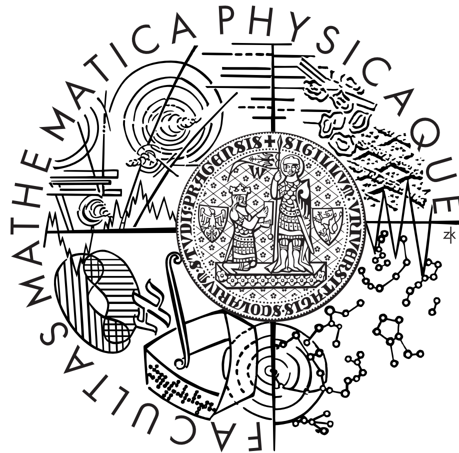
Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Miroslav Šiagi

Podobnost proteinových struktur s využitím genetického programování

Katedra softwarového inženýrství

Vedoucí bakalářské práce: RNDr. David Hoksza, PhD.

Studijní program: Informatika

Studijní obor: Programování

Praha 2012

Touto cestou chcem vyjadriť vďaku všetkým, ktorí mi počas obdobia písania tejto práce pomohli a ma podporili. Obzvlášť sa chcem poďakovať môjmu vedúcemu práce RNDr. Davidovi Hokszovi, PhD., za odborné rady, usmerňovanie, ochotu a príjemný prístup. Vďaka patrí aj mojim rodičom za trpezlivosť a podporu. Rovnako ďakujem aj priateľom a známym za ich podporu, názory i odborné rady.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V Praze dne 6. prosince 2012 Miroslav Šiagi

Název práce: Podobnost proteinových struktur s využitím genetického programování

Autor: Miroslav Šiagi

Katedra: Katedra softwarového inženýrství

Vedoucí bakalářské práce: RNDr. David Hoksza, PhD.

Abstrakt: Dôležitý aspekt bioinformatiky, ktorému sa práca venuje, je porovnávanie proteínových štruktúr. Vzhľadom na exponenciálny nárast databáz proteínových štruktúr za posledné roky bol potrebný vývoj efektívnejších metód. Riešenie daného problému nám ponúkajú evolučné výpočtové techniky. Predstavujú významný princíp, kedy počítač môže vyriešiť daný problém bez toho, aby ho človek preň explicitne naprogramoval. Zameriame sa na jednu paradigmu evolučných výpočtových techník — genetické programovanie. Vďaka stromovej reprezentácii má táto paradigma oproti zvyšným výhodu. Cieľom práce je preskúmanie možností využitia genetického programovania pri porovnávaní proteínových štruktúr. Navrhneme novú metódu — nazvanú ProSSiGen. Jej výsledky preukázali nedostatočnú presnosť klasifikácie a tiež to, že tento evolučný prístup rozhodne netreba zatratiť, len ho ďalej rozširovať a testovať. Tým budeme môcť vyvodiť záver, či je genetické programovanie pre túto úlohu vhodné.

Klíčová slova: proteinová struktura, podobnost, genetické programování

Title: Protein Structure Similarity Using Genetic Programming

Author: Miroslav Šiagi

Department: Department of Software Engineering

Supervisor: RNDr. David Hoksza, PhD.

Abstract: The thesis deals with the protein structure similarity problem which is an important aspect of bioinformatics. Due to exponential growth of protein structures in databases, the development of more effective methods is required. Principles of evolutionary computation offer a way to solve the similarity problem. We focus on one of the evolutionary paradigms — genetic programming. The main advantage of genetic programming is a tree representation. We propose new method called ProSSiGen using genetic programming. ProSSiGen is evaluated by automatic protein classification. Obtained results signify that the efficiency of our method is insufficient. Regardless of the inefficiency, there are many reasons to continue to research. One of the reasons is the capability of genetic programming.

Keywords: protein structure, similarity, genetic programming

Obsah

1 Úvod **3**
1.1 Proteíny . . . 3
1.1.1 Štruktúra bielkovín . . . 4
1.1.2 Centrálna dogma molekulárnej biológie . . . 4
1.1.3 Databázy a klasifikácia proteínových štruktúr . . . 6
1.1.4 Význam a uplatnenie podobnosti proteínov . . . 7
1.2 Evolučné výpočtové techniky . . . 7
1.2.1 Všeobecný princíp evolučných algoritmov . . . 8
1.3 Motivácia práce . . . 9

2 Genetické programovanie **11**
2.1 Stromová reprezentácia . . . 11
2.2 Počiatočná populácia . . . 12
2.3 *Fitness* funkcia . . . 13
2.4 Primárne operácie pre modifikovanie štruktúr . . . 13
2.4.1 Reprodukcia . . . 13
2.4.2 Kríženie . . . 14
2.5 Sekundárne operácie . . . 15
2.6 Terminácia . . . 16

3 Porovnávanie proteínov **17**
3.1 Sekvenčné zarovnávanie . . . 17
3.1.1 Globálne sekvenčné zarovnanie . . . 17
3.1.2 Lokálne sekvenčné zarovnanie . . . 18
3.2 Štrukturálne zarovnávanie . . . 18
3.3 Metódy pre porovnávanie bielkovín . . . 19
3.4 Existujúce riešenia využívajúce evolučné výpočtové techniky . . . 20
3.4.1 KENOBI . . . 20
3.4.2 K2 . . . 22
3.5 Analýza využitia GP pri porovnávaní . . . 22

4 Metóda ProSSiGen **24**
4.1 Návrh objektov . . . 24
4.1.1 Základné objekty . . . 24
4.1.2 Dodatočné objekty . . . 26
4.1.3 Stromové objekty . . . 26
4.2 Proces . . . 27
4.2.1 Vstupy a inicializácia . . . 28
4.2.2 Globálne sekvenčné zarovnanie (GS) . . . 28

4.2.3 Prevod PAP na APTree 29
4.3 Genetické programovanie v ProSSiGen 32
4.3.1 *Fitness* a selekčná funkcia 33
4.3.2 Kríženie objektov APTree 33
4.3.3 Mutácia nových jedincov 35
4.3.4 Terminačné kritérium 36

5 Experimenty 37
5.1 Metodika testovania 37
5.2 Presnosť klasifikácie 37
5.3 Analýza výsledkov 38
5.4 Vplyv parametrov genetického porovnania 39
5.5 Návrhy rozšírenia 40

Záver 42

Zoznam použitej literatúry 44

A Obsah priloženého CD 49

B Ilustračné obrázky 51

C Užívateľská dokumentácia 54
C.1 Požiadavky a spustenie programu 54
C.2 Testovacia časť 56

Kapitola 1

Úvod

Proteíny, tiež bielkoviny, predstavujú významnú súčasť organickej hmoty. Podstatná časť buniek všetkých organizmov je tvorená práve nimi. Utvárajú jednak organickú štruktúru, jednak sa podieľajú na plnení rôznych funkcií a na kľúčových procesoch v bunke. Budujú bunečnú štruktúru, trávia výživné látky, plnia metabolické funkcie, sprostredkovávajú tok informácií v rámci bunky a medzi bunečnými časťami. Sú súčasťou centrálnej dogmy molekulárnej biológie.

Za posledných 15 rokoch sa počet zaznamenaných bielkovín v proteínových databázach rapídne zvyšoval. Nastával problém, ako novú bielkovinu efektívne klasifikovať a pre tento účel sa využíva podobnosť proteínových štruktúr.

Položili sme si otázku, ako v procese získavania podobnosti proteínových štruktúr možno využiť istý druh evolučných výpočtových techník — genetické programovanie . Evolučné výpočtové techniky totiž predstavujú silný nástroj pre riešenie rôznych a častokrát algoritmicky náročných úloh, a navyše predstavujú veľmi významný princíp, ako počítačový program môže riešiť problém bez toho, aby sme ho preň explicitne naprogramovali. Samotné genetické programovanie nesie veľkú výhodu v stromovej reprezentácii.

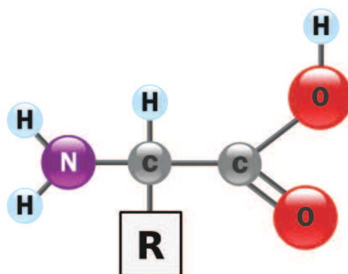
Cieľom práce je preskúmať možnosti využitia algoritmov genetického programovania pri určovaní podobnosti proteínových štruktúr a pre tento účel navrhnúť konkrétnu metódu. Program implementujúci danú metódu bude klasifikovať sadu bielkovín na základe sady už klasifikovaných bielkovín. Nakoniec určíme presnosť navrhnutej metódy. Analýza získaných výsledkov bude znamenať dôležitý poznatok využitia genetického programovania pri využití podobnosti proteínových štruktúr a pomôže nám určiť spôsob, ako genetické programovanie alebo aj celú skupinu evolučných výpočtových techník pri porovnávaní lepšie využiť.

1.1 Proteíny

Základný stavebný prvok bielkovín je aminokyselina, čo je molekula zložená z aminovej (–NH_2) a karboxylovej (–COOH) funkčnej skupiny, na ktoré je viazaný α-uhlík, na ktorom je tiež naviazaný atóm vodíka a postranný reťazec [1]. Aminokyselina v rámci postranného reťazca sa tiež nazýva ako tzv. rezíduum, ktorého „stredový" bod sa považuje α-uhlík. Rozoznávame 20 štandardných aminokyselín (a dve netradičné).

Postranný reťazec je určený genetickým kódom, ktorý reprezentuje súbor pravidiel, podľa ktorých sa genetická informácia uložená v DNA, resp. RNA,

prevádza na primárnu štruktúru proteínov.



Obr. 1.1: Všeobecná štruktúra alfa aminokyselín. Zdroj: `www.wikipedia.org`

Názvy aminokyselín sa zväčša používajú v skrátenej forme v podobe troch písmen. Avšak 20 bežných aminokyselín môžeme reprezentovať aj jedným z 26 písmen latinskej abecedy, čo šetrí miesto v počítačových databázach. Uveďme si príklad: troj-písmenkový názov *Fenylalanínu* je *Phe*, jedno-písmenkový je *F* [2].

1.1.1 Štruktúra bielkovín

Funkčné vlastnosti bielkovín sa odvíjajú od ich trojdimenzionálnej štruktúry. Schopnosť bielkovín vykonávať svoju funkciu v toľkých rôznych rolách vyplýva práve z nesmierneho množstva štrukturálnych variácií. Štrukturálna variácia je výsledkom enormného množstva kombinácií, ktoré môžeme dostať spájaním dvadsiatich aminokyselín [3].

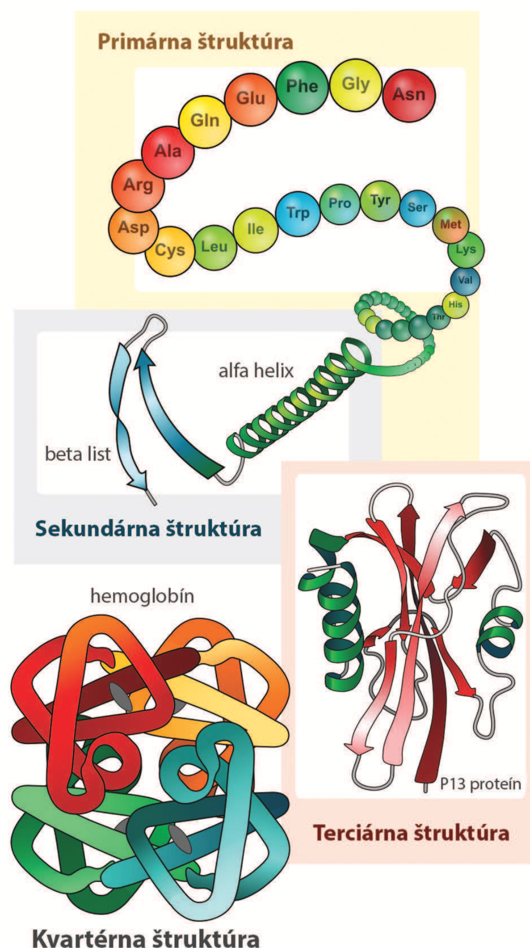
Štruktúru bielkovín vnímame zo štyroch rôznych aspektov a rozoznávame:

- primárnu štruktúru, ktorá pozostáva zo sekvencie aminokyselín pospájaných polypeptidovými väzbami,
- sekundárnu štruktúru, pri ktorej rozoznávame sekundárne štruktúry ako alfa-helix (pravotočivá závitnica) a beta-štruktúra (skladaný list beta),
- terciárnu štruktúru, ktorá je trojrozmerným usporiadaním celého polypeptidového reťazca,
- kvartérnu štruktúru, ktorá predstavuje usporiadanie niekoľkých polypeptidových reťazcoch, ak bielkovina viac týchto reťazcov obsahuje.

1.1.2 Centrálna dogma molekulárnej biológie

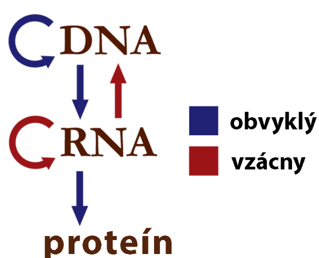
Gény, ktoré kódujú proteín, nekontrolujú tvorbu proteínu priamo. Miesto toho, informácia v DNA sa najprv skopíruje do RNA. Tento proces nazývame *transkripcia*. Informácia o nukleotidoch v tejto RNA je pomocou procesu *translácie* pretransformovaná do sekvencie aminokyselín (polypeptidového reťazca). Typicky sa proteín vytvorí z jedného alebo viacerých polypeptidových reťazcov. Tento tok genetických informácií z DNA cez RNA do proteínu sa volá centrálna dogma molekulárnej biológie [4] [5].

Pojem *mutácia* sa vzťahuje k rôznym dedičným zmenám v génoch, resp. genetickom materiáli, alebo k procesom, pomocou ktorých k zmenám prebehlo.



Obr. 1.2: Jednotlivé úrovne proteínovej štruktúry. Zdroj: `www.wikipedia.org`

Rozšírené sú dva typy mutácií: génové (bodové) a chromozómové. Bodové mutácie sa odohrávajú na úrovni jednotlivých génov, kde jedna alela (forma génu) sa premení na druhú kvôli malým zmenám v sekvencii nukleotidov (v DNA). Chromozómové mutácie sú také, pri ktorých sú postihnuté celé chromozómy alebo fragmenty chromozómov a to kvôli napr. zmene pozície alebo smeru časti DNA [4] [5].

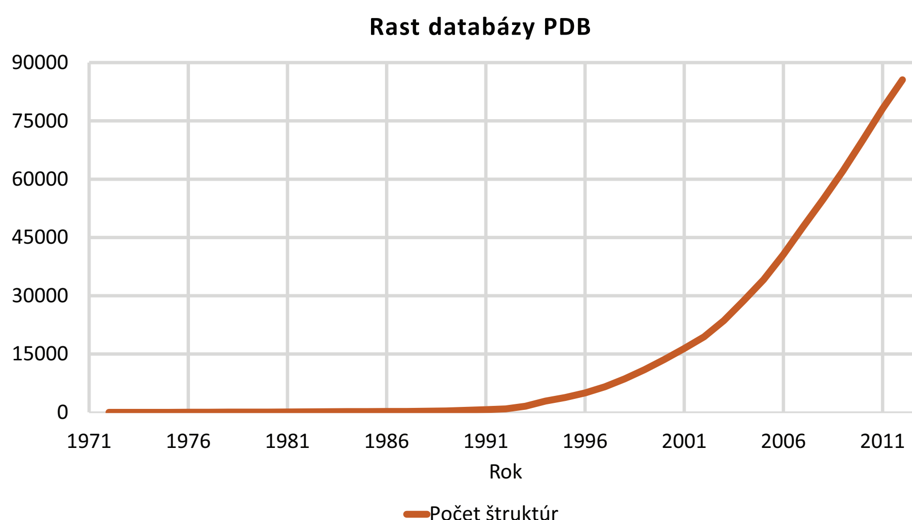


Obr. 1.3: Tok informácií v biologických systémoch. Zdroj: `www.wikipedia.org`

1.1.3 Databázy a klasifikácia proteínových štruktúr

Množstvo dôležitých zdrojov, ktoré uchovávajú informácie o sekvenciách a štruktúrach biologických molekúl, sú zaznamenané *Protein Data Bank* (PDB). Databázy sa pokúšajú spracovať dostupné informácie o štruktúrach a utriediť ich podľa štrukturálnej charakteristiky. To nám dáva výhodu skúmať evolučné vzťahy a tiež možnosť porovnania funkcii bielkovín [6].

PDB bolo založené v roku 1971 so zámerom zhromažďovať informácie o uložených štruktúrach v rôznych databázach. Celkový počet uchovaných štruktúr k 1.1.2012 je 85 582, z toho 79 275 proteínov.



Obr. 1.4: Rast počtu štruktúr zaznemenaných PDB

Fromát PDB záznamov neobsahuje len informácie o pozíciách jednotlivých atómov, ale tiež sekvenciu bielkoviny, odvodené informácie o sekundárnej štruktúre, informácie o doménach, klasifikácii, rôzne meta informácie a iné. Všetky tieto informácie sú dostupné cez FTP alebo webové rozhranie, ktoré nám poskytuje aj vizuálne náhľady štruktúry proteínov a odkazy na ďalšie databázy [7].

Keďže klasifikácia je do istej miery subjektívna, existuje viacero schém, podľa ktorých môžeme bielkoviny klasifikovať. Jednou z nich, ktorá je bežne používaná, sa riadi databáza *Structural Classification of Proteins* (SCOP).

Hierarchia databázy SCOP spočíva zo štyroch úrovní [2]:

Trieda
Hrubá klasifikácia podľa obsahu sekundárnych štruktúr α-helix a β-skladaný list.

Rod
Proteíny zdieľajú podstatnú časť sekundárnych štruktúr a majú podobné topologické rozloženie.

Nadrodina
Domény majú podobné štrukturálne záhyby a funkcie, no sekvenčná podobnosť môže byť menšia. Zdieľa sa evolučný pôvod.

Rodina
: Sekvenčná podobnosť je vysoká (30% a väčšia), taktiež môže byť podobná funkcia. Proteíny majú jasne daný evolučný vzťah.

Najvyššia úroveň (trieda) pozostáva z niekoľkých ďalších úrovní, najpodstatnejšie sú tieto štyri:

all-α
: Proteíny v tejto skupine obsahujú najmä α-helixy.

all-β
: V tejto skupine sa nachádzajú proteíny obsahujúce najmä β-skladané listy.

α/β
: Proteíny tu majú okolo α-helixov umiestnené β-skladané listy, hovoríme teda o paralelných β-skladaných listoch (β-α-β jednotky).

$\alpha+\beta$
: V tejto skupine sa v proteínoch okolo β-skladaných listov umiestňujú α-helixy, hovoríme o anti-paralelných β-skladaných listoch (alfa a beta regióny sú izolované).

Na úrovni triedy rozoznávame niekoľko ďalších úrovní ako napr. proteíny s viacerými doménami patriace do rôznych tried, proteíny neradiace sa do imunitného systému, „malé“ proteíny atď.

Okrem SCOP a iných, existuje aj ASTRAL, čo je kompendium poskytujúce databázy a nástroje využívané na analýzu proteínových štruktúr a sekvencií. ASTRAL databáza je čiastočne odvodená (podmnožina) z databázy SCOP.

