. parametr - služby

Posledními parametry jsou jména implementací použitých služeb pro selekci, křížení a fitness funkci. Každé jméno musí být ve speciálním tvaru, který tečkovou notací znázorňuje cestu k dané implementaci. Pokud uživatel omylem zadá nesprávné jméno některé ze služeb, knihovna použije výchozí službu.

3.6. Testování knihovny

Ke každému vývoji libovolné knihovny patří otestování správné funkčnosti dané knihovny. Pro knihovnu evolučních algoritmů, jejíž vytvoření je cílem této diplomové práce, byly provede testy. Tato kapitola bude reprezentovat výsledky, které v testech vyšly.

Evoluční algoritmus je znám svou variabilitou v nastavení parametrů. V tabulce 2 je přehled parametrů, které byly po dobu testování pevně zvoleny.

Pro testování byla použita jednoduchá fitness funkce, která je popsaná výše v kapitole 3.2.5. Tato fitness funkce určuje kvalitu jedince podle počtu výskytu logických „1" v chromozomu.

V testu byly mezi sebou srovnány jednotlivé implementované metody selekce. Na základě těchto metod selekce byly vytvořeny scénáře pro testovaní. Detailní výpis všech

Proměnná	**Hodnota**
Pravděpodobnost křížení (cross_rate)	0.8
Pravděpodobnost mutace (mutation_rate)	0.5
Počet generací (generation_count)	500
Maximální počet „1“ genů (positive_gens)	10
Délka chromozomu (length_chromosome)	100
Trest (penalty)	FALSE
Fitness funkce (fintess)	MyFitnessFunction
Metoda křížení (crossover)	OnePointCrossover

Tabulka 2.: Pevně nastavené parametry pro testování

Scénář	**Metoda selekce**	**Počet členů**	**Sociální injekce**	Graf
Scénář 1	ProbabilityLineSelection	50	vypnuta	grafA1
Scénář 2	RouletteSelection	50	vypnuta	grafA2
Scénář 3	TournamentSelection	50	vypnuta	grafA3
Scénář 4	ProbabilityLineSelection	50	10	grafA4
Scénář 5	RouletteSelection	50	10	grafA5
Scénář 6	TournamentSelection	50	10	grafA6
Scénář 7	ProbabilityLineSelection	100	vypnuta	grafA7
Scénář 8	RouletteSelection	100	vypnuta	grafA8
Scénář 9	TournamentSelection	100	vypnuta	grafA9
Scénář 10	ProbabilityLineSelection	100	10	grafA10
Scénář 11	RouletteSelection	100	10	grafA11
Scénář 12	TournamentSelection	100	10	grafA12

Tabulka 3.: Provedené scénáře při testování knihovny

scénářů je znázorněn v tabulce 3. Krom metod selekce byly měněny i parametry „Počet členů v populace“ a „Sociální injekce“.

Poslední sloupec tabulky 3 odkazuje na výsledný graf po spuštění knihovny evolučních algoritmů s danou konfigurací.

Graf reprezentuje vývoj generací v čase. Na svislé ose je znázorněna kvalita jedinců a na vodorovné ose index generace. Z grafu je vidět postupný vývoj od vytvoření první pseudonáhodné populace.

4. Shrnutí výsledků

Cílem této práce bylo implementovat knihovnu evolučních algoritmů. Knihovna evolučních algoritmů byla úspěšně implementována a testována. Z výsledků, které byly reprezentovány v kapitole 3.6 je zřejmé, že knihovna evolučních algoritmů funguje správně. Přímka kvality jedinců má správný vzrůstající charakter.

Obecně lze říci, že všechny testované scénáře mají podobný charakter. Zhruba do dvousté generace je velmi prudký nárůst kvality původní populace.

Nejlépe vyšla konfigurace scénáře 9. a 12. (viz. tabulka scénářů 3). Jak ukazuje graf A9 a graf A12, kvalita populace se vyšplhala k číslu 90. Oba scénáře mají společnou metodu selekce - ruleta a počet obyvatel 100. Zajímavé je, že při snížení členů v populaci na polovinu (scénář 2, scénář 5) dopadly tyto testy naopak špatně. Kvalita populace se pohybuje okolo 70. U ostatních metod se kvalita pohybovala kolem 80.

5. Závěry a doporučení

Na závěr lze říci, že evoluční algoritmy se stávají čím dál více oblíbenou metodou pro řešení problémů, u kterých nelze dosáhnout diskrétního výsledku. Tyto problémy se nazývají optimalizační problémy a hledáme u nich optimální výsledek, tedy výsledek, který je nejblíže tomu ideálnímu. Evoluční algoritmy jsou pro řešení optimalizačních problémů ideální.

V biomedicíně existuje spousta optimalizačních problémů, které mají vysokou míru neznámých proměnných a zároveň nedostatek dat. Tato diplomová práce měla za cíl implementaci evolučního algoritmu, který by mohl pomoci řešit tyto specifické problémy a pomoci tak dalšímu výzkumu.