Pripomeňme si aj databázu CATH, ktorá poskytuje informácie o bielkovinových štruktúrach a funkciách, ktoré sú evolučne podobné a klasifikujú sa podľa štyroch hlavných úrovní: triedy (berie sa na vedomie výskyt/charakter sekundárnych štruktúr), architektúry (tvar a usporiadanie sekundárnych štruktúr), topológie (štruktúry sú zoskupené podľa tvaru a prepojiteľnosti medzi sekundárnymi štruktúrami) a homologickej nadrodiny (tu proteínové domény zdieľajú spoločného predka).

1.1.4 Význam a uplatnenie podobnosti proteínov

Pri identifikácii nových génov a proteínov, biológovia často skúmajú podobnosti v sekvenciách proteínov už zo získaných dát. Podobnosti môžu viesť ku kľúčovým zisteniam hovoriacich o histórii evolúcie určitého génu alebo funkcie proteínu. Pre hľadanie podobností biológovia používajú nástroje dostupné na internete [9].

Pomocou podobnosti dokážeme proteíny klasifikovať, vyhľadávať v nich určité vzory, predikovať ich štruktúry či ohodnocovať ich význam.

1.2 Evolučné výpočtové techniky

Evolúcia je optimalizačný proces, ktorého cieľ je vylepšiť schopnosť organizmu alebo systému prežiť a to v dynamicky sa meniacom a konkurenčnom prostredí [8]. Poznáme rôzne evolúcie, no pre náš prípad sa zamerajme na biologickú

evolúciu. Za zakladateľa biologickej evolúcie sa považuje Charles Darwin (1809-1882) a vôbec prvý človek, ktorý začal teoretizovať o biologickej evolúcie, bol Jean Baptiste Lamarck (1744-1829).

Základom Lamarckovej teórie je dedičnosť. Jedinci, ktorí počas života získali nejaké schopnosti alebo črty, tieto schopnosti a vlastnosti preniesli na potomkov. Podľa tejto teórie sa jedinci postupne prispôsobujú prostrediu, získavajú potrebné schopnosti a nepotrebné strácajú.

Darwinova teória (prirodzeného výberu) sa dá zhrnúť nasledovne: Vo svete s obmedzenými zdrojmi a vyváženými populáciami jedincov, sa každý jednotlivec snaží bojovať s druhými o prežitie. Najviac životaschopní jedinci alebo jedinci s „najlepšími“ vlastnosťami majú väčšiu šancu na prežitie a reprodukciu [8]. Všetky význačné črty a potrebné charakteristiky sa dedia do ďalšej generácie. Darwin taktiež pripúšťa náhodné zmeny pri predávaní vlastností z rodičov na potomkov, čo môže mať negatívne ale aj veľmi pozitívne následky.

Evolučné výpočtové techniky (EVT) — v angličtine Evolutionary computation — znamená riešiť úlohu alebo systém, ktorá používa výpočtové modely evolučných procesov pomocou počítača.

1.2.1 Všeobecný princíp evolučných algoritmov

Evolučný algoritmus (EA) predstavuje stochastické hľadanie optimálneho riešenia na daný problém. Evolúciu ako prirodzený výber náhodne zvolenej populácie indivíduí chápeme ako prechod priestoru všetkých možných konfigurácií chromozómov.

Základný princíp EA pozostáva z niekoľkých častí [8]:

- zo zakódovania riešenia problému do chromozómov,
- z funkcie na získanie *fitness* (predstavuje silu jedinca),
- z inicializácie počiatočnej populácie,
- zo selekčných a
- z reprodukčných operácií.

Algoritmus 1.2.1: Základný evolučný algoritmus

Nech $t = 0$ je počítadlo generácií;
Vytvoríme a inicializujeme n_x-rozmernú populáciu $C(0)$ pozostávajúcu z n_s indivíduí;
while *podmienka condition*(s) *na zastavenie neplatí* **do**
    Získajme *fitness* $f(x_i(t))$ pre každého jedinca $x_i(t)$;
    Reprodukciou vytvorme potomkov;
    Aplikujme mutáciu na nových jedincov;
    Vyberme novú populáciu $C(t+1)$;
    Posuňme sa na ďalšiu generáciu, t.j. $t = t + 1$;
end

Algoritmus 1.2.1 ilustruje základný priebeh EA. Sú v ňom zahrnuté oba nasledujúce prvky Darwinovej teórie:

- Počas reprodukcie prebieha prirodzený výber, kde „najlepší“ rodičia majú väčšiu šancu, že budú vybratí a vytvoria potomkov, ktorí sa zaradia do novej populácie.
- Pomocou operácie mutácie prebiehajú náhodné zmeny.

Nasledovné paradigmy implementujú a reprezentujú rôzne pohľady na EA [8]:

Genetické algoritmy (GA)
: Modelujú genetickú evolúciu.

Genetické programovanie (GP)
: Sú založené na genetických algoritmoch, pričom indivíduá predstavujú programy (reprezentované stromami).

Evolučné programovanie
: Je odvodené zo simulácie adaptívneho správania sa v populácii (jedná sa o tzv. fenotypovú evolúciu).

Evolučné stratégie
: Sú zamerané na modelovanie strategických prvkov, ktoré kontrolujú zmeny v evolúcii (tu ide o evolúciu evolúcie).

Diferenciálna evolúcia
: Je podobná genetickým algoritmom, no líšia sa v reprodukčnom mechanizme.

Kultúrna evolúcia
: Modeluje evolúciu kultúry populácie, sústredí sa na to, ako kultúra vplýva na genetickú a fenotypovú evolúciu jedincov.

Ko-evolúcia
: Spočiatku „primitívne“ indivíduá sa vyvíjajú využívajúc spoluprácu alebo zápasenie so zvyšnými jedincami a tým získavajú potrebné vlastnosti na prežitie.

1.3 Motivácia práce

V tejto práci sa budeme zameriavať hlavne na možnosti, ktoré nám ponúkajú evolučné výpočtové techniky. Budeme sa držať hlavne ich princípov, ktoré implementujeme a ďalej sa budeme držať určitými pravidlami.

Proteín budeme vnímať z troch aspektov — ako sekvenciu, ako trojdimenzionálnu štruktúru pozostávajúcu z komplexnejších objektov a nakoniec ako stromový objekt obsahujúci informácie o týchto komplexnejších objektoch.

Náš hlavný cieľ je si vybrať jednu variantu využitia, otestovať ju na nejakej množine bielkovín a určiť, či genetické programovanie je správna voľba pri úlohe porovnávania proteínových štruktúr.

V úvodnej kapitole sme si popísali stavbu proteínu, ich spôsob ukladania do databáz a princíp evolučných algoritmov. V druhej kapitole si predstavíme genetické programovanie a podrobnejšie popíšeme jednotlivé fázy. Tretia kapitola sa

zaoberá samotným problémom porovnávania proteínových štruktúr, kde si predstavíme niekoľko existujúcich metód a rozanalyzujeme náš cieľ v zmysle, čoho sa treba držať a kde sa nám otvárajú možnosti. Vo štvrtej kapitole si popíšme konkrétny návrh a priebeh celého naimplementovaného programu a v piatej kapitole si predstavíme výsledky experimentov.

Kapitola 2

Genetické programovanie

V tejto kapitole si popíšeme paradigmu EA — genetické programovanie, na ktoré sa budeme zvyšok práce zameriavať.

Genetické programovanie (GP) tvorí rozšírené vnímanie genetických algoritmov (GA). Umožňuje nám riešiť problémy, ktoré môžeme popísať komplexnejšie. V riešení problémov používame štruktúry, ktoré sú obsahovo bohatšie, všeobecnejšie a predstavujú hierarchické počítačové programy s dynamicky sa meniacou veľkosťou, tvarom a vlastnosťami.

GP vyvinul a prepracoval John R. Koza [10]. Testoval a meral dosiahnutý výkon vývoja programu (indivídua), ktorý posudzoval podľa dosiahnutej *fitness*. Proces riešenia problémov môže byť teda formulovaný ako hľadanie najlepšieho počítačového programu (jedinca) v priestore všetkých možných počítačových programov. Konkrétne, prehľadávací priestor je priestor všetkých možných počítačových programov, ktoré pozostávajú z funkcií a termov vhodné pre doménu daného problému.

V prílohe tejto práci si môžeme prezrieť diagram B.4 vyjadrujúci paradigmu genetického programovania.

2.1 Stromová reprezentácia

V tradičných GA a GP sú štruktúry podliehajúce evolúcii vnímané ako jedinci, resp. indivíduá v populácii. Každý jedinec je reprezentovaný stromovou štruktúrou, ktorá má hierarchický význam. Túto štruktúru budeme definovať podľa definícií Johna R. Kozu [10].

Množina možných štruktúr v GP je množina všetkých možných kompozícií funkcií, ktoré sa môžu skladať rekurzívne z danej množiny N_{func} funkcií z $F = \{f_1, f_2, \ldots, f_{N_{func}}\}$ a z množiny N_{term} termov z $T = \{t_1, t_2, \ldots, a_{N_{term}}\}$. Každá jedna funkcia f_i poberá $z(f_i)$ argumentov, t.j. funkcia f_i má aritu $z(f_i)$. Dôležitú vlastnosť, ktorú musia funkcie spĺňať, je *uzavretosť*.

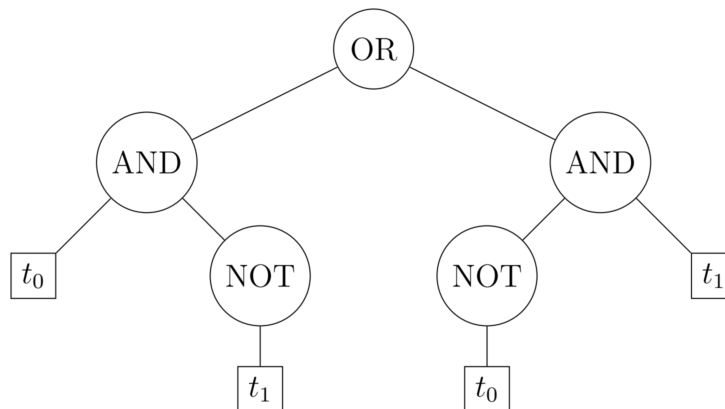
Funkcie v množine funkcií môžu zahrňovať:

- aritmetické operácie ($+$, $-$, $*$, atď.),
- matematické funkcie (*sin*, *cos*, *exp*, *log*, ...),
- operácie typu *Boolean* (napr. *AND*, *OR*, *NOT*),
- podmienené výrazy (ako napr. *If-Then-Else*),

- „iteračné“ funkcie (ako napr. *Do-Until*),
- rekurzívne funkcie a
- iné funkcie špecifikované nejakou doménou.

Termy typicky predstavujú atomické prvky (napr. vstupy, senzory, detektory, alebo stavy premenných nejakého systému) alebo konštanty (napr. číslo 3 alebo hodnota *NIL*). Príležitostne môžu byť termy funkciami bez argumentov.

Uveďme si príklad. Majme množinu funkcií $F = \{AND, OR, NOT\}$, množinu termov $T = \{t_0, t_1\}$, kde t_0 a t_1 sú premenné typu *Boolean*, ktoré slúžia ako argumenty pre dané funkcie. Ďalej uvážme *Boolean* výraz (t_0 *AND NOT* t_1) *OR* (*NOT* t_1 *AND* t_0). Evolučný program bude mať na vstupe množinu funkcií a termov, a bude mať za úlohu nájsť riešenie, kedy daná formula bude vždy pravdivá. Toto riešenie popisuje strom 2.1 nižšie.



Obr. 2.1: Stromová reprezentácia operácie *XOR*

2.2 Počiatočná populácia

Populáciu na začiatku vytvoríme náhodne s danou maximálnou hĺbkou. Pre každého jedinca z množiny funkcií náhodne vyberieme jeden prvok, ktorý bude predstavovať koreň. Ďalej pre každý ne-koreňový uzol stromu vyberieme jeden prvok z prieniku množín funkcií a termov. Akonáhle je pre nejaký uzol zvolený prvok z množiny termov, tento uzol sa stane listom.

Pokiaľ možno, tak na začiatku sa snažme vytvoriť indivíduá s jednoduchou štruktúrou. Táto štruktúra počas priebehu evolúcie narastie, ak bude táto možnosť povolená. Pre počiatočných jedincov, resp. stromy, môžeme rovnako nastaviť maximálny alebo minimálny počet synov i iné parametre vhodné pre stromovú štruktúru.

Pri prvotnom vytváraní jedincov je dobré zabezpečiť, aby sme nevytvárali duplikáty, čo nám zaistí unikátnosť jedincov. V neskorších generáciách je vytváranie duplikátov neodmysliteľná súčasť evolúcie pomocou genetickej operácie reprodukcie.

2.3 *Fitness* funkcia

Fitness funkcia popisuje „vitálnosť“ jedinca. Je veľmi dôležitým prvkom genetického procesu, podľa ktorého dochádza k rozhodnutiu, či daný jedinec prežije alebo sa zreprodukuje, alebo sa priradí na vstup inej genetickej operácii.

Fitness môže byť meraná rôznymi spôsobmi. Buď ju meriame ako nejakú absolútnu hodnotu pre každého jedinca, alebo pri ohodnocovaní zohľadňujeme aj zvyšok populácie, alebo túto hodnotu môžeme normalizovať a využiť pri tom rôzne distribučné funkcie.

Pri ohodnocovaní jedinca, sa môže do *fitness* započítať aj určitá penalizácia, ak daná štruktúra obsahuje neželateľný charakter alebo ak je jedinec sémanticky nekorektný.

2.4 Primárne operácie pre modifikovanie štruktúr

Táto sekcia popisuje dve hlavné operácie pre modifikovanie štruktúr podliehajúcich vývoju v genetickom programovaní a to:

- Darwinovú reprodukciu a
- kríženie (pohlavnú rekombináciu).

2.4.1 Reprodukcia

Operácia reprodukcie je hlavným „motorom“ Darwinovej prirodzenej selekcie a prežitia „najlepšieho“ jedinca. Táto operácia je nepohlavná, prebieha len na jednom rodičovi a pri každom vykonaní vzniká jeden potomok.

Tento úkon pozostáva z dvoch krokov. Najprv vyberieme jedného jedinca z populácie a to podľa niektorej selekčnej metódy založenej na hodnote *fitness*. Pri druhom kroku sa indivíduum bez zmeny skopíruje z aktuálnej do novej populácie (generácie).

Existuje viacero metód, pomocou ktorých môžeme vybrať jedinca z populácie a prekopírovať ho do novej populácie. Jedna z najpopulárnejších je metóda založená na úmernosti hodnoty *fitness*. Popisuje ju už John H. Holland v roku 1975 a sú na nej postavené výsledky Hollandových teórií.

Ak $f(s_i(t))$ je *fitness* jedinca s_i v populácii generácii t, M je veľkosť populácie, potom podľa selekčnej metódy založenej na úmernosti *fitness*, sa jedinec s_i skopíruje do ďalšej populácie s pravdepodobnosťou

$$\frac{f(s_i(t))}{\sum_{j=1}^{M} f(s_j(t))}. \tag{2.4.1}$$

Miesto vyššie opisovanej selekčnej metódy poznáme aj *tournament* a *rank* selekcie. Pri *rank* selekcii, je výber jedinca založený na rebríčku (umiestnení) jedincov v populácii podľa *fitness*. Tento typ selekcie redukuje efekt dominanty indivíduí s najvyššou hodnotou *fitness*. Efekt môžeme potlačiť nastavením vhodnej hodnoty premennej, podľa ktorej môžeme dať pri výbere jedinca z populácie väčšiu prednosť jedincovi s vyššou hodnotou *fitness*. *Rank* selekcia má tiež tú

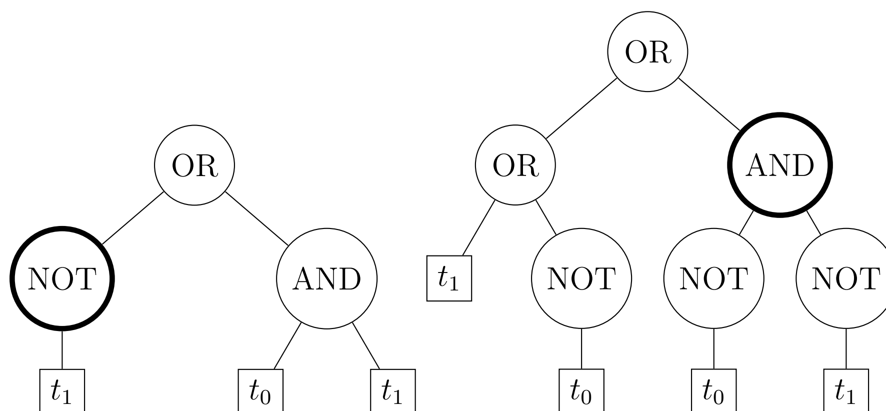
výhodu, že sa týmto spôsobom vedia zdôrazniť i minimálne rozdiely hodnôt *fitness*.

Pri *tournament* selekcii, si náhodne z populácie vyberieme skupinu jedincov (typicky dvoch) a vyberieme toho s lepšou hodnotou *fitness* (normovanou *fitness*). Tento proces môžeme prirovnať k boju dvoch býkov o právo páriť sa s jalovicou.

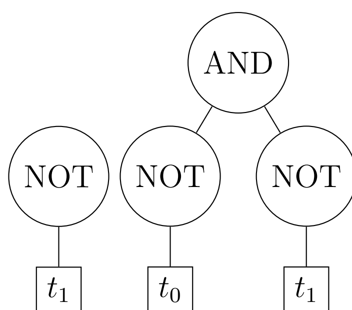
Hodnotu *fitness* nemusíme prepočítavať pre jedinca, ktorý bol reprodukovaný z predošlej generácie do novej. Ak by sme napr. 10% populácie zreprodukovali do novej generácie, pre týchto jedincov nemusíme *fitness* počítať znova a ušetríme nejaký čas pre výpočet.

2.4.2 Kríženie

Operácia kríženia vytvára pri procese evolúcie vlastnosť rôznorodosti a väčšej variácie štruktúr. Na začiatku vyberieme dvoch rodičov z populácie pomocou selekcie ako aj pri reprodukcii. Následne vyberieme náhodný bod v štruktúre rodičov, ktorý určí časti štruktúr, ktoré sa prekrížia, resp. vymenia.



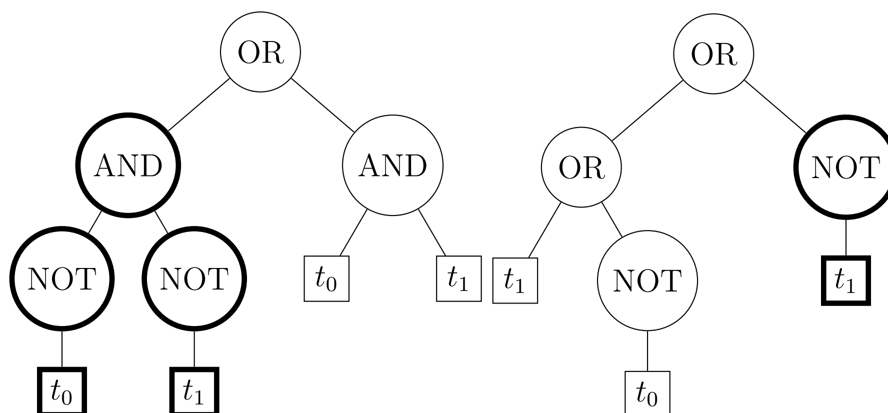
Obr. 2.2: Dve rodičovské indivídua a body kríženia



Obr. 2.3: Fragmenty, ktoré sa obmenia

Pretože sa pri tejto operácii obmieňajú celé podstromy, a pretože funkcie spĺňajú vlastnosť uzavretosti, kríženie vždy vyprodukuje syntakticky korektné štruktúry.