Cíl se povedl splnit a vzniklá implementace knihovny evolučních algoritmů je tak k dispozici biomedicínskému výzkumu. Testování prokázalo, že knihovna funguje správně.

Další směry, kam by mohl vývoj této knihovny pokračovat, je rozhodně přidání fitness funkcí. Jak bylo v úvodu zmíněno, jako fitness funkce by mohla sloužit neuronová síť.

Dále by bylo zajímavé vytvořit grafické uživatelské prostředí, které by například mohlo podporovat funkci automatických testů. Uživatel by si nastavil intervaly rozsahu parametrů a program by sám pouštěl algoritmus s různými parametry.

Seznam obrázků

1 Struktura prohledávacího algoritmu. 10
2 Schéma obecného evolučního algoritmu. 16
3 Proces dvoubodového křížení jedince. 17
4 Proces mutace jedince. 18
5 Ukázka vývojového nástroje Intellij IDEA. 24
6 Rozložení balíčku knihovny. 25
7 Znázornění jednobodového křížení. [24] 35

Seznam grafů

A1 Výsledky testu - scénář 1 I
A2 Výsledky testu - scénář 2 II
A3 Výsledky testu - scénář 3 II
A4 Výsledky testu - scénář 4 III
A5 Výsledky testu - scénář 5 III
A6 Výsledky testu - scénář 6 IV
A7 Výsledky testu - scénář 7 IV
A8 Výsledky testu - scénář 8 V
A9 Výsledky testu - scénář 9 V
A10 Výsledky testu - scénář 10 VI
A11 Výsledky testu - scénář 11 VI
A12 Výsledky testu - scénář 12 VII

Seznam tabulek

1 Biologie vs. optimalizace 14

2 Pevně nastavené parametry pro testování 50
3 Provedené scénáře při testování knihovny 50

Seznam ukázek kódu

1 Atributy třídy Jedinec . . . 26
2 Metoda generateChromosome . . . 27
3 Metoda getRandomPositions . . . 28
4 Atributy třídy populace . . . 28
5 Metoda createRandomIndividual . . . 30
6 Rozhraní CrossoverService . . . 30
7 Rozhraní FitnessFunctionService . . . 31
8 Rozhraní SelectionService . . . 31
9 Metoda crossover třídy BinaryMaskCrossover . . . 32
10 Metoda generateRandomBinaryMask třídy BinaryMaskCrossover . . . 33
11 Metoda applyMask třídy BinaryMaskCrossover . . . 33
12 Metoda crossover třídy OnePointCrossover . . . 34
13 Metoda generateQualityLine třídy ProbabilityLineSelection . . . 36
14 Metoda randomSelectIndividual třídy ProbabilityLineSelection . . . 37
15 Metoda randomSelectIndividual třídy TournamentSelection . . . 37
16 Metoda randomSelectIndividual třídy RouletteSelection . . . 38
17 Metoda calculateFitnessFunction třídy MyFitnessFunction . . . 39
18 Veřejné rozhraní Evolučního algoritmu . . . 40
19 Konstruktor třídy MyEvolution . . . 41
20 Metoda createInitialPopulation třídy MyEvolution . . . 41
21 Metoda createNextPopulation třídy MyEvolution . . . 41
22 Metoda doPenalty třídy MyEvolution . . . 42
23 Cyklus metody createNewPopulation třídy MyEvolution . . . 43
24 Metoda mutace třídy MyEvolution . . . 44
25 Metoda doItSocialInjection třídy MyEvolution . . . 44
26 Metoda main třídy App . . . 45
27 Proces vytváření generací třídy App . . . 46
28 Výpočet fitness funkce v populaci . . . 46
29 Příklad logování informací pro uživatele . . . 47
30 Ukázka výchozí konfigurace. . . . 47

Literatura

[1] "05_rozbory_presnosti:0508_globalni_optimalizacni_metody | inggeo - portál inženýrské geodézie." http://inggeo.fsv.cvut.cz/wiki/doku.php?id=05_rozbory_presnosti:0508_globalni_optimalizacni_metody. (Accessed on 08/02/2016).

[2] I. Zelinka, Z. Oplatková, P. Ošmera, M. Šeda, and F. Včelař, *Evoluční výpočetní techniky*. BEN, 2008.

[3] J. TVRDÍK, "Evoluční algoritmy," *Skripta, Přírodovědecká fakulta, Ostravská univerzita*, 2004.

[4] B. T. Mihulka, *Univerzální darwinizmus jako vědecká teorie*. PhD thesis, Masarykova univerzita, Filozofická fakulta, 2015.

[5] "Evolution - wikipedia, the free encyclopedia." https://en.wikipedia.org/wiki/Evolution. (Accessed on 08/02/2016).

[6] M. Eigen, *Ingo Rechenberg Evolutionsstrategie Optimierung technischer Systeme nach Prinzipien der biologishen Evolution*. mit einem Nachwort von Manfred Eigen, Friedrich Frommann Verlag, Struttgart-Bad Cannstatt, 1973.

[7] J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*, vol. 1. MIT press, 1992.

[8] L. J. Fogel, *Artificial Intelligence Through Simulated Evolution.[By] Lawrence J. Fogel... Alvin J. Owens... Michael J. Walsh*. John Wiley & Sons, 1966.

[9] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press, 1975.

[10] J. Pavlovič, "Multikriteriální hybridní evoluční algoritmy pro výběr a optimalizaci dekontaminačních technologií," *Brno: Masarykova univerzita*, 2006.

[11] J. Hynek, *Genetické algoritmy a genetické programování*. 2008.

[12] "Evoluční algoritmy pro řešení globálních optimalizačních problémů, author=DRAGON, PRÁCE ONDŘEJ,"

[13] M. Mach, “Evolučné algoritmy,”

[14] E. Volná, “Evoluční algoritmy a neuronové sítě,” 2012.

[15] T. Král, “Knihovna evolučních optimalizačních algoritmů v prostředí java,” 2011.

[16] V. CHUROVÁ, *Optimalizační algoritmy inspirované živou přírodou*. PhD thesis, Masarykova univerzita, Přírodovědecká fakulta, 2015.

[17] L. Bajer, “Urychlení evolučních algoritmů pomocí směsí rozdělení pravděpodobnosti,”

[18] “Application programming interface - wikipedia, the free encyclopedia.” `https://en.wikipedia.org/wiki/Application_programming_interface`. (Accessed on 08/08/2016).

[19] V. Hordějčuk, “Utility (knihovna) - vojtěch hordějčuk.” `http://voho.cz/wiki/utility/`. (Accessed on 06/05/2016).

[20] T. Pitner, “Základní pojmy oop. třída, objekt, jeho vlastnosti. metody, proměnné. konstruktory..” `http://is.muni.cz/el/1433/podzim2006/PB162/um/02/printable.html#minimodule33`, 9 2006. (Accessed on 06/06/2016).

[21] V. portál Háka Software, “Java - rozhraní, polymorfismus, výjimky.” `http://portal.haka-software.cz/index.php?option=com_content&view=article&id=9:java-rozhrani-polymorfismus-vyjimky&catid=3:programovani-java&Itemid=13`, 2016. (Accessed on 06/14/2016).

[22] V. Hordějčuk, “Binární čísla a bitové operace - vojtěch hordějčuk.” `http://voho.cz/wiki/bit/`. (Accessed on 06/21/2016).

[23] Tastyfish, “Generování (pseudo)náhodných čísel.” `http://www.itnetwork.cz/algoritmy/matematicke/algoritmus-generovani-pseudo-nahodnych-cisel/`. (Accessed on 06/24/2016).

[24] “Epm model.” `http://dcm.ffclrp.usp.br/epmmodel/`. (Accessed on 06/24/2016).

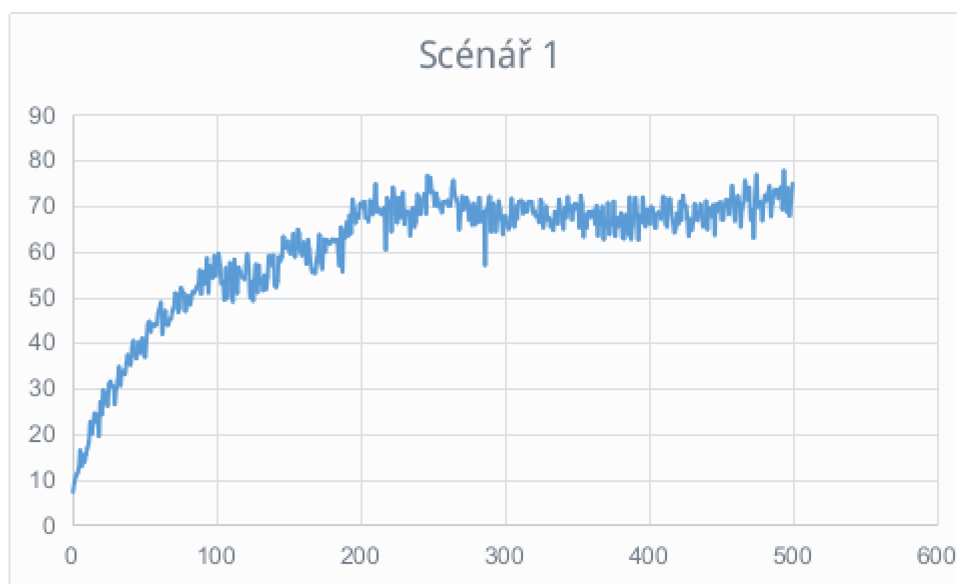
[25] Oracle, “Map (java platform se 6).” `http://docs.oracle.com/javase/6/docs/api/java/util/Map.html`. (Accessed on 06/27/2016).