Maximálna prípustná veľkosť štruktúry (zväčša daná maximálnou hĺbkou stromu) je dôležitý prvok pri vytváraní potomkov. Tento limit nám zaručuje, že evolúcia nebude manipulovať s veľkými štruktúrami, ktoré by mohli spomaliť celý priebeh. Navyše, veľmi veľké štruktúry nemusia predstavovať ideálne riešenie.



Obr. 2.4: Potomkovia ako výsledok kríženia

Treba si teda uvedomiť, že počet všetkých možných štruktúr závisí od hĺbky stromu a počtu synov.

Po vytvorení potomkov skontrolujeme hĺbku oboch štruktúr a ak hĺbka nejakej z nich prekračuje limit, tak miesto potomka vložíme do novej populácie príslušného rodiča.

2.5 Sekundárne operácie

Okrem dvoch hlavných operácií si predstavíme aj niekoľko ďalších voliteľných. Koza [10] ich predstavuje päť, ktoré sa podľa neho oplatí spomenúť a to operácie:

- mutácie,
- permutácie,
- editácie,
- enkapsulácie (zaobalenia) a
- decimácie (zničenia).

Podstatou týchto sekundárnych operácií je nejakým spôsobom zmeniť alebo vymeniť istý prvok alebo časť štruktúry jedinca.

Pri mutácii sa môže vymeniť náhodne vybraný uzol za nejaký iný toho istého druhu, resp. ak sa jedná o funkciu s aritou n, tak sa zamení s inou funkciou s aritou n, ktorú by sme náhodne vybrali z množiny funkcií. Podobne platí prípad aj pre termy.

Permutovať môžeme napr. poradie listov.

Editovať môžeme v zmysle, že niektoré konštrukcie zjednodušíme — napr. ako pri úprave výrazu *(NOT (NOT (NOT t_0)))* na výraz *(NOT t_0)*. Enkapsulovať alebo zaobaliť sa dá nejaký podstrom, kde zmeníme koreň podstromu za nový term alebo funkciu, ktorú môžeme definovať za behu.

Pre niektoré problémy, kedy sa v populácii nachádza veľké percento indivíduí s veľmi malou hodnotou *fitness*, sa môže distribúcia hodnôt *fitness* podstatne skresliť. Tým pádom by sa naša evolúcia vyvíjala veľmi ťažko a ako riešenie

takéhoto problému a problému s podobnou tématikou skreslenia distribúcie *fitness* je použitie operácie decimácie. Táto operácia je kontrolovaná dvoma parametrami: percentom a podmienkou, kedy sa operácia vykoná. Uveďme si príklad, prvý parameter majme nastavený na 10% a podmienku takú, že sa operácia vykoná pri počiatočnej generácii. Keď sa operácia vykoná, zmaže sa 10% populácie. Ak sa táto operácia pri prvom behu vykoná, zopakujeme prvý beh s populáciou 10-krát väčšou. Jedincov, ktorých mažeme, vyberáme podľa pravdepodobnosti so základnou hodnotou *fitness* 2.4.1.

2.6 Terminácia

Evolúcia, ktorá je paradigmou genetického programovania predstavuje v prírode nekonečný proces. Avšak pre naše účely, beh genetického programovania je ukončený, ak splní terminačné kritérium.

Toto kritérium je typicky definované ako maximálny počet behov, resp. počet generácií, po ktorých sa proces zastaví. No tiež môže znamenať napr. podmienku, pri ktorej splnení sa evolúcia zastaví — to môžeme využiť v prípade, ak je náročné získať 100% správny výsledok a podmienka by bola: „ak existuje jedinec, ktorý spĺňa očakávaný výsledok na 90% správny, potom terminačné kritérium považujeme za splnené“.

Kapitola 3

Porovnávanie proteínov

V tejto kapitole si predstavíme niektoré metódy a ich princíp, ktorým sa proteíny porovnávajú. Tiež spomenieme riešenia, ktoré pre porovnanie využívajú evolučné výpočtové techniky. Nakoniec rozanalyzujeme možnosti pre využitie genetického programovania.

3.1 Sekvenčné zarovnávanie

Sekvenčné zarovnanie je procedúra porovnávania dvoch (prípadne viacerých) sekvencií, t.j. primárnej štruktúry a to vyhľadávaním súvislej podpostupnosti jednotlivých znakov alebo vzorov, ktoré sú v danej sekvencii v rovnakom poradí. Táto metóda slúži ako základ pre mnohé algoritmy i pre štrukturálne zarovnávanie opísané v časti 3.2.

Rozoznávame dva druhy sekvenčného zarovnávania: globálne a lokálne [12]. Rozdiel je zobrazený na obrázku 3.1. Pri globálnom zarovnaní zahrňujeme celé proteínové (primárne) sekvencie a snažíme sa zarovnať čo najviac aminokyselín. Sekvencie, ktoré sú podobné a majú približne rovnakú dĺžku, sú vhodnými kandidátmi pre tento typ zarovnania. V lokálnom zarovnaní sa zarovnávajú (párujú) úseky s najväčšou hustotou spárovaných aminokyselín. Tento typ je zas vhodnejší pre sekvencie, ktoré sú si podobné v niektorých úsekoch a odlišujú sa v iných, sekvencie s rôznymi dĺžkami alebo sekvencie, ktoré zdieľajú napr. doménu.

```
                        L G P S S K Q T G K G S - S R I W D N
Globálne zarovnanie     |         |     | | |       |     |
                        L N - I T K S A G K G A I M R L G D A

                        - - - - - - - T G K G - - - - - - - -
Lokálne zarovnanie                    | | |
                        - - - - - - - A G K G - - - - - - - -
```

Obr. 3.1: Odlišnosť medzi globálnym a lokálnym zarovnaním

3.1.1 Globálne sekvenčné zarovnanie

Prvý algoritmus pre vyhľadávanie globálneho zarovnania bol zverejnený v roku 1970, ktorého autormi sú Needleman and Wunsch [13], podľa ktorých je algo-

ritmus aj pomenovaný. Jeho podstatou je využitie dynamického programovania. Tento algoritmus bolo však dobré upraviť tak, aby zahrňoval penalizáciu za medzeru a odlišné ohodnocovanie párovaných aminokyselín.

Uvážme dve sekvencie reprezentujúce primárnu štruktúru bielkovín A a B: $S_A = (a_1, a_2, \ldots, a_n)$ a $S_B = (b_1, b_2, \ldots, b_m)$. Na pozícii $V[i, j]$ v matici V nájdeme ohodnotenie optimálneho zarovnania, kde i je prvok zo sekvencie S_A a j je prvok zo sekvencie S_B. Pomocou rekurzívneho algoritmu vyjadrením rovnicami 3.1.1 nižšie popísanými prejdeme všetky prvky $[i, j]$ matice V.

$$V[i,j] = max \begin{cases} V[i-1,j-1] + \psi(a_i, b_j) \\ V[i,j-1] + \Omega \\ V[i-1,j] + \Omega \end{cases} \tag{3.1.1a}$$

$$V[i,0] = i \times \Omega \tag{3.1.1b}$$

$$V[0,j] = j \times \Omega \tag{3.1.1c}$$

Funkcia $\psi\,(a_i, b_j)$ vracia hodnotu, ktorá predstavuje mieru podobnosti aminokyselín a_i a b_j. Hodnota Ω je penalizácia za medzeru v zarovnaní.

3.1.2 Lokálne sekvenčné zarovnanie

Jednoduchá modifikácia algoritmu globálneho zarovnania predstavuje algoritmus pre lokálne sekvenčné zarovnanie, ktorý identifikuje najdlhšie spárované homologické podsekvencie [14]. Modifikácia spočíva v zakázaní záporných hodnôt v matici. Vyššie spomínaný algoritmus 3.1.1 preto prepíšeme do nasledujúceho tvaru:

$$V[i,j] = max \begin{cases} V[i-1,j-1] + \psi(a_i, b_j) \\ V[i,j-1] + \Omega \\ V[i-1,j] + \Omega \\ \mathbf{0} \end{cases} \tag{3.1.2a}$$

$$V[i,0] = i \times \Omega \tag{3.1.2b}$$

$$V[0,j] = j \times \Omega \tag{3.1.2c}$$

Celkové zarovnanie (pre oba typy) predstavuje cestu z pozície najväčšej hodnoty v matici V do pozície s nulovou hodnotou cez susedov s najväčšími hodnotami. Tento prechod voláme tiež *backtracking*.

3.2 Štrukturálne zarovnávanie

Algoritmy pre určovanie štrukturálnej podobnosti proteínov vyhľadávajú ekvivalentné rezíduá medzi dvoma proteínmi v priestore, resp. na úrovni terciárnej štruktúry. Táto množina ekvivalentných rezíduí sa nazýva *štrukturálne zarovnanie*, ktoré má využitie v dvoch hlavných oblastiach výskumu.

Prvá z nich je, že takéto zarovnanie môže slúžiť pre indikáciu rovnakých záhybov a tvarov medzi dvoma proteínmi. Vysoká štrukturálna podobnosť je často výsledkom funkčnej alebo evolučnej podobnosti bielkovín. Navyše oproti

sekvenčnej podobnosti môže štrukturálne porovnanie identifikovať spoločné znaky aj medzi evolučne vzdialenejšími bielkovinami. V druhej oblasti štrukturálne porovnanie predstavuje najpresnejšiu výpočtovú techniku ako opísať ekvivalenciu párov rezíduí s ohľadom na funkciu. Preto toto porovnanie znamená neoddeliteľnú súčasť metód štrukturálnej predikcie, modelovania homologických prvkov a iných metód. Štrukturálna podobnosť slúži ako zlatý štandard pre predikčné metódy a tiež pre klasifikáciu bielkovín.

Čo sa týka zložitosti zarovnávacích algoritmov, pri sekvenčnom zarovnávaní vieme nájsť optimálne zarovnanie v kvadratickom čase, pričom nájdenie optimálneho párovania proteínových štruktúr predstavuje NP-úplný problém.

3.3 Metódy pre porovnávanie bielkovín

Existuje mnoho metód, ktoré môžeme pre porovnanie použiť alebo ktoré určujú akýsi stupeň podobnosti štruktúr. Väčšinou sú založené na štrukturálnom zarovnávaní alebo na zarovnaní intra alebo inter-molekulárnych atomických vzdialeností. V krátkosti si popíšeme metódy, ktoré provnával Singh Amit P. a Brutlag Douglas L. [11] a niektoré ďalšie:

DALI (Holm a Sander [15])
: Algoritmus využíva matice vzdialeností, ktoré sú vypočítané zo vzdialeností rezíduí (atómov α-uhlíkov) v trojdimenzionálnom priestore. Tieto matice sú najprv dekomponované na elementárne submatice. Podobné submatice sú následne spárované a zložené do väčších konzistentných párov. Metódou *Monte Carlo* sa optimalizuje „podobnostné“ skóre, ktoré sa spočíta vzhľadom na ekvivalentné intra-molekulárne vzdialenosti.

STRUCTURAL (Gerstein a Levitt [16])
: V tejto metóde sa Gerstein a Levitt snažili dosiahnuť presnejšie zarovnanie zachovávajúce štrukturálne oblasti. Používali premennú, ktorá penalizovala medzeru v zarovnaní vzhľadom na pozíciu v sekundárnej štruktúre. Najprv spomedzi všetkých možných spárovaných zarovnaní našli štruktúru, ktorá bola mediánom medzi štruktúrami a následne zarovnávali už len k tejto „mediánovej“ štruktúre.

VAST (Gibrat a spol. [17])
: Gibrat a spol. sa v tomto prípade pozreli na problém ako problém týkajúci sa teórie grafov. Všetky páry sekundárnych štruktúr, ktoré boli rovnakého typu, reprezentovali ako vrcholy v grafe. Dva vrcholy boli spojené, ak vzdialenosť a uhol medzi párom sekundárnych štruktúr korešpondovali nejakej danej hranici. Pri porovnávaní sa na základe tohto hľadal najväčší podgraf s rovnakými vlastnosťami.

MINAREA (Falicov a Cohen [18])
: Tento algoritmus je založený na probléme hľadania minimálnej plochy pomocou dynamického programovania. Vytvorí sa akýsi virtuálny polypeptidový reťazec, spájajúci jednotlivé rezíduá v proteínovej štruktúre a trianguláciou medzi vzdialenosťami rezíduí sa získa určitá plocha oblasti. Pri

počítaní vzdialeností sa využíva *RMSD* (root-mean-square deviation, resp. priemerná kvadratická odchýlka)

LOCK (Singh a Brutlag [19])
V tomto algoritme sa sekundárne štruktúry každého proteínu reprezentujú ako vektory a používa sa 7 funkcií určujúcich skóre zarovnania (od úrovne sekundárnych štruktúr až po úroveň atomickú).

Kpax (Ritchie a spol. [20])
Pri štrukturálnom zarovnaní sa využíva predvídanie kovalentných usporiadaní medzi rezíduami. To sa následne využíva na určenie lokálnych oblastí, v ktorých sa časti polypeptidových reťazcov môžu nachádzať a porovnať za použitia ohodnocovacích funkcií zvaných *Gaussian overlap scoring functions*. Aplikuje sa tiež algoritmus globálneho zarovnania aj s penalizáciou medzery v zarovnaní.

SProt (Galgonek J.; Hoksza D. a Skopal T. [21])
Základnou „porovnávacou" jednotkou pri tejto metóde je sféra, ktorá pozostáva z v priestore priľahlých (blízkych) aminokyselín, ktoré sú rozdelené do niekoľkých skupín. Na týchto sférach sa aplikuje lokálne a globálne zarovnanie. Napokon sa využije indexácia, ktorá urýchľuje vyhľadávací proces v databáze bielkovinových štruktúr.

3.4 Existujúce riešenia využívajúce evolučné výpočtové techniky

Skúmaním podobnosti proteínových štruktúr sa podrobne zaoberal Szustakowski a Weng [22] [23], ktorí vyvinuli metódy KENOBI a K2. Ich cieľom bol algoritmus, ktorý generoval presné a biologicky zmysluplné zarovnania za využitia genetických algoritmov. K2 pritom predstavovala rozšírenie, resp. zlepšenie metódy KENOBI.

3.4.1 KENOBI

Pri počiatočnom štrukturálnom zarovnaní bielkovín použil Szustakowski filozofiu, kedy k sebe najprv zarovnal sekundárne štruktúry rovnakého typu a následne rozšíril toto zarovnanie o zvyšné rezíduá nájdené v slučkách a ohyboch štruktúry. To predstavovalo inicializačnú populáciu pre genetický algoritmus. Ďalej sa tieto zarovnania modifikovali a prekombinovali aj na úrovni zvyšných rezíduách. Následne túto metódu testovali na niekoľkých rodinách proteínov.

Myšlienka prvotného zarovnania štruktúry podľa SSE (prvkov sekundárnych štruktúr[1]) znamená, že týmto spôsobom vylúčime nezmyselné párovania častí, ktoré k sebe štrukturálne nepatria. Okrem toho sa nám zvýši výkon a zmenší prehľadávací priestor.

Pri párovaní rezíduí sa držíme troch podmienok: rezíduá sú najbližšími susedmi v priestore, musia sa od seba nachádzať v danej maximálnej vzdialenosti

[1] Anglicky SSE znamená *Secondary Structure Element* a SSE budeme chápať buď ako α-helix, alebo ako β-skladaný list.

a tiež v oblasti, v ktorej sú štyri po sebe idúce spárované rezíduá. Týmito podmienkami vylúčime páry, ktoré by mohli byť v priestore veľmi blízko alebo by nemuseli byť ekvivalentné. Ekvivalentné by neboli napríklad v prípade, ak by sa postranné reťazce bielkovín pretínali v priestore v rovnakom bode a to by nemuselo znamenať, že na tomto mieste sú rezíduá ekvivalentné.

Rozhodujúce pre Szustakowského bolo vybrať správnu ohodnocovaciu, resp. *fitness* funkciu. Použil preto *elastic similarity score* 3.4.1, čím sa zaoberal už Holm a Sander [15].

$$S = \begin{cases} \sum_{i=1}^{L} \sum_{j=1}^{L} \left(\theta - \dfrac{|d_{ij}^A - d_{ij}^B|}{d_{ij}^*} \right) e^{-(d_{ij}^*/\alpha)^2}, & i \neq j \\ \theta, & i = j \end{cases} \tag{3.4.1}$$

Toto ohodnocovanie využíva intra-molekulárne vzdialenosti. Je definované párovanými rezíduami: i_A-tým (patriacim proteínu A) a i_B-tým (z proteínu B); L je počet párovaných rezíduí; d_{ij}^A je prvok matice vzdialeností proteínu A (analogicky pre i_B-tý prvok), znamená vzdialenosť medzi i_A-tým a i_B-tým rezíduum. d_{ij}^* je priemer hodnôt d_{ij}^A a d_{ij}^B; θ je konštanta určujúca podobnostnú mieru (nastavená na 0.20) a $e^{-(d_{ij}^*/\alpha)^2}$ je obalová funkcia ($\alpha = 20$ Å [2]). Táto obalová funkcia je navrhnutá tak, aby od seba veľmi vzdialené páry neskresľovali celkové skóre. Reziduá, ktoré nie sú spárované, nemajú podiel na celkovom skóre.

Algoritmus 3.4.1: Schéma genetického algoritmu v KENOBI

```
Vytvorme počiatočnú populáciu zo všetkých možných SSE zarovnaní;
while podmienka na zastavenie neplatí do
    Každé zarovnanie modifikujme mutačnými operáciami;
    Aplikujme kríženie na jedincov;
    Príjmime alebo odmietnime zmeny vykonané na každom jedincovi;
end
```

Algoritmus 3.4.1 ilustruje princíp genetického algoritmu metódy KENOBI. Každé SSE zarovnanie je reprezentované ako zoznam párovaných SSE. Každé SSE je párované s SSE rovnakého typu alebo tzv. *null* SSE. Szustakowski vyžadoval, aby každé SSE obsahovalo aspoň 4 párované rezíduá.

	α-helixy		**β-skladané listy**		
Proteín A	**1**	**2**	**1**	**2**	**3**
Proteín B	**1'**	**2'**	**1'**	**null**	**2'**

Obr. 3.2: Párovanie SSE v KENOBI

Terminačné kritérium sa pri GA splní, ak sa vykoná počet všetkých kôl alebo ak sa najlepšie skóre nezmení po 20 po sebe idúcich iterácií genetického algoritmu,

[2] Å (ångström) je často používaná jednotka pre vyjadrovanie vzdialeností a veľkostí atómov, molekúl a rôznych biologických štruktúr. 1 Å = 10^{-10} m.

alebo ak sa priemerné skóre populácie vyrovná skóre najlepšieho jedinca.

Po skončení GA sa zarovnania ešte upravia, vylepšia; Szustakowski tiež na ten účel implementoval „shake“ operáciu, ktorá posúva párovanie na úrovni rezíduí.

Nakoniec sa vyhľadávajú dodatočné ekvivalentné páry rezíduí v proteínových reťazcoch a to hlavne v oblastiach mimo SSE.

3.4.2 K2

V predošlej sekcii sme si opísali niektoré hlavné časti programu KENOBI, ktorý Szustakowski vo svojej dizertačnej práci rozšíril na program K2, ktorý obsahoval dve hlavné nové vylepšenia. Jedno spočíva v SSE zarovnaní, ktoré sa reprezentovali vektormi. Tieto zarovnania sa inicializovali pred GA a celý proces značne zrýchlili. Druhé vylepšenie je založené na štatistických výpočtoch určujúce hodnotu výsledných zarovnaní.

Fungovanie GA sa zachovalo, rovnako ako aj *fitness* funkcia a operácie. V prílohe na obrázku B.1 je ilustrovaná operácia *swap* a obrázok s rodičmi B.2 a synmi B.3, a zobrazujú kríženie SSE zarovnaní.

3.5 Analýza využitia GP pri porovnávaní

Uvedomme si, že náš cieľ je nájsť spôsob využitia GP pri štrukturálnom porovnaní a GP je veľmi silný nástroj založený na princípe evolúcie živých organizmov. Otázka, resp. podstata evolučných výpočtových metód sa dá opísať parafrázovaním A.L. Samuela [24]:

> Ako sa počítače môžu naučiť riešiť problém bez toho, aby sme ich preň explicitne naprogramovali?

Ak je našou úlohou zarovnať dve bielkoviny, po počiatočnom spárovaní s využitím prípadných základných metód získame nejaké párovanie. Už toto párovanie môže predstavovať systém prepojenia vyšších objektov miesto prepojenia objektov základných ako napr. objektov reprezentujúce aminokyseliny. Tento systém je ale treba použiť v GP a to tak, aby sme nestratili možnosti, ktoré GP ponúka.

Môžeme povedať, že hlavný znak GP je strom reprezentujúci jedinca v evolúcii. V našom prípade bude strom teda znamenať párovanie, ktoré budeme chcieť optimalizovať do najlepšieho možného párovania, resp. stromu. Počiatočné párovanie potrebujeme previesť na „stromový“ objekt.

Strom, ktorý môžeme zostaviť z funkcií a termov ako ich definoval John R. Koza, dokáže niesť veľký význam nad párovaním a dovoľuje nám postaviť komplexnejšie konštrukcie. Taký strom môže na rôznych úrovniach poskytovať iné možnosti, ktoré by GP a časti GP obohatili. Toto je jeden z hlavných dôvodov, prečo je dobré si vybrať GP a použiť stromovú reprezentáciu miesto napr. klasických genetických algoritmov alebo evolučného programovania.

Pri návrhu sa budeme zameriavať na princípy GP. Tiež budeme chcieť ponúknuť iné riešenie od riešení, než sa doposiaľ vyvinuli. Už spočiatku pri vytváraní základných objektov ako napr. objektov reprezentujúce aminokyseliny budeme myslieť na to, že ak vytvoríme komplexnejší objekt alebo objekt nesúci vyšší význam, budeme ho neskôr môcť využiť v GP v stromovej reprezentácii jedinca.

Tiež bude dôležité a prínosné, ak sa budeme držať čo najmenších algoritmických zložitostí a využijeme rôzne heuristiky. Pri využití stromovej reprezentácii môže implementácia algoritmov niesť zvýšenú náročnosť.

Kapitola 4

Metóda ProSSiGen

V predošlej kapitole sme sa zbežne oboznámili s metódami porovnávajúce štruktúru bielkovín a získali sme tiež náčrt, ako využiť GP pri tomto procese porovnávania. Nakoniec sme si v analýze zhrnuli možnosti a smer, akým by sme sa mali k cieľu vybrať. V tejto časti si navrhneme objekty reprezentujúce jednotlivé časti celého priebehu zarovnávania a potom popíšeme celý proces programu ProSSiGen[1], ktorý je implementovaný v jazyku C++ s využitím knižníc Qt.

Jazyk C++ sme si vybrali kvôli jeho veľkej výhode a to rýchlosti, čo sa nám pri takejto algoritmicky náročnej úlohe hodí.

Qt knižnice v programe používame pre pohodlnejšie spracovanie konfiguračného súboru vo formáte XML. Taktiež pre čítanie a zápis do súborov a pre implementáciu viacerých vlákien programu.

Aby sme sa pri návrhu objektov lepšie orientovali v úlohách jednotlivých objektov, načrtnime celý proces týmito bodmi:

- načítanie súboru, ktorý popisuje danú bielkovinu,
- vytvorenie základných objektov,
- vykonanie globálneho sekvenčného zarovnania,
- prevod zarovnania na stromovú štruktúru,
- algoritmy GP a výsledok.

4.1 Návrh objektov

Objekty používajúce v programe ProSSiGen implementujúce túto metódu môžeme podľa využitia rozdeliť do troch skupín: na základné objekty, na objekty týkajúce sa globálneho zarovnania (dodatočné objekty) a na objekty reprezentujúce stromovú štruktúru v GP („stromové“ objekty).

4.1.1 Základné objekty

Základné objekty vytvárame už počas načítavania informácií o bielkovine zo súboru, ktorý celú bielkovinu popisuje. Poďme si ich zadefinovať a popísať ich úlohu.

[1]ProSSiGen - skratka pre ***P**rotein **S**tructure **Si**milarity Using **Gen**etic Programming.*

PdbAtom

Snáď najelementárnejší objekt, ktorý popisuje konkrétnu molekulu, jej názov a trojdimenzionálne súradnice. Ďalej obsahuje identifikátor (ID) a názov rezídua, ku ktorému sa viaže a tiež označenie bielkovinového reťazca[2].

PdbResidue

Objekt reprezentujúci rezíduum, ktorý obsahuje svoje ID, jedno-písmenkový názov aminokyseliny, ktorej je súčasťou a vektor (myslíme tým zoznam implementovaný ako *std::vector*) ukazovateľov na atómy patriace tejto aminokyseline. Rezíduum vnímame v tomto objekte ako ukazovateľ na atóm, ktorý je rezíduum.

ASphere

Okolo každého rezídua si predstavme sféru s nejakým polomerom a stredom, ktorý reprezentuje konkrétne rezíduum. Do tejto sféry ďalej zahrňme rezídua, ktorých všetky atómy (resp. celá aminokyselina) priestorovo patria do tejto sféry. Uvažujme len rezíduá, ktoré sa nachádzajú bezprostredne za, resp. pred stredom tejto sféry. Všetky rezíduá, ktoré predchádzajú stredom sféry a súčasne jej patria, nazveme *upstream backbone* a všetky rezíduá, ktoré po strede sféry v reťazci nasledujú, nazveme *downstream backbone*.

Samotný objekt obsahuje ukazovatele na rezíduá predstavujúce tieto čiastočné reťazce, obsahuje ukazovateľ na rezíduum a tiež premennú vyjadrujúcu, či je sféra typu *null*, kedy ASphere neobsahuje žiadne ukazovatele. Naša ASphere je okresanou verziou *aa-sphere*, ktorú navrhol Galgonek a spol. v metóde SProt [21].

SSE

Už z predošlej kapitoly vieme, že SSE znamená prvok (element), ktorý je sekundárnou štruktúrou. SSE implementujeme ako abstraktný objekt, ktorý obsahuje informáciu o type (*SSEType*), ktorý má hodnotu buď *Helix* — myslíme ním SSE typu α-helix, alebo *Sheet* — SSE typu β-skladaný list, alebo *Loop* — zvyšné časti reťazca, alebo *NIL* — podobne ako v metóde KENOBI [22], kde Szustakowski používal SSE typu *null*. Naše SSE tiež obsahuje počiatočnú a terminačnú ASphere.

Objekty SSE a informácie o nich sa načítavajú priamo zo súboru, t.j. nepoužívame žiadne ďalšie metódy na determináciu sekundárnych štruktúr.

PdbHelix

Objekt predstavujúci α-helix, ktorý je odvodený (implementačne dedí) od objektu SSE. Je určený svojím seriálovým číslom, obsahuje ID rezídua, ktorým štruktúra začína a ID rezídua, ktorým daný α-helix končí - to platí aj o ID reťazca.

PdbSheet

Podobne ako PdbHelix, tak β-skladaný list obsahuje svoje seriálové číslo, počiatočné a koncové ID rezídua. Navyše obsahuje počet vlákien, z ktorých sa list skladá a údaj o smere medzi susednými vláknami - či je ten smer paralelný (rovnaký) alebo anti-paralelný (opačný).

[2] Anglicky: *chain*.

PdbLoop
Tento objekt obsahuje svoje ID a počiatočné, a koncové ID rezídua. Predstavuje časti štruktúry, ktoré nie sú ani α-helixom ani β-skladaným listom.

NilSSE
SSE typu *NIL* je objekt typu SSE (abstraktný objekt) a neobsahuje žiadne rezídua.

ProteinObj
Objekt obsahujúci zoznamy, ktoré uchovávajú ukazovatele na objekty typu PdbAtom, PdbResidue, PdbHelix, PdbSheet, PdbLoop a ASphere. Reprezentujeme ním teda celý proteín s konkrétnym kódom ID.

4.1.2 Dodatočné objekty

ProteinAlignedPair (PAP)
Tento objekt v sebe ukladá ukazovateľ na dva proteíny, ktoré sa najprv sekvenčne zarovnajú. Výsledkom toho je zoznam dvojíc ukazovateľov na objekty typu ASphere[3]. Tieto dvojice sú už zarovnané a spárované.

TraceMatrixField
Pomocná štruktúra, ktorá predstavuje prvok matice, ktorý môže mať viac hodnôt typu *TraceMatrixFieldEnum* (buď je hodnota *Done*[4], alebo *Left*, alebo *Up*, alebo *Diag*). Pri získavaní celého zarovnania (pri procese *backtrack*) nám TraceMatrixField hovorí, ktorými smermi v matici sa môžeme vydať, aby sme získali kompletné zarovnanie.

4.1.3 Stromové objekty

Tieto objekty predstavujú jednotlivé uzly stromu. Spravidla každý objekt sa nachádza na konkrétnej úrovni, pričom strom má nemennú hĺbku.

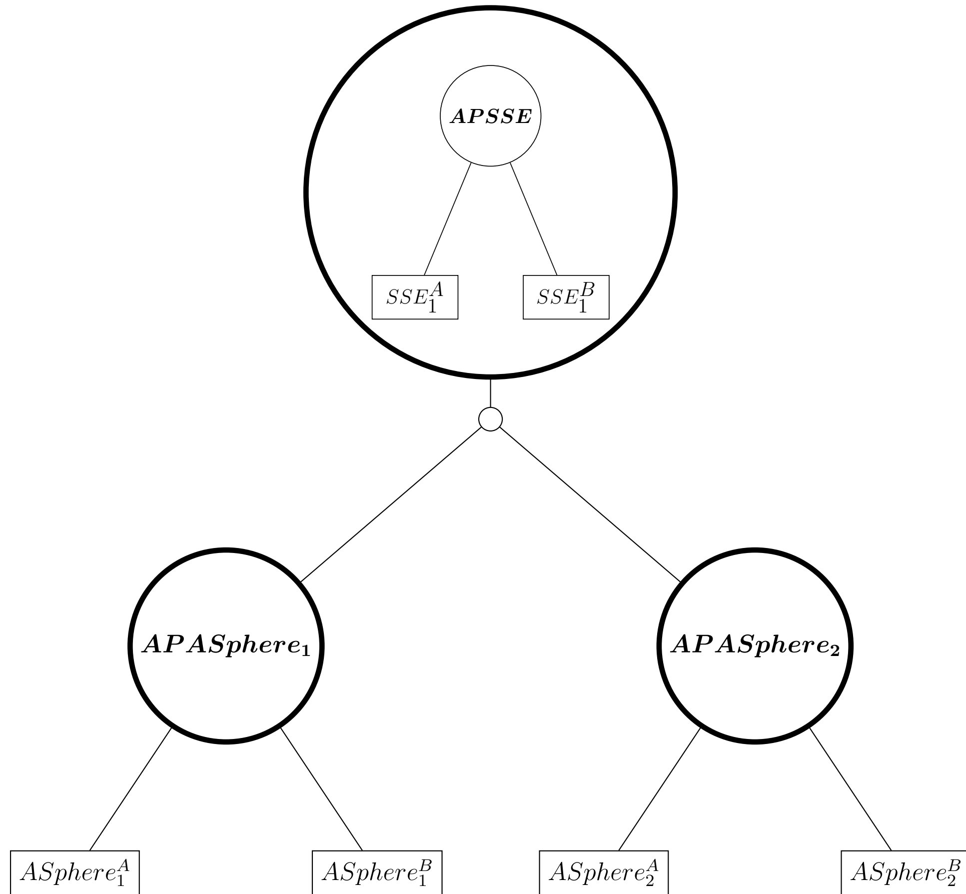
APASphere
Objekt reprezentujúci list stromu, ktorý sa skladá z *prvej* a *druhej* a-sféry (objektu typu ASphere), t.j. sféry patriacej prvej (neklasifikovanej) a druhej (klasifikovanej) bielkovine. Implementačne sa jedná o ukazovatele na tieto a-sféry.

APSSE
Uzol typu APSSE je objekt, ktorý obsahuje vektor vyššie opisovaných listov typu APASphere. Ďalej uchováva dva ukazovatele: na *prvý* a *druhý* objekt SSE.

Vzťah medzi APSSE a APASphere je taký, že *prvá* sekundárna štruktúra je reprezentovaná všetkými prvými AP a-sférami (objekty typu APASphere) z vektoru, ktorý daný APSSE obsahuje. Nápodobne to platí s *druhými* objektami.

[3]Implementačne std::vector<std::pair<ASphere*,ASphere* >>.
[4]Hodnota *Done* sa priraďuje prvku na nulových súradniciach v matici, aby sme vedeli, kedy máme proces *backtrack* ukončiť a vrátiť kompletné zarovnanie.

Tento vzťah znázorňuje obrázok 4.1, na ktorom máme jeden uzol typu APSSE, ktorý obsahuje dva objekty SSE - jeden patriaci bielkovine A, druhý patriaci bielkovine B. Ich index znamená, že sa jedná o prvé objekty typu SSE v bielkovine (sú na začiatku). Na APSSE sa môže napájať ľubovoľné množstvo AP a-sfér, my znázorňujeme prvé dve z celej sekvencie patriacej bielkovine A, resp. B.



Obr. 4.1: Vzťah medzi APASphere a APSSE

APTree

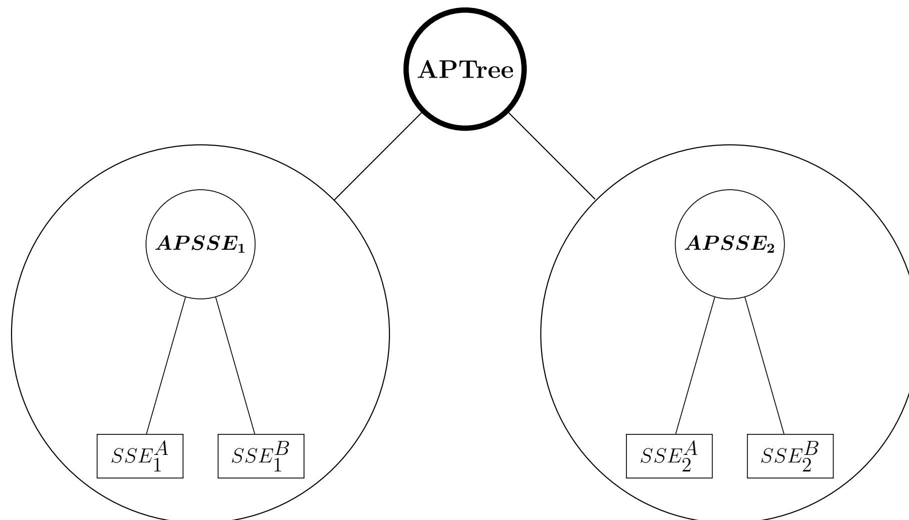
Tento objekt predstavuje koreň celej stromovej štruktúry. Obsahuje vektor ukazovateľov na objekty typu APSSE.

V našom prípade množina funkcií, v stromovej štruktúre predstavujú uzly, ktoré spájajú svojich potomkov (v strome), t.j. tieto uzly majú spájaciu funkciu a sú uzavreté v zmysle, že dovoľujú spájať len uzly s určitým typom (APTree môže spájať len uzly typu APSSE a APSSE môže spájať uzly typu APASphere).

4.2 Proces

Poďme si ďalej popísať priebeh prvých častí celej metódy, jej vstupy i dôležité algoritmy.

K programu je viazaný tiež konfiguračný súbor, v ktorom môžeme nastavovať rôzne parametre týkajúce sa samotného načítavania, globálneho sekvenčného zarovnávania a GP.



Obr. 4.2: Vzťah medzi APTree a APSSE

4.2.1 Vstupy a inicializácia

Program ProSSiGen sa spúšťa cez príkazový riadok, resp. v termináli a má tri vstupné parametre. Prvé dve sú ID porovnávajúcich sa bielkovín a tretí je názov adresára (relatívna cesta k nemu odkiaľ sa program spúšťa), kde sa nachádzajú PDB súbory s týmito bielkovinami. Tento proces sa vykonáva zavolaním metódy *TestReadTwoProteinFiles* v triede *Testing*.

Ku každému ID bielkoviny je potrebné ešte udať ID reťazca. Ako je formát PDB súbory definovaný, máme k dispozícii na stránkach *Protein Data Bank*[5]. Príklad týchto parametrov môže vyzerať napr. takto: „1c3v_ 1kgsb pdb_database\". To znamená, že načítavame bielkovinu s ID 1c3v a ak je piaty znak nie je písmeno, budeme načítavať prvý reťazec (a reťazec); druhú bielkovinu s ID 1kgs a budeme porovnávať reťazec b a popisujúce PDB súbory (s koncovkou *.ent*) sa nachádzajú v adresári pdb_database\.

Po spracovaní parametrov nasleduje načítavanie a vytváranie základných objektov popisovaných v časti 4.1.1, o čo sa stará trieda *LoaderParser*.

Pre podrobnejší opis, ako sa program spúšťa, si odporúčame prezrieť užívateľskú dokumentáciu v prílohe C.

4.2.2 Globálne sekvenčné zarovnanie (GS)

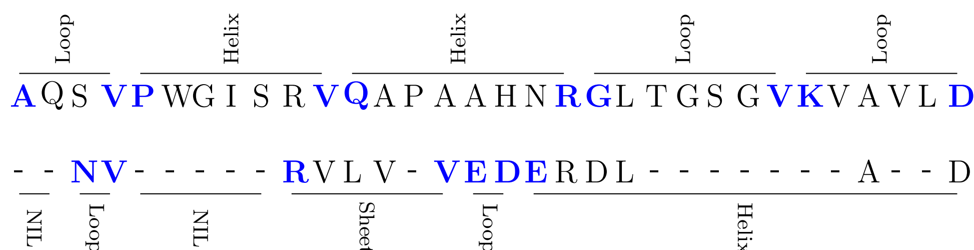
Po načítaní aplikujeme na bielkoviny algoritmus Needleman-Wunsch, ktorý je implementovaný funkciou *AlignTwoProteins* triedy *ProteinComparison*, ktorá vracia vektor ProteinAlignedPair objektov. Typicky ich býva niekoľko, keďže v *backtrace* matici môže existovať viac ciest. Nad týmito vektormi dokončíme definíciu SSE objektov a všetkým a-sféram priradíme konkrétny SSE, ktorého sú súčasťou.