[26] “Value object vs. data transfer object (vo vs. dto) : Adam bien’s weblog.” `http://www.adam-bien.com/roller/abien/entry/value_object_vs_data_transfer`. (Accessed on 07/06/2016).

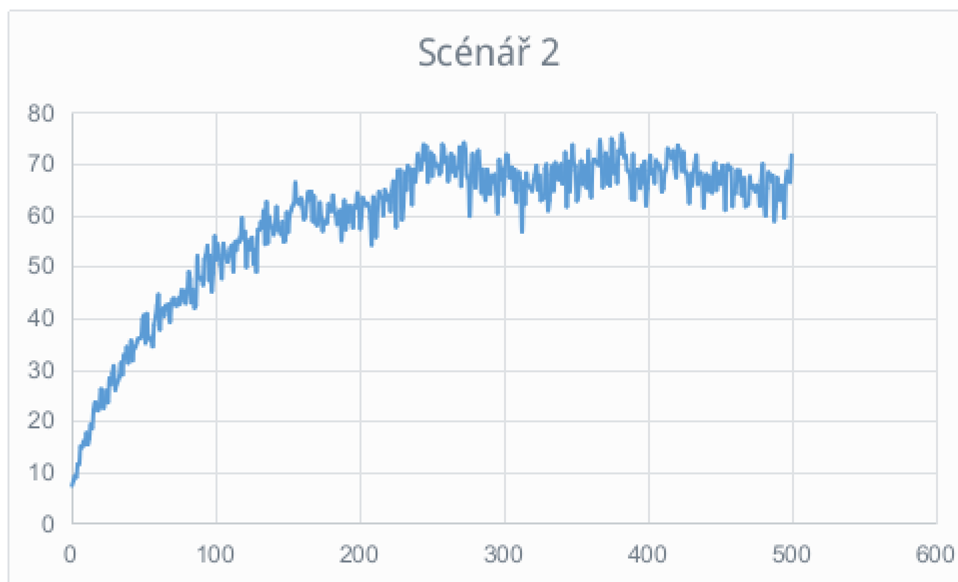
[27] "Understanding constructors | javaworld." http://www.javaworld.com/article/2076204/core-java/understanding-constructors.html. (Accessed on 07/11/2016).

[28] ".properties - wikipedia, the free encyclopedia." https://en.wikipedia.org/wiki/.properties. (Accessed on 07/24/2016).