Pripomeňme si funkciu $\psi(a_i, b_j)$ vracajúcu hodnotu podobnosti aminokyselín a_i a b_j. Pre každú dvojicu aminokyselín máme k dispozícii tabuľku BLOSUM62 [25], v ktorej sa tieto hodnoty nachádzajú. Táto tabuľka sa odporúča pri ohodnocovaní používať spolu s Ω o hodnote -4. Ak nebude v konfiguračnom

[5]http://www.wwpdb.org/documentation/format33/v3.3.html

súbore zadaná, potom sa použije obyčajné ohodnocovanie, ktoré hovorí, že ak sa dve aminokyseliny zhodujú, ohodnotíme ho číslom 1, inak je to 0.

Existujú iné BLOSUM tabuľky a vyjadrujú predpokladanú evolučnú vzdialenosť porovnávaných proteínových sekvencií. Výsledok tohto zarovnania je vo veľkej miere závislý od týchto „skórovacích“ matíc.



Obr. 4.3: Ilustrácia objektu ProteinAlignedPair s vyznačenými typmi SSE

Ako môžeme vidieť na obrázku 4.3, všetky SSE okrem typu *NIL* majú svoju počiatočnú a koncovú a-sféru, ktoré sú vždy ne-*null*. A-sféry sú zobrazené jednopísmenovým názvom aminokyseliny a v prípade *null* a-sféry je miesto písmena čiarka.

4.2.3 Prevod PAP na APTree

Po globálnom zarovnaní a vytvorení objektov PAP potrebujeme tieto objekty prekonvertovať na stromovú štruktúru, konkrétne na typ APTree. Vzhľadom na možnú veľkú dĺžku bielkovinového reťazca bola snaha o zložitosť prevodu rovnajúcej sa $O(n)$, kde n je dĺžka zarovnania PAP.

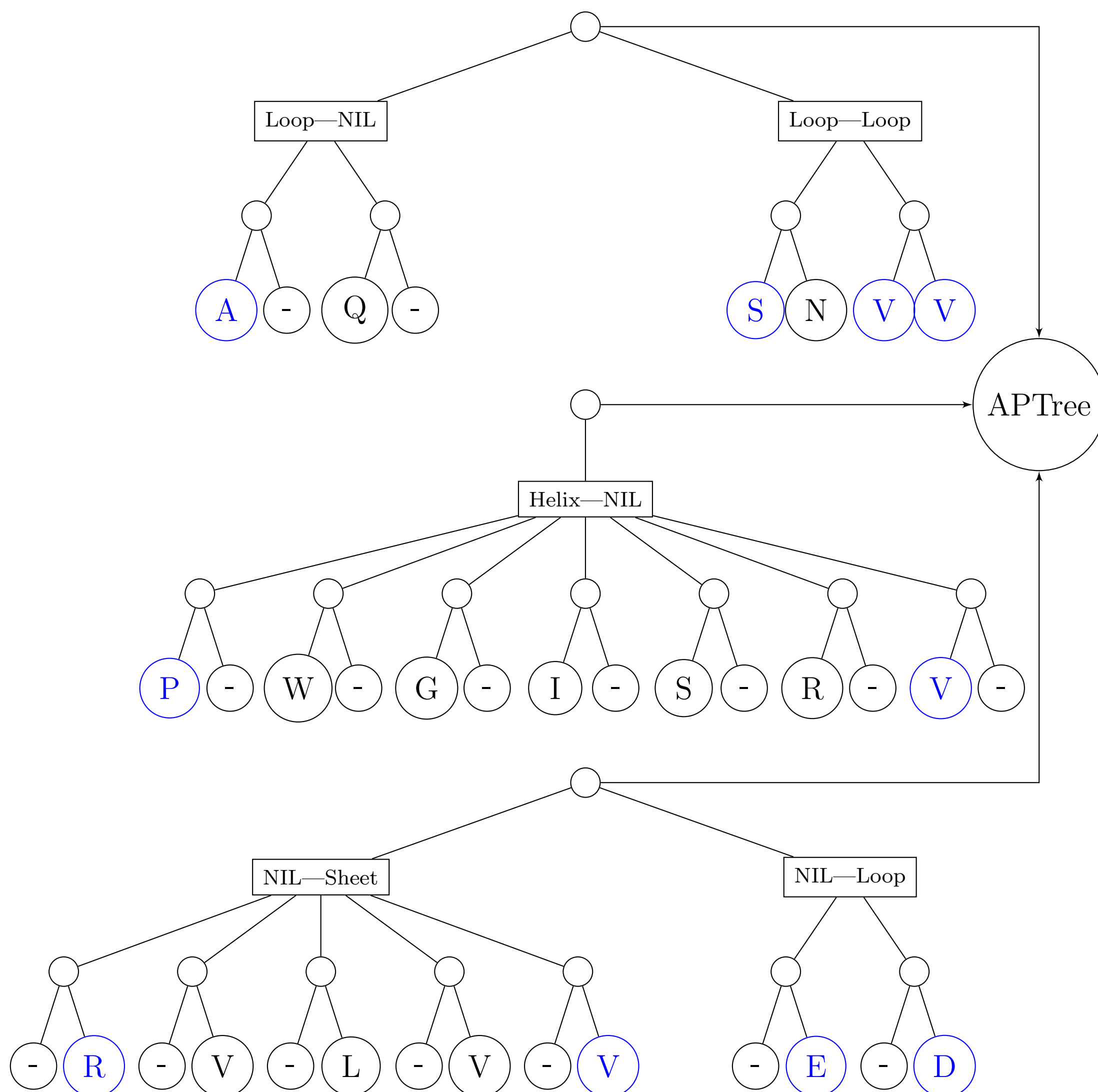
V algoritme postupne prechádzame dvojice a-sfér v zarovnaní PAP, pričom zohľadňujeme faktory ako: či niektorá a-sféra v zarovnaní je začiatkom alebo koncom SSE; či sa konkrétna a-sféra patriaca niektorej bielkovine nachádza v strede SSE (nie je koncom ani začiatkom).

Prevodom sa snažíme zoskupovať SSE rovnakého typu do jedného objektu APSSE. Načrtnime si podstatu prevodu opísaním obrázka 4.4, ktorý je výsledkom prevodu niekoľkých prvých dvojíc a-sfér, resp. niekoľkých prvých AP a-sfér. Prvý objekt APSSE obsahuje v sebe SSE typu *Loop* z prvej bielkoviny a SSE typu *NIL* z druhej bielkoviny. A-sféry *A* a *Q* z prvej bielkoviny patria SSE typu *Loop*, *null* a-sféry patria druhému SSE typu *NIL* (SSE typu *NIL* môžu patriť len *null* a-sféry).

Nasledujúce dve spárované dvojice a-sfér patria rovnakému typu SSE - typu *Loop*. Vytvoríme teda APSSE obsahujúce dva SSE typu *Loop* a do dvoch AP a-sfér vložíme príslušné a-sféry.

Ďalej nasledujú SSE typu *Helix* a *NIL*, celú prvú SSE zlúčime s celou SSE typu *NIL*, pričom k druhej *NIL* SSE pridáme ďalšie dve *null* a-sféry, ktoré dodatočne vytvoríme. Keby sme na mieste druhých a-sfér v posledných dvoch dvojiciach tejto *Helix* SSE mali a-sféry patriace typu *Helix* (miesto typu *Sheet*), vznikol by nový APSSE objekt, kam by sa priradili tieto posledné dve dvojice a-sfér.

Popíšme si komplexne, ako algoritmus funguje. Na vstupe je vektor AP a-sfér, čo sú dvojice a-sfér predstavujúce celý objekt PAP. Pre prechod používame jeden hlavný iterátor, ktorý ukazuje na tieto dvojice. Pri prechádzaní dvojíc, sa



Obr. 4.4: Časť výsledného prevodu zo zarovnania

zaoberáme hlavne tým, či a-sféry, na ktoré iterátor práve ukazuje, predstavujú začiatok nejakého SSE. Špeciálne rozoberáme prípady, či sa jedná o SSE rovnakého alebo rozdielneho typu, alebo sa jedná o SSE typu *NIL*. Ďalej všetky SSE okrem SSE typu *NIL* môžu byť *spárované* už s nejakou SSE rovnakého typu, čo znamená, že sme do stromu vložili APSSE s týmito SSE. Ak narazíme na dvojicu a-sfér reprezentujúce SSE rovnakého typu a ak jedno z SSE je už *sprárované*, potom tieto SSE nebudeme vkladať do jedného APSSE, ale pre už spárované SSE vytvoríme SSE typu *NIL*.

Nech SSE_X znamená SSE typu X a SSE_* znamená SSE ľubovoľného typu SSE okrem *NIL*. Popíšme si jednotlivé prípady, na ktoré druhy SSE reprezentované dvojicou a-sfér môžeme natrafiť (vždy na prvom mieste bude a-sféra z prvej bielkoviny):

začiatok SSE_X vs. začiatok SSE_X

V tomto prípade vytvoríme novú APSSE s SSE o typu X a vložíme všetky dvojice a-sfér, až kým aspoň jedna z nich neukončí dané SSE. Ak prejdeme jedno SSE celé a druhé nie, to druhé označíme ako *spárované*.

začiatok SSE_* vs. začiatok *spárovaného* SSE_X

Všeobecne v tomto prípade do nového APSSE objektu priradíme ako prvé SSE o typu *NIL* a druhé bude SSE_X. SSE_* vkladáme do zásobníku, pre prípad, že SSE_* sa neskôr spáruje s nasledujúcim SSE z druhej bielkoviny.

začiatok *spárovaného* SSE_X vs. začiatok SSE_*

Analogicky ako v predchádzajúcom prípade.

SSE_{NIL} vs. pokračovanie *spárovaného* SSE_*

V tomto prípade iterujeme ďalej, pričom dvojice a-sfér si ukladáme do zásobníka. Nakoniec môžeme natrafiť na jedno z troch nasledujúcich situácií:

začiatok SSE_* vs. pokračovanie *spárovaného* SSE_*

Vyprázdnime zásobník plný APSSE, ktoré vložíme do stromu a potom vložíme nové APSSE s SSE o type *NIL* a X.

SSE_{NIL} vs. koniec *spárovaného* SSE_*

Tu len vyprázdnime zásobník plný APSSE, ktoré vložíme do stromu.

začiatok SSE_* vs. koniec *spárovaného* SSE_X

Prakticky rovnaký prípad ako prvý.

pokračovanie *spárovaného* SSE_* vs. SSE_{NIL}

Obdobne ako predchádzajúci prípad.

začiatok SSE_Y vs. začiatok SSE_X

Vytvoríme si dve APSSE, v jednom sa budú nachádzať SSE o type Y a NIL, v druhom SSE o type NIL a X. Postupne do APSSE vložíme príslušné AP a-sféry. Ak bude jedna z SSE končiť, t.j. natrafíme na terminačnú a-sféru, APSSE s touto SSE vložíme do stromu. Ak skončia na jednom mieste obe SSE (natrafíme na miesto, kde sú obe a-sféry terminačné), najskôr do stromu vložíme APSSE s prvou SSE a potom druhú APSSE.

začiatok SSE_X vs. pokračovanie SSE_X

Vytvoríme APSSE, do ktorého vložíme oba SSE_X a vložíme ho do stromu. Až doiterujeme na miesto nejakej terminačnej a-sféry, potom druhé SSE_X označíme ako *spárované*.

pokračovanie SSE_X vs. začiatok SSE_X

Podobne ako v predošlom prípade.

SSE_{NIL} vs. pokračovanie SSE_*

V tomto prípade iterujeme a vkladáme do zásobníka AP a-sfér dvojice a-sfér odkazujúce sa na SSE typu *NIL* a SSE_* až dokiaľ nenarazíme na nejaký začiatok (SSE, ktoré by nasledovalo po SSE typu *NIL*), resp. koniec SSE_*.

pokračovanie SSE_* vs. SSE_{NIL}

Analogicky podľa predošlého prípadu.

začiatok SSE_Y vs. pokračovanie SSE_X

Vytvoríme APSSE o SSE typoch Y a *NIL* a vložíme do zásobníka APSSE. Ak natrafíme na koniec SSE_Y, vyprázdnime zásobník APSSE. Ak súčasne

skončí aj SSE_X, najprv vložíme APSSE s SSE_X a až potom vyprázdnime zásobník. V prípade, že SSE_Y končiť nebude, do stromu vložíme len APSSE s SSE_X.

pokračovanie SSE_X vs. začiatok SSE_Y
Podobne ako v predchádzajúcom prípade.

začiatok SSE_* vs. SSE_{NIL}
V tomto prípade iterujeme a vkladáme do zásobníka AP a-sfér dvojice a-sfér odkazujúce sa na SSE typu SSE_* a *NIL* až pokým nenarazíme na nejaký začiatok (SSE, ktoré by nasledovalo po SSE typu *NIL*), resp. koniec SSE_*. V prípade začiatku resp. konca pokračujeme ďalej v iterácii, inak v prípade, kedy nastanú obe situácie, vyprázdnime všetky zásobníky, vložíme APSSE do stromu aj s novo vytvoreným APSSE obsahujúce AP a-sféry zo zásobníka.

SSE_{NIL} vs. začiatok SSE_*
Obdobne podľa predošlého opisu.

Tento celý algoritmus spúšťame v niekoľkých vláknach, podľa hodnoty v konfiguračnom súbore. Každé vlákno obstaráva a konvertuje jeden objekt PAP na APTree, ktorý si zoberie zo spoločného zásobníka, kde sú uložené objekty PAP.

4.3 Genetické programovanie v ProSSiGen

V tejto sekcii si opíšeme priebeh GP v programe. Implementačne sa o priebeh stará trieda *GP*. Algoritmus 4.3.1 nižšie ilustruje princíp chodu GP v našej metóde.

Algoritmus 4.3.1: Proces GP pri porovnávaní

```
Nech t = 0 je počítadlo generácií;
Do prvotnej populácie C(t) zaradíme všetky APTree objekty;
while terminačná podmienka neplatí do
    Pre každého jedinca x_i(t) spočítajme hodnotu fitness f(x_i(t));
    Zoraďme jedincov v populácii podľa fitness;
    Vytvorme nových potomkov, na ktorých bol aplikovaný operátor
    mutácie a vložme ich do novej populácie C(t + 1);
    Vyberme novú populáciu C(t + 1);
    Posuňme sa na ďalšiu generáciu, t.j. t = t + 1;
end
```

Po tom, čo sme previedli všetky objekty PAP na objekty APTree, sme tiež získali jedincov v počiatočnej populácii. V prípade, že by týchto jedincov bolo málo (menej ako 10), je možné nechať niektorých rodičov „prežiť“ a presunúť do novej populácie. Tým by sme zvýšili počet jedincov v populácii, získali by sme tým viac štruktúrnych variácií. Avšak, zatiaľ nám postačia noví jedinci.

4.3.1 *Fitness* a selekčná funkcia

Pre každú novú generáciu jedincov vrátane počiatočnej generácie musíme spočítať hodnotu *fitness* všetkých indivíduí. Máme niekoľko možností, aké ohodnotenie použiť, no budeme používať funkciu RMSD[6]:

$$RMSD(x_i) = \sqrt{\frac{\sum_{j=1}^{M_{x_i}} (a_j - b_j)^2}{M_{x_i}}} \tag{4.3.1}$$

Funkcia 4.3.1 $RMSD(x_i)$ pre jedinca x_i vráti jeho *fitness* hodnotu. M_{x_i} znamená počet spárovaných ne-*null* a-sfér; a_j je j-ta spárovaná ne-*null* a-sféra prvej (neklasifikovanej) bielkoviny; podobne pre b_j.

Výpočet prebieha vo viacerých vláknach v zmysle, že jedno vlákno procesu počíta *fitness* pre jedného jedinca a o výpočet RMSD sa stará metóda *ComputeFitness* z triedy *Scoring*.

Po spočítaní *fitness* všetkých jedincov v populácii ich zoradíme podľa tejto hodnoty, aby sme v ďalšej časti mohli použiť selekčnú metódu *Linear Rank Selection* opisovanú Thomasom Bäckom [26].

Majme generáciu $C(t) = \{x_0(t), x_1(t), \ldots, x_{\mu-1}(t)\}$ s jedincami o počte μ, ktorú sme zoradili od „najlepšieho" jedinca po „najhorší", t.j. jedinec $x_0(t)$ v populácii $C(t)$ bude mať najlepšiu hodnotu *fitness*. *Linear Rank Selection* spočíva v tom, že jedincov pre kríženie alebo reprodukciu vyberáme na základe rebríčka a prednosť vo výbere dávame lepšie ohodnoteným jedincom. Výhoda je, že ak majú dvaja jedinci čo len minimálny rozdiel v ohodnotení, tento rozdiel sa prejaví rovnomerne medzi všetkými jedincami. Rovnako ak by v populácii existovali indivídua s veľmi dobrým ohodnotením, pri iných selekčných metódach by mali tieto indivídua veľmi veľkú prevahu v populácii, ale náš typ selekcie tento efekt eliminuje.

Whitley [27] predstavil vo svojej práci vzorec 4.3.2, pomocou ktorého priamym výpočtom získame index jedinca za požitia tejto selekčnej metódy.

$$i = \left\lfloor \frac{\mu}{2(c-1)} \left(c - \sqrt{c^2 - 4(c-1)\chi} \right) \right\rfloor \tag{4.3.2}$$

V ňom Whitley používa dve premenné: c, čo je *selective pressure* parameter, pre ktorý musí platiť $1 < c \leq 2$ a čím je hodnota bližšia k 2, tým budeme častejšie vyberať „lepších" jedincov; χ je reálne náhodné číslo, pričom $\chi \in [0; 1]$. Typicky má c hodnotu 1.8. Táto metóda je implementovaná funkciou *PerformLinearRankSelection* triedy *GPsUtility*.