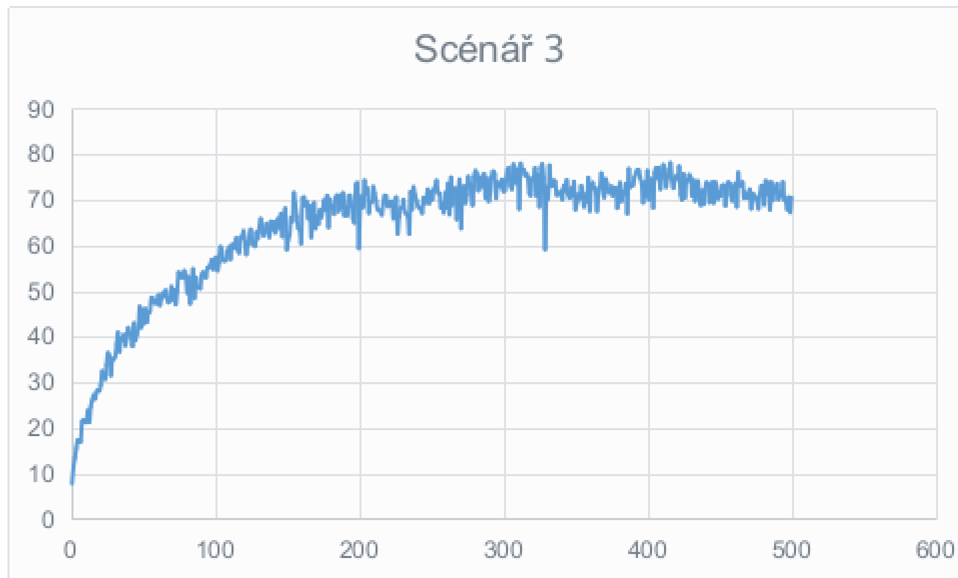
A. Výsledky testů



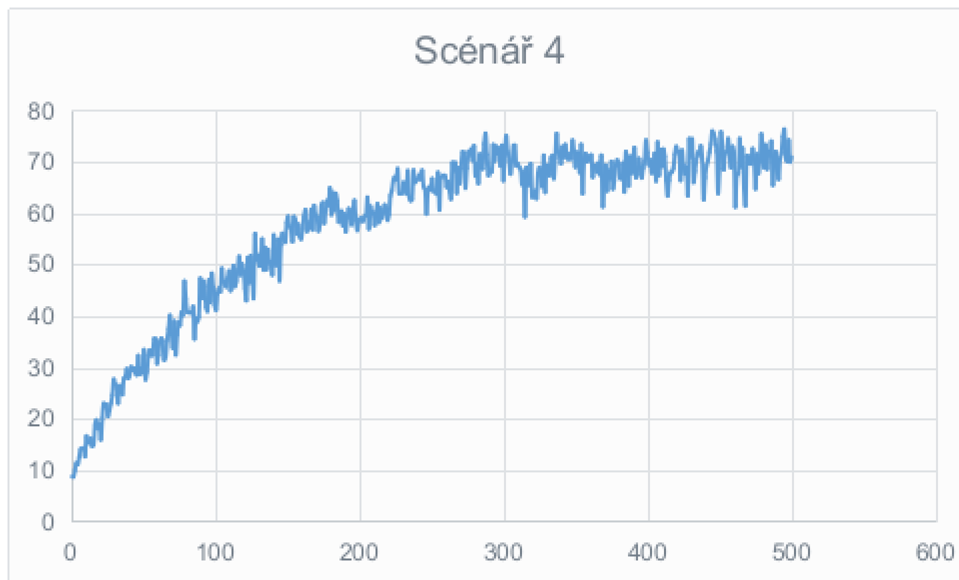
Graf A1: Výsledky testu - scénář 1



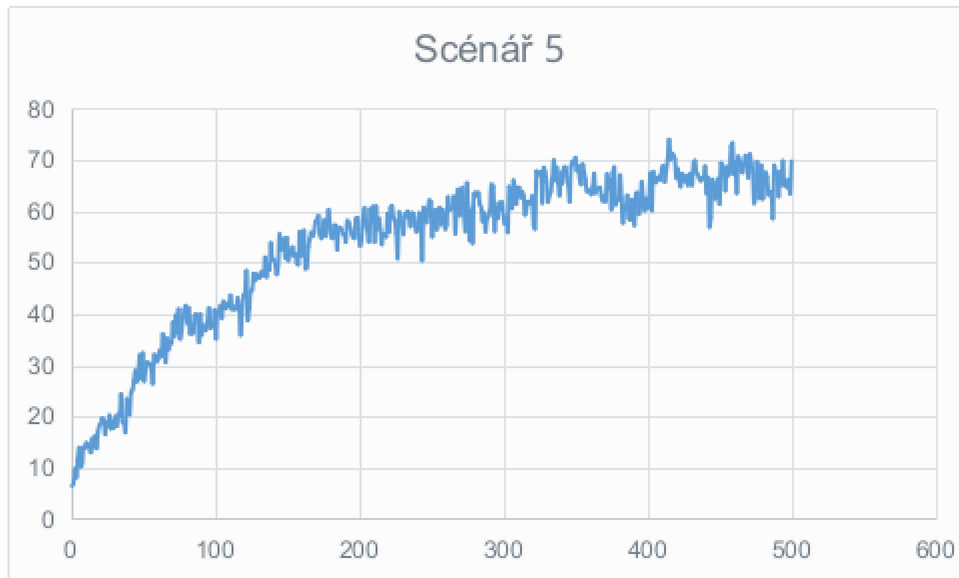
Graf A2: Výsledky testu - scénář 2



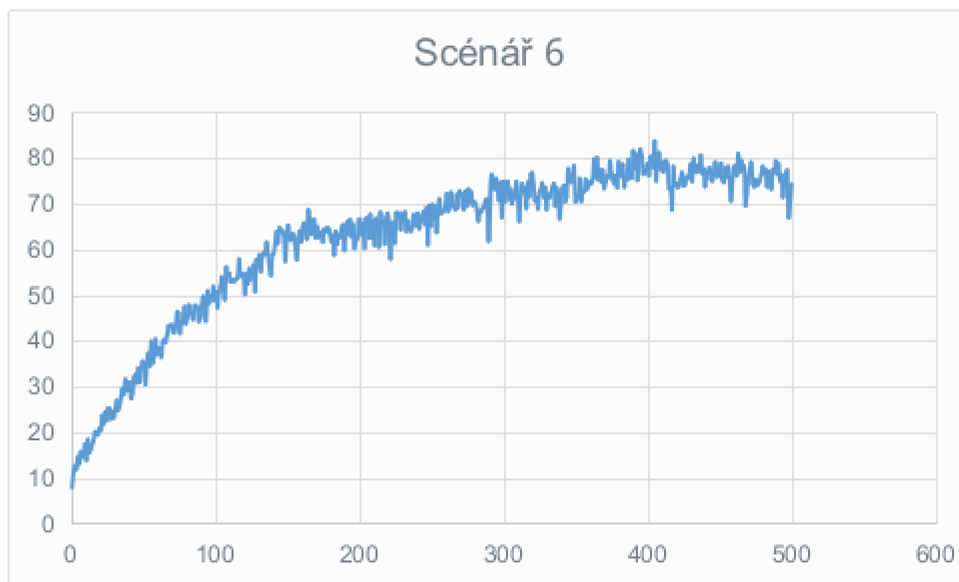
Graf A3: Výsledky testu - scénář 3



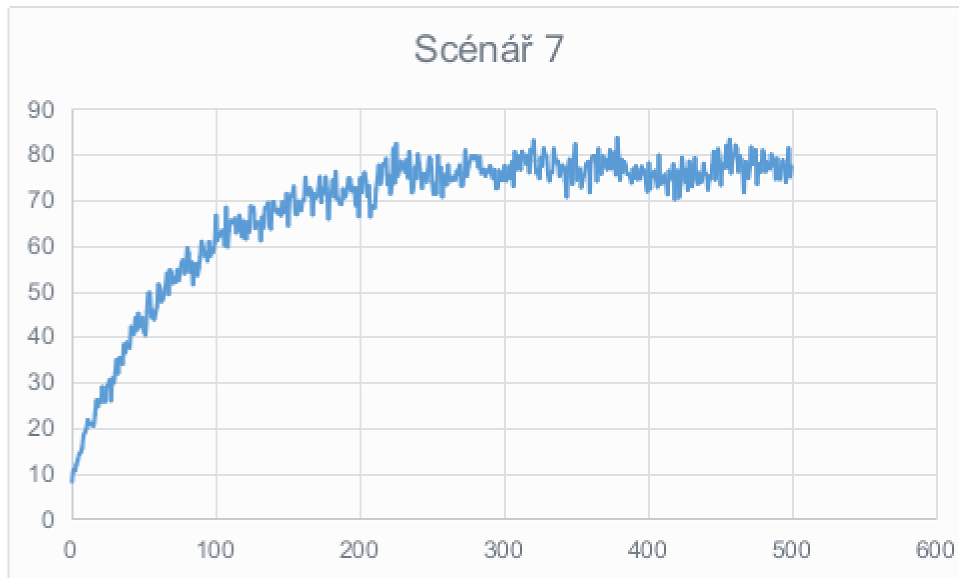
Graf A4: Výsledky testu - scénář 4



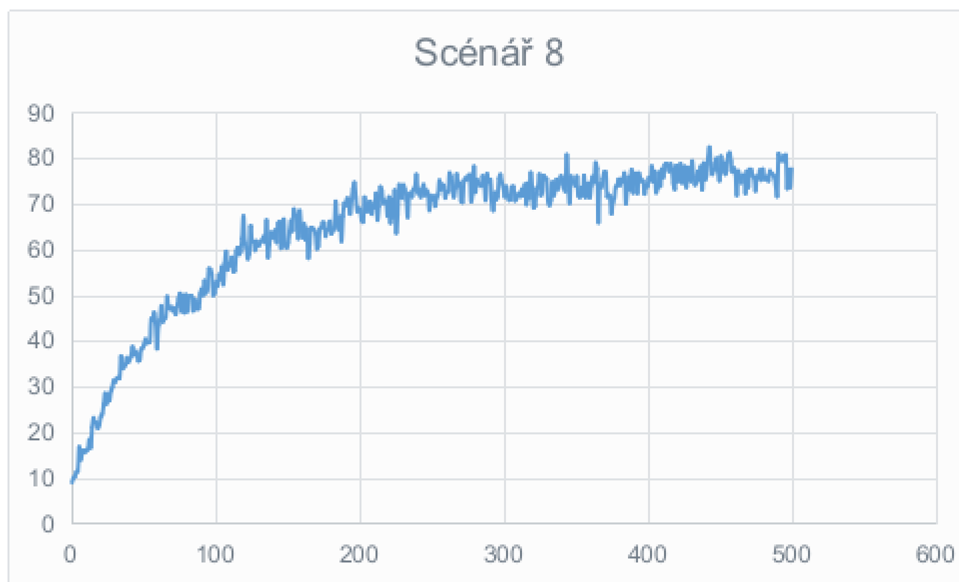
Graf A5: Výsledky testu - scénář 5



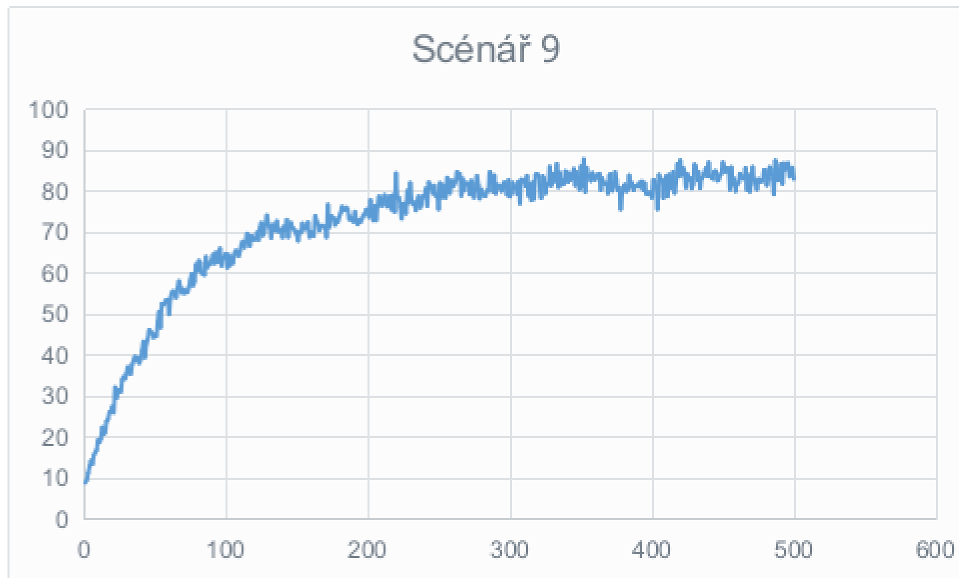
Graf A6: Výsledky testu - scénář 6



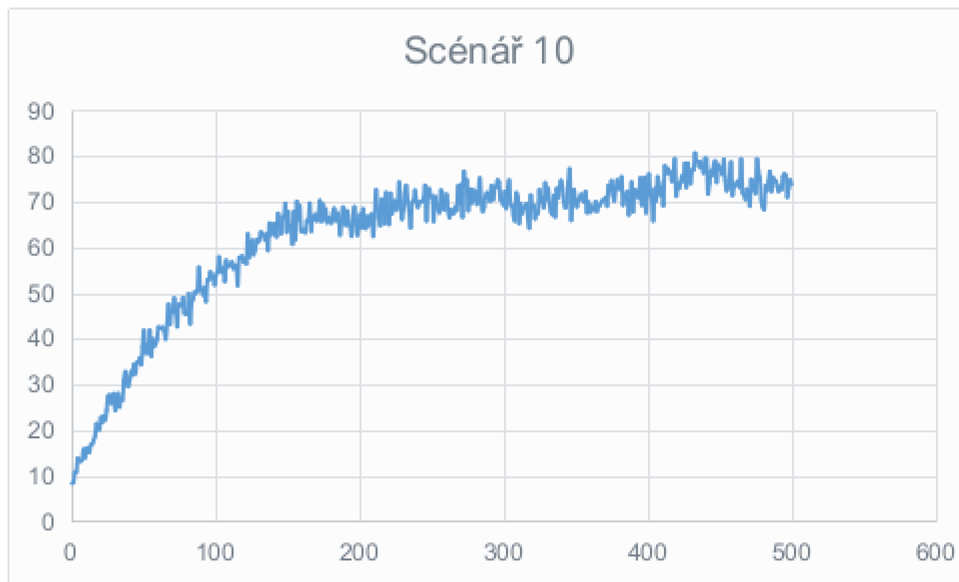
Graf A7: Výsledky testu - scénář 7



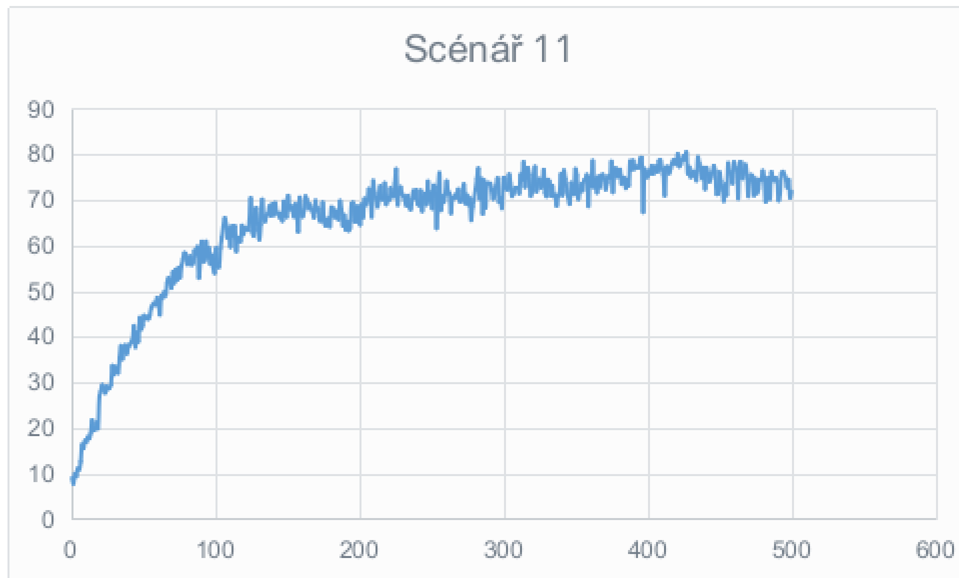
Graf A8: Výsledky testu - scénář 8



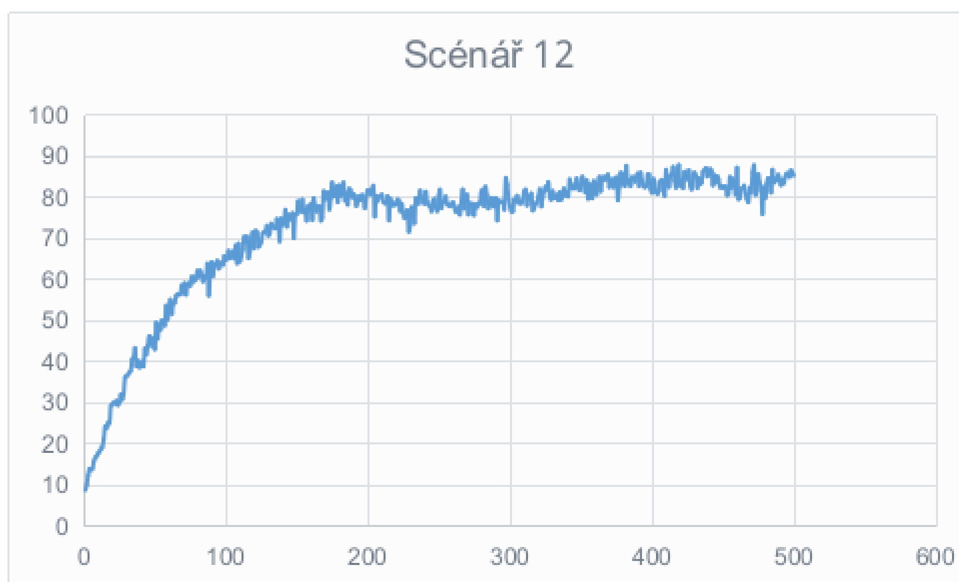
Graf A9: Výsledky testu - scénář 9