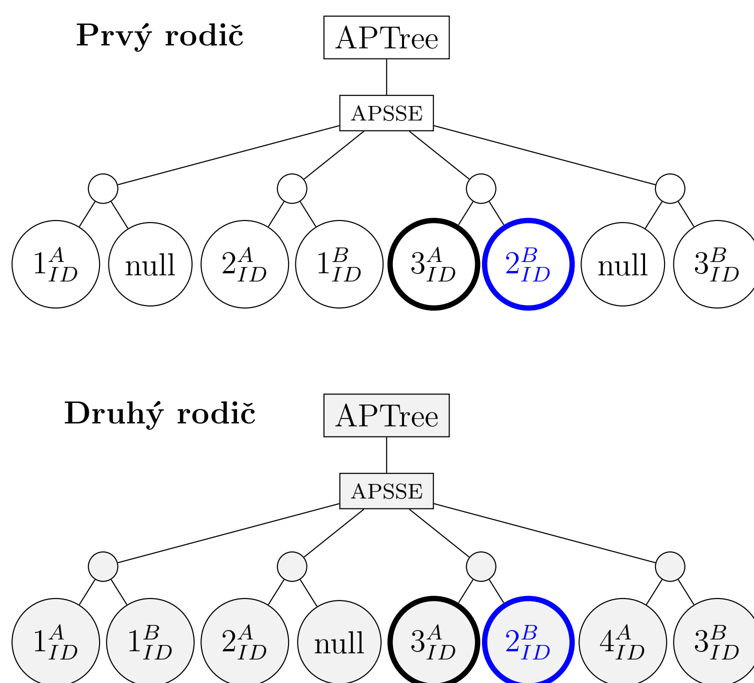
4.3.2 Kríženie objektov APTree

Ohodnotenú a zoradenú populáciu rozdelíme na sub-populácie, ktorých počet závisí od počtu vlákien. V bode, kedy sa máme rozhodnúť, či budeme reprodukovať, alebo krížiť, za nás rozhodne parameter p_r a p_c konfigurovateľný v konfiguračnom súbore. Ak sa rozhodne, že budeme reprodukovať, pomocou selekčnej metódy vyberieme jedinca, ktorého zreprodukujeme do novej populácie.

[6] Root-Mean-Square Deviation.

V prípade kríženia vyberieme pomocou *Linear Rank Selection* dvoch jedincov, na ktorých zavoláme funkciu *PerformCrossover*, ktorá vratí dvoch potomkov. Pri krížení je dôležité vybrať správny náhodný bod v štruktúre oboch rodičov. Rodičia, ktorí sú reprezentovaný objektom APTree, obsahujú dve *mapy*[7] s názvom *FirstPairingResiduesIDs* a *SecondPairingResiduesIDs*. Prvá *mapa* má ako kľúče identifikačné čísla rezíduí prvej bielkoviny, ktorých hodnota je ID rezídua druhej bielkoviny, ktoré je s ním v APTree spárované. Obdobne to platí aj pre druhú *mapu*.

Pri výbere náhodného bodu kríženia, si určíme kľúč k_1 na náhodnej pozícii i v *mape* prvého rodiča s hodnotou h_1. Následne sa pozrieme, či v *mape* druhého rodiča na pozícii i existuje kľúč k_2 a zistíme si hodnotu, na ktorú ukazuje — označme ju h_2. Ak sa obe hodnoty h_1 a h_2 rovnajú, bod kríženia nám vyhovuje. Mohlo by sa tiež totiž stať, že rezíduum s ID k_2 by bolo spárované len s *null* a-sférami, ktorých ID sa do týchto *máp* nezaznamenávajú.

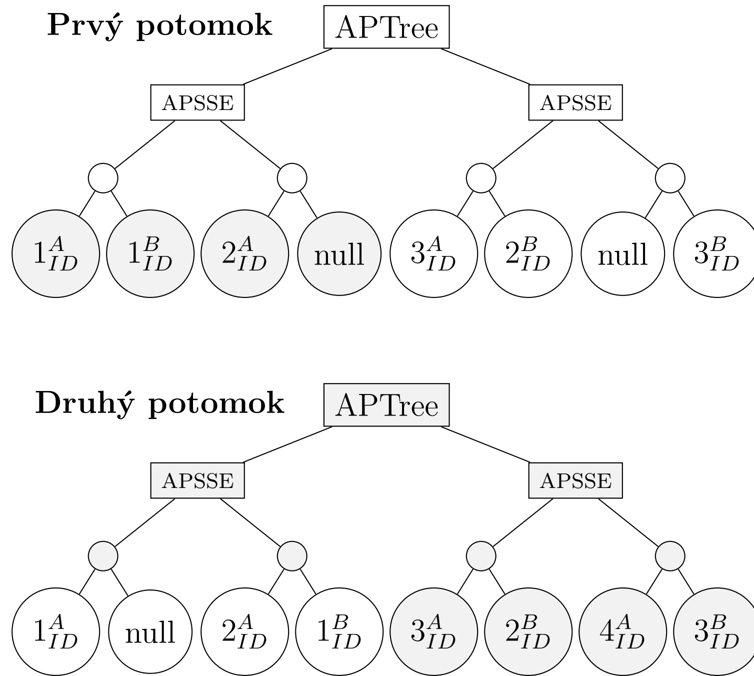


Obr. 4.5: APTree objekty ako rodičia s bodom kríženia

Vo väčšine prípadov sa pri krížení hodnoty kľúčov (ID rezíduí) k_1 a k_2 rovnajú, čo je ideálny prípad. Ak sa hodnoty nerovnajú, vznikne v potomkoch náhodná mutácia, ktorej výsledok môže byť napr. duplikácia niektorej a-sféry za odstránenia inej a-sféry. Takáto náhodná mutácia je súčasťou princípu evolučných algoritmov.

Ako ďalší následok kríženia môže byť tiež rozdelenie jedného APSSE na dve APSSE objekty ako vidíme na obrázku 4.6. To dáva väčšie možnosti pre mutačný proces v nasledujúcej sekcii.

[7] Implementačne std::map<int,int>.



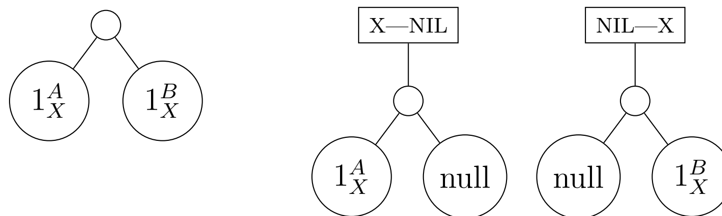
Obr. 4.6: APTree objekty ako potomkovia

4.3.3 Mutácia nových jedincov

Proces mutácie, ktorý aplikujeme na všetkých nových jedincov zaručuje zmenu usporiadania a párovania a-sfér. Na začiatku tohto procesu si v jedinci vyberieme dva susedné APSSE objekty z vektoru objektov APSSE, ktorý obsahuje koreň APTree stromu. Podľa toho, aké vlastnosti majú SSE v týchto susedných APSSE objektoch, vykonáme iné usporiadanie. Budeme sa riadiť podľa týchto vzorov:

APSSE(SSE_X;SSE_X) vs. APSSE(SSE_X;SSE_X)

V tomto prípade, kedy dva susedné APSSE objekty obsahujú SSE rovnakých typov, si vyberieme posledný objekt AP a-sfér z prvého APSSE (z vektoru objektov APASphere), alebo prvý objekt AP a-sfér z druhého APSSE. Túto AP a-sféru rozdelíme na dve nové AP a-sféry a to tak, že prvú a-sféru spárujeme s novou *null* a-sférou a podobne spárujeme druhú a-sféru s novou *null* a-sférou. S nimi vytvoríme aj nové objekty APSSE, do ktorých ich zaradíme. Vzor výsledku by mal vyerať ako je ilustrované na obrázku 4.7.



Obr. 4.7: Vznik nových AP a-sfér a objektov APSSE

APSSE(SSE_X;SSE_X) vs. APSSE(SSE_X;SSE_{NIL})

Keď tri SSE objekty sú rovnakého typu a jeden SSE je typu *NIL*, vyberáme

poslednú AP a-sféru z prvého objektu APSSE (keby boli objekty APSSE vo vzore v opačnom poradí, vybrali by sme prvú AP a-sféru z druhého APSSE objektu). Túto AP a-sféru tiež rozdelíme na dve nové AP a-sféry a jednu z nich (tú, ktorá korešponduje s poradím typov SSE objektov) vložíme do druhého APSSE objektu (resp. do vektoru AP a-sfér), kde je jeden SSE objekt typu *NIL*.

Obdobné postupy aplikujeme aj pri ostatných permutáciách (ďalších troch), kde jeden SSE objekt je typu *NIL*.

APSSE(SSE_X;SSE_{NIL}) vs. APSSE(SSE_{NIL};SSE_X)
Ak proti sebe máme postavené APSSE objekty s SSE objektami o typoch X a *NIL* vs. *NIL* a X, vyberieme poslednú AP a-sféru z prvého APSSE a prvú AP a-sféru z druhého APSSE, a spojíme ich dokopy s vytvorením nového objektu APSSE s SSE o typoch X a X.

APSSE(SSE_{NIL};SSE_X) vs. APSSE(SSE_X;SSE_{NIL})
Tu postupujeme rovnako ako v predchádzajúcom prípade.

4.3.4 Terminačné kritérium

Evolučný proces sa zastaví na generácii $C(t)$ alebo ak sa v populácii nájde jedinec, ktorý spĺňa určitú hranicu ohodnotenia. Počet generácií aj hraničné ohodnotenie môžeme meniť v konfiguračnom súbore.

Kapitola 5

Experimenty

V predchádzajúcej časti sme si navrhli a naimlementovali metódu ProSSiGen. Pre zhodnotenie chodu a presnosti metódy je proces testovania metódy nevyhnutný. V tejto kapitole si popíšeme metodiku testovania, z daných výsledkov určíme presnosť metódy, rozanalyzujeme výsledky a samotnú metódu, a riešenia pre zlepšenie.

5.1 Metodika testovania

Po návrhu a implementácii novej metódy typicky nasleduje krok determinácie úspešnosti danej metódy. Podstatou tohto kroku je automatizovaná klasifikácia proteínových štruktúr, ktorá sa následne skontroluje podľa manuálnej klasifikácie SCOP.

Na testovanie použijeme množiny dát, na ktorej sa testovala metóda Vorolign [28]. Za účelom určenia presnosti tieto množiny ďalej využívali aj mnohé iné metódy.

Spomínané množiny proteínových štruktúr sú dané výťahom databázy ASTRAL-25 v1.65 [29], ktorá obsahuje 4357 štruktúr. Množina, ktorá obsahuje štruktúry, ktoré budeme klasifikovať predstavuje rozdiel SCOP databáz verzií v1.67 a v1.65 a je tvorená 979 štruktúrami. V našom kontexte ich budeme rozlišovať ako klasifikované a neklasifikované štruktúry. Pre dvojice klasifikovaná-neklasifikovaná štruktúra platí, že ich sekvencie sú si podobné na najviac 25%.

Ďalej sa zameriame na vplyv genetického programovania pri porovnávaní konkrétnej dvojice štruktúr. Vyberieme si dvojicu štruktúr, na ktorej budeme sledovať vplyv zmeny jednotlivých parametrov pre algoritmy GP.

Testovanie prebiehalo na počítači s konfiguráciou 2xCPU Xeon X5660 2.8GHz, 24GB RAM, 500GB WD Velociraptor vo virtualizovanom OS Debian 6.0.6. Nakoľko sú testy časovo náročné, pre každú porovnávajúcu dvojicu štruktúr sme nastavili maximálny čas porovnávania 15 sekúnd a spustili sme niekoľko desiatok (60–80) inštancií programu, pričom každý využíval práve jedno vlákno procesora.

5.2 Presnosť klasifikácie

Testovanie našej metódy prebiehalo na približne polovici dostupných testovacích štruktúr. Už z týchto výsledkov môžeme dedukovať charakter metódy. Úspešnosť

klasifikácie triedy sa pohybuje na úrovni približne 35%, na úrovni rodu sú to 4%, na úrovni nadrodiny, rodiny a celkovej zhody sú to len 2%.

V tabuľke nižšie 5.1 môžeme vidieť percentuálne úspešnosti iných metód, ktorých dáta sme použili z prác od Hokszu a Galgonka [30] [21]. Jedná sa o metódy SProt [21], db-iTM [30], Vorometric-TM [31], Vorolign [28] a BLAST [32].

Naša metóda teda veľmi zaostáva oproti napr. BLAST, čo je metóda využívajúca striktne sekvenčné podobnosti. Oproti sofistikovanejším metódam ako SProt, Vorolign [1], db-iTM a Vorometric-TM, naša metóda ešte viac zaostáva. Posledné dve vymenované metódy sú založené na tzv. TM-score [33], ktoré vo väčšine prípadov predstavuje relevantnejšie ohodnotenie ako napr. RMSD.

Metóda	Rodina	Nadrodina	Rod	Trieda
SProt	90.7	96.9	98.6	-
db-iTM	86.6	95.8	98.2	-
Vorometric-TM	90.7	94.9	97.6	-
Vorolign	86.4	92.4	97.7	-
BLAST	48.9	52.5	52.8	-

Tabuľka 5.1: Úspešnosti klasifikácie metód na jednotlivých úrovniach

5.3 Analýza výsledkov

Pripomeňme, že testovanie metódy prebiehalo len na určitej časti množiny neklasifikovaných štruktúr, takže odchýlka môže mať ešte určitú hodnotu. Ďalej uvážme všetky dvojice štruktúr, ktoré sa porovnávali a fakt, že priebeh evolučného procesu bol negatívne ovplyvnený troma hlavnými bodmi:

- obmedzenou dobou chodu porovnávania jednej dvojice,
- obmedzeným počtom generácií a
- komplexnou implementáciou základných operácií GP.

Pre praktickosť testovania sa zvolila maximálna doba porovnávania jednej dvojice štruktúr (15 sekúnd). Je to kvôli tomu, že vývoj populácie, v ktorej sa nachádzajú jedinci s rozmanitou a zložitejšou štruktúrou, aplikácia kríženia a mutácie môže byť časovo náročnejšia. Inak by sme mohli predpokladať, že po dlhšej dobe a po mnohých vygenerovaných populáciách by sme získali veľmi dobré výsledky.

Typická situácia, ktorá nastáva v GP, je, že po určitej dobe algoritmus nie je schopný vyprodukovať „lepších“ jedincov, t.j. hodnota fitness konverguje. To normálne nastáva po určitom počte generácií. V teste sme použili hornú hranicu 2000 generácií. Tiež platí, že pre každú dvojicu táto hranica by mala byť odlišná (kvôli rozmanitosti štruktúr), no vopred túto hranicu pre každú dvojicu určiť nevieme.

[1] Vorolign je metóda využívajúca dynamické programovanie na globálnej úrovni za použitia ohodnocovacích matíc a využívaním princípov Voronových diagramov.

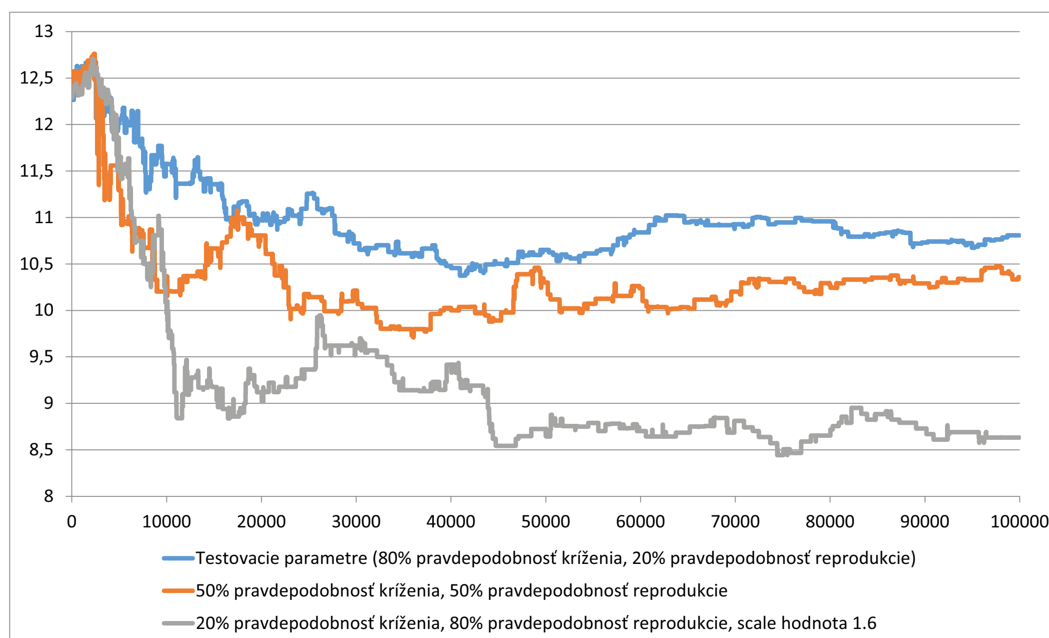
Komplexnou implementáciou základných operácií GP rozumieme obšírne preskúmanie možností operácií GP, ktoré sú aspoň dostatočne implementované. Pri algoritmoch GP niekedy stačí doimplementovať práve jednu operáciu na to, aby sa výsledky evolúcie zlepšili rádovo o desiatky percent. Je teda isté, že spôsoby v našej metóde nie sú dostatočné aj napriek tomu, že sme vo všeobecnosti pokryli nemalú časť možností.

Ako následok prvých dvoch vyššie spomínaných bodov je, že všetky generácie sa nemusia stihnúť vyvinúť. V našom testovaní je to až 85% percent. To znamená, že algoritmy GP potrebujú viac času alebo ďalšie rozšírenia a implementáciu detailov.

5.4 Vplyv parametrov genetického porovnania

V testovaní sme používali nasledovné hodnoty dôležitých parametrov:
počet generácií: 2000,
pravdepodobnosť kríženia: 80%,
pravdepodobnosť reprodukcie: 20% a
parameter Selective pressure: 1.8.

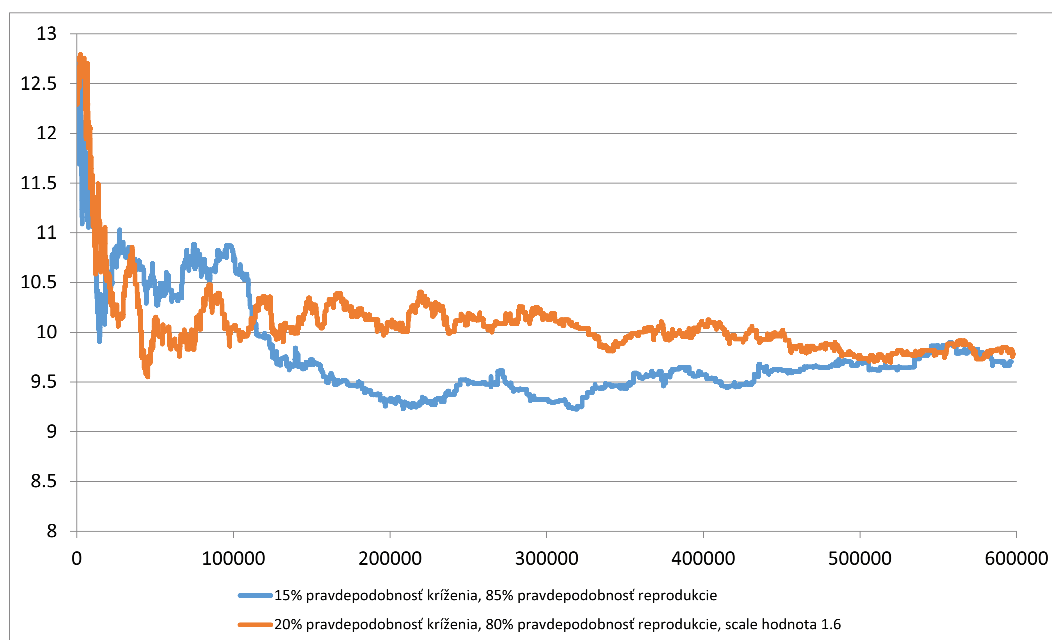
Na grafe 5.1 znázorňujúcom priebehy hodnoty *fitness* vzhľadom na počet vygenerovaných populácií nám na danej dvojici poskytuje prehľad dôvodov, ktoré dokazujú, že zarovnanie proteínových štruktúr sa dokáže zlepšovať.