Graf A10: Výsledky testu - scénář 10



Graf A11: Výsledky testu - scénář 11



Graf A12: Výsledky testu - scénář 12

B. Obsah DVD

Součástí práce je přiložené DVD, které obsahuje

- Zdrojové soubory knihovny evolučních algoritmů
- Spustitelný skript „evoluce.bat"
- Tuto diplomovou práci

B.1. Spustitelný skript

Na přiloženém DVD se nachází soubor se jménem „evoluce.bat". Jedná se o spustitelný soubor, který zajistí spuštění knihovny evolučních algoritmů v příkazové řádce, který použije přiložený konfigurační soubor.

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2015/2016

Studijní program: Aplikovaná informatika
Forma: Prezenční
Obor/komb.: Aplikovaná informatika (ai2-p)

Podklad pro zadání DIPLOMOVÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Rezner Lukáš	Plotišťská 39, Hradec Králové - Věkoše	I14747

TÉMA ČESKY:

Evoluční algoritmy v biomedicínském výzkumu

TÉMA ANGLICKY:

Evolutionary algorithms in biomedical research

VEDOUCÍ PRÁCE:

Ing. Karel Mls, Ph.D. - KIT

ZÁSADY PRO VYPRACOVÁNÍ:

Cíl práce: Aplikace evolučních algoritmů při zpracování dat v biomedicínském výzkumu.
Osnova práce:
Úvod, Cíl, Metodika zpracování, Teoretická část, Praktická část, Návrh konkrétního algoritmu, Testování na experimentálních datech, Statistické vyhodnocení výsledků, Závěr, Literatura, Přílohy

SEZNAM DOPORUČENÉ LITERATURY:

-WANG, Ling, et al. Feature selection based on meta-heuristics for biomedicine. Optimization Methods and Software, 2014, 29.4: 703-719.
-HOLMES, John H. Methods and applications of evolutionary computation in biomedicine. Journal of biomedical informatics, 2014, 49.C: 11-15.
-KOZA, John R. Survey of genetic algorithms and genetic programming. In: Wescon Conference Record. WESTERN PERIODICALS COMPANY, 1995. p. 589-594.
-SUKUMAR, N.; PRABHU, Ganesh; SAHA, Pinaki. Applications of Genetic Algorithms in QSAR/QSPR Modeling. In: Applications of Metaheuristics in Process Engineering. Springer International Publishing, 2014. p. 315-324.
-ZANDKARIMI, Majid, et al. Prediction of Pharmacokinetic Parameters Using a Genetic Algorithm Combined with an Artificial Neural Network for a Series of Alkaloid Drugs. Scientia pharmaceutica, 2014, 82.1: 53.

Podpis studenta: .... **Datum:** 14.10.2015

Podpis vedoucího práce: .... **Datum:** 14.10.2015