Obr. 5.1: Porovnanie priebehu RMSD v evolúci zarovnania dvojíc štruktúr 1OYIa a 1ADRa počas 100 000 generácií

Môžeme pozorovať, že po určitej generácii sa nám hodnota RMSD ustáluje a výrazne sa už nemení. Najväčší skok v hodnôt nastáva po 10000. generácii. Ďalej môžeme vidieť, že obmedzenie kríženia jedincov nám pomáha pri ohodnocovaní. Z toho môžeme vyvodiť, že kríženie v našich stromových štruktúrach niekedy zbytočne generuje nevhodné párovania.

Uvedomme si, že vykonané testy prebiehali len na najviac 2000 generáciách, čo nie je postačujúce. Avšak z časového hľadiska nie je veľmi prípustné, aby nová metóda potrebovala veľa času na kvalitné výsledky.



Obr. 5.2: Porovnanie priebehu RMSD v evolúci zarovnania dvojíc štruktúr 1OYIa a 1ADRa počas 600 000 generácií

Priebeh hodnoty RMSD na grafe 5.2 tiež značí, že rapídnejšia zmena v hodnote RMSD môže nastať aj po 100 000. generácii — ako vidíme na modrej krivke. V takom prípade by algoritmy GP potrebovali ešte viac času na to, aby vyprodukovali adekvátne zarovnanie.

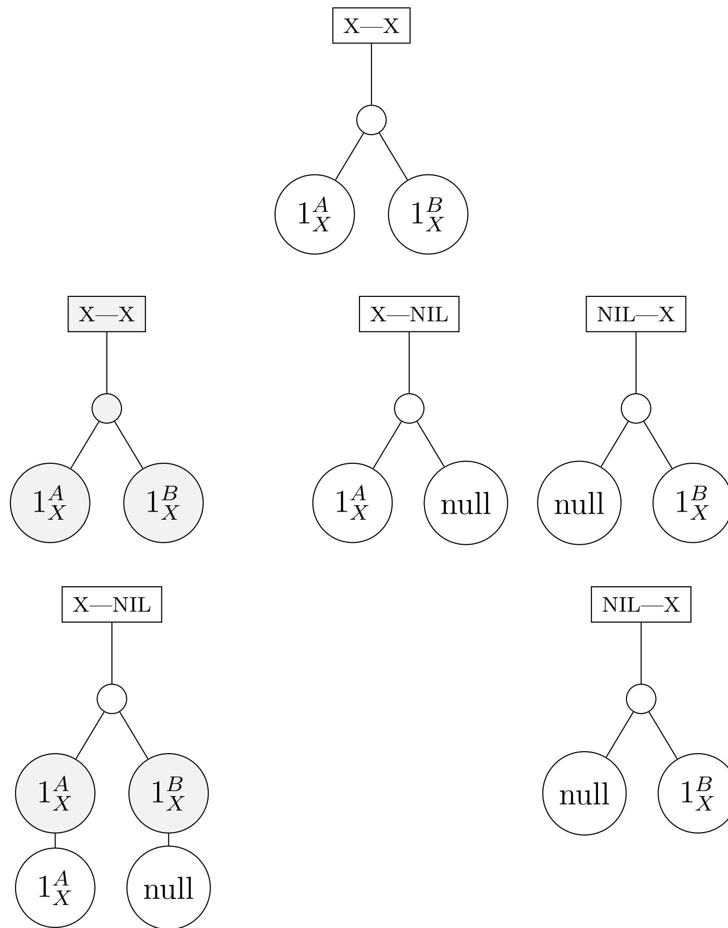
5.5 Návrhy rozšírenia

Prvý návrh rozšírenia by mohol predstavovať sofistikovanejšie ohodnocovanie stromovej štruktúry a celého jedinca. Predstavme si, že každá dvojica spárovaných a-sfér by mohla mať vo svojom spoločnom koreni hodnotu podobnosti. Spočítali by sme ju využívajúc ostatné rezíduá patriace do týchto a-sfér, ktoré sa nachádzajú na *upstream* a *downstream backbone*. Podobným štýlom by sme mohli ohodnocovať jednotlivé spárované dvojice SSE objektov a hodnotu uložiť do APSSE.

Ohodnocovanie jednotlivých uzlov by sa dalo následne využiť pri operácii náhodnej mutácie. Ak by ohodnotenie dvojice objektov bolo dobré, náhodná mutácia by mala menšiu šancu na uplatnenie. Naopak, horšie ohodnotenie dvojice objektov by ponúklo uplatnenie na náhodnú zmenu, resp. preusporiadanie.

Druhý návrh by spočíval v implementácii mutačných operácií, ktoré by preusporiadavali dvojice SSE objektov v rámci susedných APSSE objektov. Táto možnosť by mala rapídne zlepšiť evolučný priebeh.

Posledný návrh nesúci veľký význam a nové možnosti by bol pridanie ďalšej úrovne do stromovej štruktúry. Predstavme si aplikáciu mutačnej operácie, kedy dvojicu a-sfér odstránime z vektora AP a-sfér. Miesto zmazania by sme ich len



Obr. 5.3: Vznik novej úrovne v stromovej reprezentácii

skopírovali a pôvodné a-sféry by sme zmenili na zderivovaný objekt, ktorý by niesol informáciu, že dotyčné a-sféry sa ohodnocovaní jedinca nemajú brať do úvahy. Tieto a-sféry by sa v skutočnosti tvárili, akoby tam neboli. Na obrázku 5.3 zderivované a-sféry a jej príslušný APSSE objekt majú vyfarbený uzol šedou farbou. Po tejto zmene by mohla nastať situácia, kedy by sme chceli do vektora napojiť susedné a-sféry. Miesto klasického napojenia do vektoru by sa však susedné a-sféry napojili priamo na derivované a-sféry a vytvorili by tak novú úroveň.

Nová úroveň stromu by takto niesla informáciu, ako stromová štruktúra vyzerala v minulosti. Na základe tohto poznatku by sa dali vyvodzovať ďalšie závery a napr. aj sofistikovanejšie ohodnotenia celého jedinca. Vytvorením novej úrovne sa začíname približovať k lamarkizmu. Tento smer môže priniesť ďalšie zlepšenia v evolučnom procese.

Zo spomenutých navrhnutých riešení by implementácia pridanej úrovne bola možno príliš zložitá, resp. réžia zostavovania stromovej štruktúry by bola náročná a preto je otázne, či by toto rozšírenie bolo vhodné. Navyše sme boli limitovaní výpočtovými zdrojmi, čo by nám ešte viac sťažovalo priebeh testovania. Celému riešeniu by najviac pomohla implementácia mutačných operátorov, ktoré by pracovali na úrovni SSE.

Záver

V práci sme si predstavili bielkovinu ako dôležitý biologický prvok. Vzhľadom na ich zvyšujúci sa počet v proteínových databázach sa vynárala otázka efektívneho spôsobu klasifikácie neznámej, resp. neklasifikovanej bielkoviny. Nakoľko sa klasifikácia proteínovej štruktúry vykonáva pomocou hľadania podobnosti medzi bielkovinami, zadali sme si túto úlohu aj my, avšak zo zámerom využiť genetické programovanie využívajúce stromovú reprezentáciu.

Predstavili sme všeobecnú dogmu genetického programovania, opísali typické algoritmy pre sekvenčné zarovnávanie, niekoľko známych metód pre štrukturálne zarovnanie a bližšie si opísali známe dve metódy štrukturálneho zarovnania využívajúce genetické algoritmy. Ďalej sme si popísali smer, ktorým je dobré sa pre návrh novej metódy vydať a uviedli sme metódu ProSSiGen.

V našej metóde sme najprv využili algoritmus globálneho sekvenčného programovania pre počiatočné zarovnanie. Následne sme zarovnanie previedli na stromové štruktúry reprezentujúce jedinca v populácii v evolučnom procese genetického programovania. Prevod sa zakladal na párovaní rovnakých typov sekundárnych štruktúr v zarovnaní. Naimplementovali sme operáciu kríženia a operáciu mutácie zahrňujúcu 7 typov preusporiadania časti štruktúry jedinca. Použili sme metódu *linear rank selection* pre výber jedinca, o ktorom sa rozhodovalo, či sa bude krížiť, alebo reprodukovať. Operáciu mutácie sme aplikovali na všetkých jedincov, pretože z hľadiska párovania je potrebné prejsť čo najviac kombináciami párovania jednotlivých a-sfér, resp. rezíduí. Ako *fitness* funkciu určovaciu podobnosť štruktúr sme použili funkciu RMSD.

Z výsledkov experimentov môžeme vyvodiť niekoľko záverov. Napriek tomu, že presnosť našej metódy je v porovnaní s ostatnými známymi slabá a v metóde sa vo veľkej miere vyskytujú negatívne vplyvy, sme mohli vidieť, že genetické programovanie dokáže párovanie vylepšovať. Charakter algoritmov genetického či evolučného programovania je taký, že ich niekedy stačí pre danú úlohu rozšíriť o jednu vhodnú operáciu nato, aby evolučný proces dokázal úspešne a efektívne úlohu vyriešiť. Nájdenie vhodnej operácie, resp. množiny vhodných operácií, by však vyžadovalo ešte hlbšie preskúmanie nesúce potrebu implementácie a testovania návrhu.

Na začiatku pri výbere vhodnej metódy sme považovali stromovú reprezentáciu pri genetickom programovaní za veľkú výhodu. Žiaľ, ukázalo sa, že táto reprezentácia nesie so sebou aj nevýhodu a to veľkú réžiu udržovania stromovej štruktúry. Tento fakt sa podpísal na náročnosti implementácie.

Využitie metód založených na GP pri porovnávaní proteínových štruktúr netreba zatratiť, no len ďalej testovať a implementovať ďalšie operácie a detaily, ktoré stromová štruktúra predstavujúca štrukturálne zarovnanie ponúka. Medzi ne patrí napríklad rozšírenie daného stromu o ďalšiu úroveň, ktorá by nás posunu-

la k využitiu lamarkizmu v GP; tiež vylepšenie ohodnocovania *fitness* s využitím jednotlivých úrovní v stromovej štruktúre a implementovanie ďalších mutačných prípadov, ktoré by pracovali nielen na úrovni AP a-sfér, ale aj na úrovni SSE objektov. Týmito ďalšími rozšíreniami by sme sa však dostali už za rozsah tejto práce. Pri implementácii sme mysleli na obšírnejšiu základnú implementáciu a navrhnutí vhodnej stromovej štruktúry.

Zoznam použitej literatúry

[1] Branden, Carl-Ivan a Tooze, John. *Introduction to Protein Structure.* 2. edícia, pp. 3–5, New York: Garland Publishing, 1998. ISBN 0-8153-2304-2.

[2] Buxbaum, Engelbert. *Fundamentals of Protein Structure and Function.* New York: Springer Science+Business Media, LLC., 2007. ISBN 978-0-387-26352-6.

[3] Paolella, Peter. *Introduction to Molecular Biology, First Edition.* Pp. 51–58, USA: The McGraw-Hill Companies, Inc., 1998. ISBN 0-697-20939-3.

[4] Raineri, Deanna. *Introduction to Molecular Biology.* Massachusetts: Blackwell Science, Inc., 2001. ISBN 0-632-04379-2.

[5] Hartl, Daniel L. a Jones, Elizabeth W. *Genetics: Analysis of Genes and Genomes.* 5. edícia, Massachusetts: Jones & Bartlett Learning, 2001. ISBN 0-7637-0913-1.

[6] Whitford, David. *Proteins Structure and Function.* Strana 180, Chichester: John Wiley & Sons Ltd, 2005. ISBN 0-471-49893-9.

[7] Hoksza, David. *Similarity Search in Protein Databases.* Doktorská dizertácia, Katedra softwarového inženýrství Univerzity Karlovy, 2010.

[8] Engelbrecht, Andries P. *Computational Intelligence: An Introduction.* 2. edícia, Chichester: John Wiley & Sons Ltd, 2007. ISBN 978-0-470-03561-0.

[9] Fogel, Gary B. a Corne, David W. *Evolutionary Computation in Bioinformatics.* USA: Elsevier Science, 2003. ISBN 1-55860-79-8.

[10] Koza, John R. *Genetic Programming: on the Programming of Computers by Means of Natural Selection.* 6. edícia, Massachusetts Institute of Technology, 1998. ISBN 0-262-11170-5.

[11] Singh, Amit P. a Brutlag, Douglas L. *Protein Structure Alignment: A Comparison of Methods* [online]. 2000, [cit. 2012-11-15]. `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.33.3694`.

[12] Mount D.W. *Bioinformatics: Sequence and Genome Analysis* . Prvá edícia, New York: Cold Spring Harbor Laboratory Press, 2001. ISBN 978-0879696085.

[13] Needleman, S.B. a Wunsch, C.D. *A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins.* 48(3): pp. 443–453, Journal of Molecular Biology, 1970. ISSN 0022-2836.

[14] Smith, T.F. a Waterman, M.S. *Identification of Common Molecular Subsequences.* 147(1): pp. 195-7, Journal of Molecular Biology, 1981.

[15] Holm, L. a Sander, C. *Protein Structure Comparison by Alignment of Distance Matrices* [online]. 1993, [cit. 2012-11-17].
http://www.ncbi.nlm.nih.gov/pubmed/8377180.

[16] Gerstein, M. a Levitt, M. *Using Iterative Dynamic Programming to Obtain Accurate Pair-Wise and Multiple Alignments of Protein Structures* [online]. 1996, [cit. 2012-11-17].
http://www.ncbi.nlm.nih.gov/pubmed/8877505.

[17] Gibrat, J.F. a spol. *Surprising Similarities in Structure Comparison* [online]. 1996, [cit. 2012-11-17].
http://www.ncbi.nlm.nih.gov/pubmed/8804824.

[18] Falicov, A. a Cohen, F.E. *A Surface of Minimum Area Metric for the Structural Comparison of Proteins* [online]. 1996, [cit. 2012-11-17].
http://www.ncbi.nlm.nih.gov/pubmed/8637017.

[19] Singh, Amit P. a Brutlag, Douglas L. *Hierarchical Protein Structure Superposition Using Both Secondary Structure and Atomic Representations* [online]. 1997, [cit. 2012-11-17].
http://www.ncbi.nlm.nih.gov/pubmed/9322051.

[20] Ritchie, D.W. a spol. *Fast Protein Structure Alignment using Gaussian Overlap Scoring of Backbone Peptide Fragment Similarity* [online]. 2012, [cit. 2012-11-17].
http://www.ncbi.nlm.nih.gov/pubmed/23093609.

[21] Galgonek, J.; Hoksza, D. a Skopal, T. *SProt: Sphere-Based Protein Structure Similarity Algorithm* [online]. 2011, [cit. 2012-11-17].
http://www.ncbi.nlm.nih.gov/pubmed/22166105.

[22] Szustakowski, J.D. a Weng, Z. *Protein Structure Alignment Using a Genetic Algorithm* [online]. 2000, [cit. 2012-11-20].
http://www.ncbi.nlm.nih.gov/pubmed/10707029.

[23] Szustakowski, J.D. a Weng, Z. *K2: Protein Structure Comparisons and Their Statistical Significancem* [online]. 2002, [cit. 2012-11-20].
http://www.ncbi.nlm.nih.gov/pubmed/10707029.

[24] Samuel, A.L. *Some Studies in Machine Learning Using the Game of Checkers.* 3(3), pp. 210–229, IBM Journal of Research and Development, 1959.

[25] Henikoff, S. a Henikoff, J.G. *Amino Acid Substitution Matrices from Protein Blocks.* PNAS 89(22): pp. 10915–10919, 1992.

[26] Bäck, Thomas. *Evolutionary Algorithms in Theory and Practice.* New York: Oxford University Press, 1996. ISBN 0-19-509971-0.

[27] Whitley, Darrell. *The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best.* Pp. 116-121, Proceedings of the third international conference on Genetic algorithms, 1989.

[28] Birzele, F. a spol. *Vorolign–Fast Structural Alignment Using Voronoi Contacts* [online]. 23(2): e205-11, Bioinformatics. 2007, [cit. 2012-11-30]. http://www.ncbi.nlm.nih.gov/pubmed/17237093.

[29] Chandonia, J.M. a spol. *The ASTRAL Compendium in 2004* [online]. 32(Database issue): D189-92, Nucleic Acid Res. 2004, [cit. 2012-11-30]. http://www.ncbi.nlm.nih.gov/pubmed/14681391.

[30] Hoksza, D. a Galgonek, J. *Alignment-Based Extension to DDPIn Feature Extraction* [online]. International Journal of Computational Bioscience. 2010, [cit. 2012-11-30]. http://siret.ms.mff.cuni.cz/hoksza/papers/ijcb2010.pdf.

[31] Sacan, A.; Toroslu, I.H.; Ferhatosmanoglu, H. *Integrated Search and Alignment of Protein Structures* [online]. 24(24), pp. 2872–2879, Bioinformatics. 2008, [cit. 2012-11-30]. http://www.ncbi.nlm.nih.gov/pubmed/18945684.

[32] Altschul, S.F. a spol. *Gapped BLAST and PSI-BLAST: a New Generation of Protein Database Search Programs* [online]. 25: pp. 3389–3402, Nucleic Acids Res. 1997, [cit. 2012-11-30]. http://www.ncbi.nlm.nih.gov/pubmed/9254694.

[33] Zhang, Y. a Skolnick, J. *Scoring Function for Automated Assessment of Protein Structure Template Quality.* 57(4): pp. 702-710, Proteins: Structure, Function, and Bioinformatics, 2004.

[34] *Qt 4.7: Installation* [online]. c2008-2011, [cit. 2012-12-05]. http://doc.qt.digia.com/4.7-snapshot/installation.html.

Zoznam obrázkov

1.1 Všeobecná štruktúra alfa aminokyselín 4
1.2 Jednotlivé úrovne proteínovej štruktúry 5
1.3 Tok informácií v biologických systémoch 5
1.4 Rast počtu štruktúr zaznemenaných PDB 6

2.1 Stromová reprezentácia operácie *XOR* 12
2.2 Dve rodičovské indivídua a body kríženia 14
2.3 Fragmenty, ktoré sa obmenia 14
2.4 Potomkovia ako výsledok kríženia 15

3.1 Odlišnosť medzi globálnym a lokálnym zarovnaním 17
3.2 Párovanie SSE v KENOBI 21

4.1 Vzťah medzi APASphere a APSSE 27
4.2 Vzťah medzi APTree a APSSE 28
4.3 Ilustrácia objektu ProteinAlignedPair s vyznačenými typmi SSE . 29
4.4 Časť výsledného prevodu zo zarovnania 30
4.5 APTree objekty ako rodičia s bodom kríženia 34
4.6 APTree objekty ako potomkovia 35
4.7 Vznik nových AP a-sfér a objektov APSSE 35

5.1 Porovnanie priebehu RMSD v evolúci zarovnania dvojíc štruktúr 1OYIa a 1ADRa počas 100 000 generácií 39
5.2 Porovnanie priebehu RMSD v evolúci zarovnania dvojíc štruktúr 1OYIa a 1ADRa počas 600 000 generácií 40
5.3 Vznik novej úrovne v stromovej reprezentácii 41

B.1 Operácia *hop* v GA 51
B.2 Rodičovské indivídua pre operáciu *kríženia* v GA 51
B.3 Potomkovia po operácii *kríženia* v GA 52
B.4 Diagram paradigmy genetického programovania 53

Zoznam tabuliek

5.1 Úspešnosti klasifikácie metód na jednotlivých úrovniach 38

Dodatok A

Obsah priloženého CD

Obsah priloženého CD nosiča je organizovaný nasledovne:

- **programatorska_dokumentacia/**
  adresár obsahujúci vygenerované dokumentácie programom Doxygen
  - **format_hmtl/**
    - **index.php**
    - **...**
  - **programatorska_dokumentacia.pdf**
- **projekt_pre_linux_a_testovanie/**
  zdrojové kódy pre linuxový operačný systém
  - **include/**
    - **...**
  - **src/**
    - **pdb_database/**
      niekoľko vybraných súborov reprezentujúcich jednotlivé bielkoviny
      - **...**
    - **configuration.xml**
      konfiguračný súbor
    - **execute_tests.sh**
      hlavný skript pre spustenie viacero inštancií
    - **get_sim.sh**
      skript pre spracovanie
    - **ids_complete_diff.txt**
      súbor s identifikátormi neklasifikovaných proteínových štruktúr
    - **ids_complete_orig.txt**
      súbor s identifikátormi klasifikovaných proteínových štruktúr
    - **...**
- **projekt_pre_visual_studio/**
  spustiteľný projekt pre program Visual Studio (optimalizovaný pre verziu 2012)

  - **...**

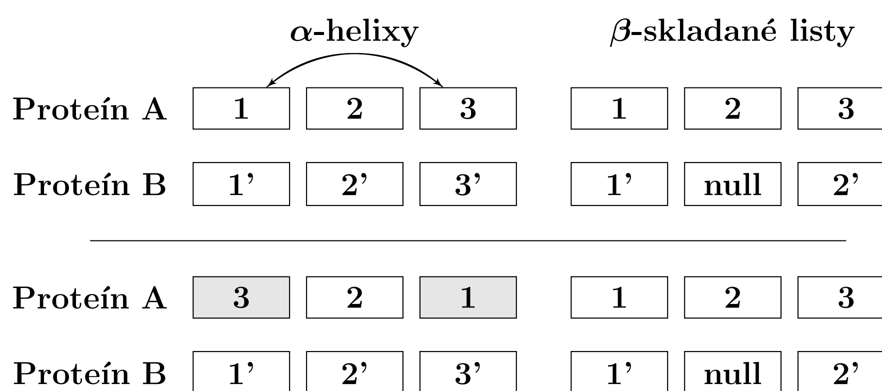
- **spustitelny_program_pre_windows/**
  adresár so spustiteľnou verziou programu

  - **ProSSiGen.exe**
    hlavný spustiteľný program
  - **...**

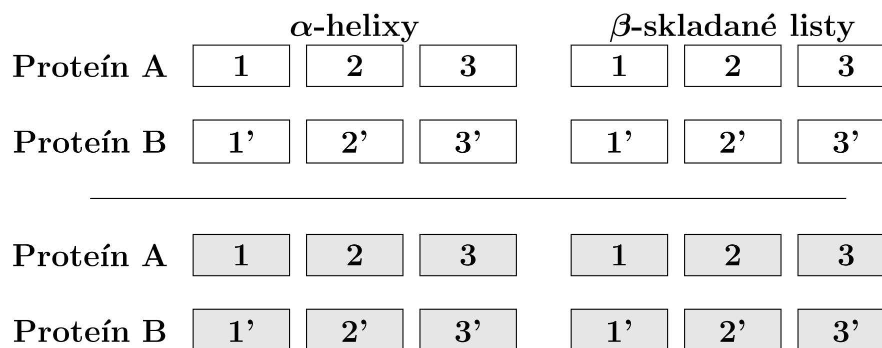
- **pdb_database.rar**
  archív s kompletnou sadou proteínových štruktúr pre testovacie účely
- **bakalarska_praca_miroslav_siagi.pdf**
  text tejto bakalárskej práce
- **ukazka_dat_vysledkov_testovania.rar**
  archív obsahujúci adresár s podadresármi (s celkovým počtom súborov približne 100 000), ktoré sú pomenované podľa ID neklasifikovanej štruktúry so zaznamenanými hodnotami RMSD
- **uzivatelska_dokumentacia.pdf**
  užívateľská dokumentácia

Dodatok B

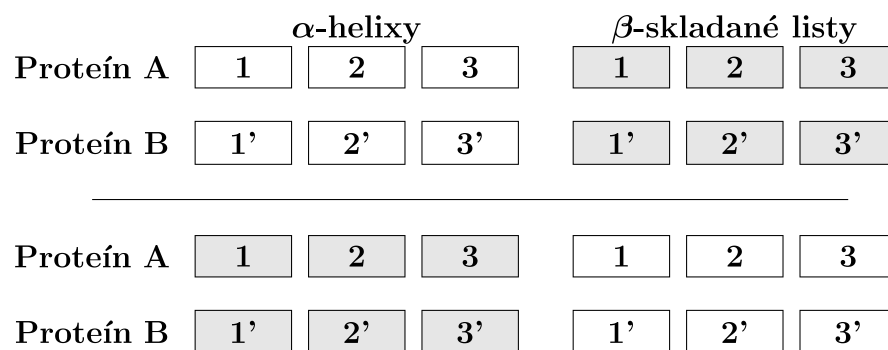
Ilustračné obrázky



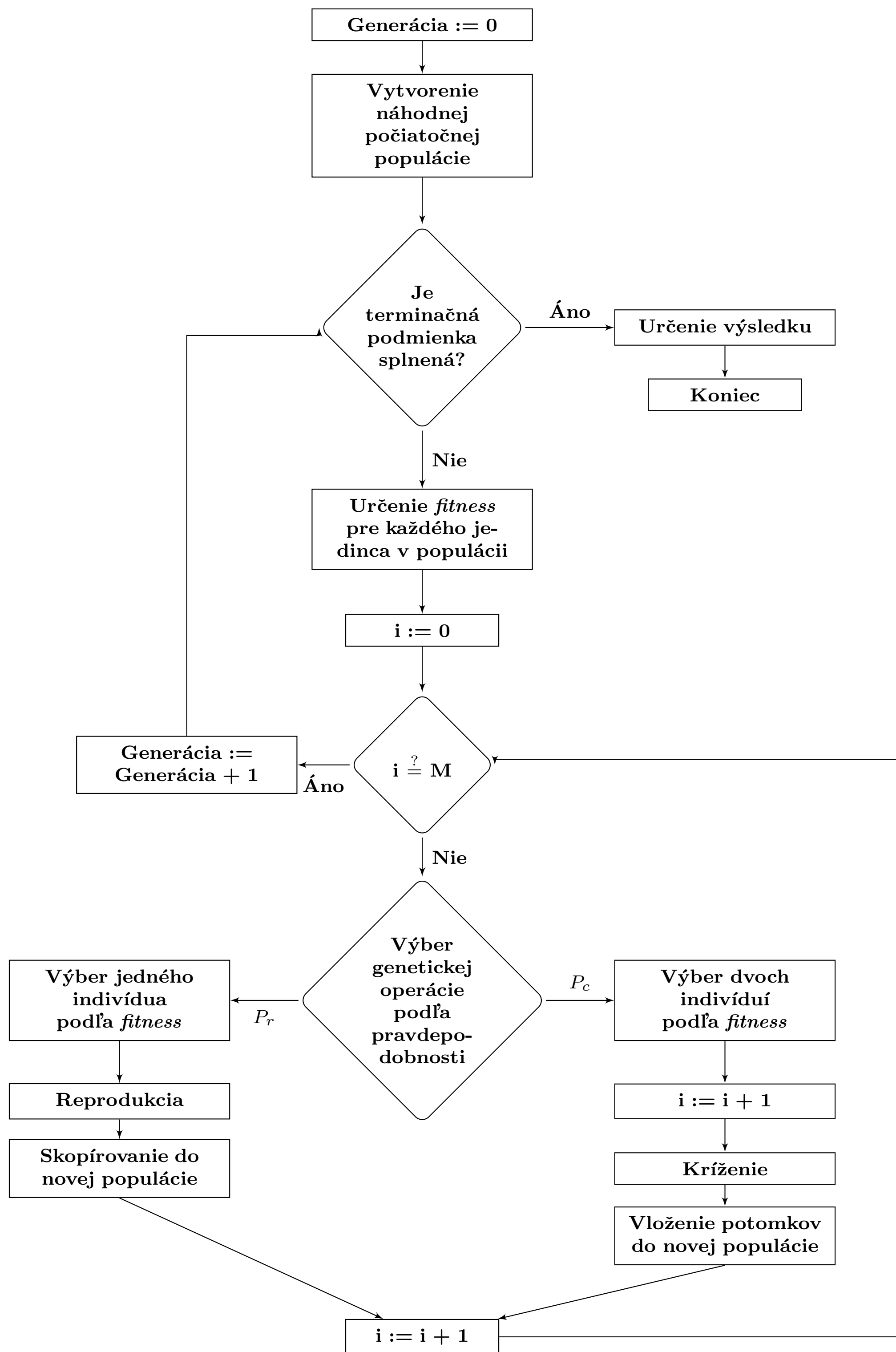
Obr. B.1: Operácia *hop* v GA



Obr. B.2: Rodičovské indivídua pre operáciu *kríženia* v GA



Obr. B.3: Potomkovia po operácii *kríženia* v GA



Obr. B.4: Diagram paradigmy genetického programovania

Dodatok C

Užívateľská dokumentácia

C.1 Požiadavky a spustenie programu

Program je prispôsobený na spúšťanie a testovanie na „Linux-based“ operačnom systéme. Podmienkou je mať nainštalovaný Qt framework (odporúčaná verzia je 4.7.4 alebo vyššia) a kompilátor pre C++.

Bližšie informácie inštulácii Qt framework sa nachádzajú na oficiálnych internetových stránkach [34]. Na linuxových operačných systémoch sa dá Qt framework nainštalovať jednoducho pomocou príkazu *apt-get install*.

Minimálna konfigurácia hardware predstavuje aspoň 4GB RAM, 1GHz CPU.

K dispozícii je aj projekt pre Visual Studio 2012, avšak pre testovanie vo veľkom nie je prispôsobený.

Skripty na účel testovania, spolu so zdrojovým kódom pre Linux i Windows (projekt pre Visual Studio 2012) sú dostupné v repozitári na nasledovnej URL adrese: `http://code.google.com/p/prossigen/`[1].

Na priloženom CD v adresári *spustitelny_program_pre_windows* môžeme nájsť skompilovanú a spustiteľnú verziu programu, ktorý sa dá spustiť cez príkazový riadok. Tento spustiteľný *ProSSiGen.exe* súbor bol testovaný na OS Windows 8 64bit s nainštalovaným Qt framework, nainštalovaným Visual Studiom o verzii 2010 a 2012.

V prípade, že sa nám nepodarí spustiteľný súbor spustiť, je možno potrebné nainštalovať Qt framework a skompilovať projekt pre Visual Studio, ktorý nájdeme v adresári *projekt_pre_visual_studio*. V prípade neúspešnej kompilácie, je potrebné skontrolovať príslušné hlavičkové súbory a cesty ku Qt knižniciam a to nasledovne pre *Debug* mód pre jednotlivé položky takto:
Project – Properties – C/C++ – General – Additional Include Directories:
<include adresár projektu ProSSiGen>;
<cesta, kde je nainštalované Qt> \include;
<cesta, kde je nainštalované Qt> \include\QtCore

Project – Properties – Linker – General – Additional Library Directories:
<include adresár projektu ProSSiGen>;
<cesta, kde je nainštalované Qt> \lib

[1] Pre priame stiahnutie SVN repozitára treba použiť príkaz „svn checkout http://prossigen.googlecode.com/svn/trunk2012/ prossigen-read-only“

Project – Properties – Linker – Input – Additional Dependencies:
qtmain.lib;QtCore4.lib;<zvyšné knižnice pôvodne zahrnuté>

V prípade *Release* módu sa nastavenia projektu mierne líšia:
Project – Properties – VC++ Directories – Include Directories:
<cesta, kde je nainštalované Qt> \include;
<cesta, kde je nainštalované Qt> \include\QtCore;
<zvyšné položky pôvodne zahrnuté>

Project – Properties – C/C++ – General – Additional Include Directories:
<include adresár projektu ProSSiGen>;
<cesta, kde je nainštalované Qt> \include;
<cesta, kde je nainštalované Qt> \include\QtCore

Project – Properties – Linker – General – Additional Library Directories:
<cesta, kde je nainštalované Qt> \lib;
<zvyšné položky pôvodne zahrnuté>

Project – Properties – Linker – Input – Additional Dependencies:
qtmain.lib;QtCore4.lib;<zvyšné knižnice pôvodne zahrnuté>

Projekt sa nechá následne skompilovať a výsledný spustiteľný súbor sa následne spúšťa z príkazového riadku vo formáte:
> ProSSiGen.exe <ID prvej štruktúry> <ID druhej štruktúry> <cesta k adresáru uložených štruktúr>[2]

Pre spustenie na Linuxe sú potrebné zdrojové kódy preň určené, ktoré skompilujeme v adresári projektu *src* nasledovnými príkazmi:
$ *qmake*
$ *make clean*
$ *make*

Vygeneruje sa nám spustiteľný súbor *prossigen*, ktorý spustíme v termináli podobne ako cez príkazový riadok. Uveďme si príklad:
./prossigen "1nh8a" "1h3d_" pdb_database/

Pri spustiteľnom súbore sa musí taktiež nachádzať konfiguračný súbor *configuration.xml* so zobrazeným ukážkovým obsahom nižšie.

```
1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2  <prossigen_settings>
3    <asphere>
4      <radius>9</radius>
5    </asphere>
6    <needleman_wunsh_settings>
7      <!-- if another matrix is used, the default scoring
8           will be applied, i.e. 1 for match, 0 for penalty -->
9      <scoring_matrix>BLOSUM62</scoring_matrix>
10     <gap_penalty>-4</gap_penalty>
```

[2] Dôležité je, aby súbory popisujúce jednotlivé proteíny mali koncovku súboru *.ent*, viď priložený archív s databázou proteínových súborov určené pre testovanie.

```
11    </needleman_wunsh_settings>
12    <gp>
13      <termination_criterion_setting>
14        <individual_score>1</individual_score>
15        <number_of_generations>2000</number_of_generations>
16      </termination_criterion_setting>
17      <number_of_threads>1</number_of_threads>
18      <prob_of_crossover>0.9</prob_of_crossover>
19      <prob_of_reproduction>0.1</prob_of_reproduction>
20      <!-- The value must be in range of (1;2] -->
21      <selective_pressure_parameter>1.8</selective_pressure_parameter>
22    </gp>
23  </prossigen_settings>
```

C.2 Testovacia časť

Testovanie pre veľké množstvo štruktúr je prispôsobené pre linuxový operačný systém. Spolu s priloženým zdrojovým kódom na CD sa nachádzajú aj potrebné skripty pre tento účel a sú to:
execute_tests.sh, get_next_diff_id_generic.sh, run_test_generic.sh, time_checker_generic.sh

Ako hlavný skript pre spustenie je prvý vymenovaný a spúšťa sa so štyrmi parametrami:
./execute_tests.sh <súbor1> <súbor2> <cesta_k_priečinku> <počet_inštancií>

Súbor1 je súbor, kde na každom riadku sa nachádza ID neklasifikovanej bielkoviny s ID reťazca, *súbor2* je súbor s identifikátormi štruktúr klasifikovaných bielkovín, *cesta_k_priečinku* je cesta k priečinku, kde sa nachádzajú súbory s koncovkou *.ent* popisujúce jednotlivé bielkoviny a *počet_inštancií* je prirodzené číslo predstavujúce počet inštancií programu, ktoré sa spustia paralelne. Pri použití jedného vlákna pre inštanciu a počet inštancií sa odporúča počet troj násobku dostupných procesorových vlákien.

Na CD sa tiež nachádzajú priložené súbory s identifikátormi štruktúr s názvom *ids_complete_diff_unique.txt* a *ids_complete_orig_unique.txt*.

Pre testovanie sa odporúča použiť menšia sada neklasifikovaných štruktúr a testovať ich po častiach. Skript totiž zo súboru *súbor1* postupne vyberá jednotlivé ID štruktúr a akonáhle jednotlivé inštancie zistia, že je súbor prázdny, inštancia sa ukončí. Platí, že jedna inštancia obstaráva vždy jednu neklasifikovanú štruktúru so všetkými klasifikovanými.

Chod inštancií, ktoré v skutočnosti prebiehajú v pozadí, môžeme sledovať v termináli a údaje o ohodnotení jednotlivých dvojíc štruktúr sa ukladajú do priečinka *data/*. V tomto priečinku sú ďalšie adresáre pomenované podľa neklasifikovaných štruktúr a v nich sa nachádzajú súbory obsahujúce poradové číslo generácie evolučného procesu a *fitness* (RMSD) získané pri porovnávaní s jednotlivými klasifikovanými štruktúrami. V priečinku *data/* ešte môžeme vidieť súbor *main_results.txt*, do ktorého sa zaznamenávajú najmenšie hodnoty RMSD jednotlivých porovnávajúcich dvojíc štruktúr, ktorých evolučný proces dobehol úspešne do konca.

V súbore *logs/log.txt* sa nachádzajú všetky výpisy o procesoch jednotlivých porovnávaní ako načítaní jednotlivých súborov, dokončenia priebehu sekvenčného zarovnania, dokončenia prevodu na objekty AP-Tree a zakončenia algoritmov genetického programovania. Rovnako sa tu môžu nachádzať informácie o výnimkách a chybách.

Implementované sú dva základné typy výnimiek. Prvou z nich je výnimka týkajúca sa stavu, kedy program nedokáže nájsť súbor s danou štruktúrou. Pripomíname, že všetky súbory (s koncovkou *.ent*) sa majú nachádzať v spoločnom priečinku na jednej úrovni. Výnimka nesúca informácie o tejto chyba sa volá *CannotReadFileException* a pod týmto názvom sa v chybovom výpise môže vyskytovať.

Druhý typ výnimky sa nazýva *PdbFileParseException*, ktorá sa vyskytne vtedy, ak PDB súbor (v našom prípade s koncovkou *.ent*) obsahuje pre nás dôležité informácie, ktoré sú ale popísané v nesprávnom formáte. Zoznam možných chýb môžeme nájsť v implementácii v triede *PdbFileParseException* v štruktúre *struct FieldMap*.

Po ukončení všetkých inštancií následne spustíme skript *get_sim.sh*, ktorý prejde priečinok *data/* so zaznamenanými výsledkami a do súboru *similarities.txt* na každý riadok vloží dvojicu štruktúr najviac podobných s príslušnou hodnotou RMSD. Tieto dvojice môžeme následne porovnať s manuálnou klasifikáciou SCOP[3].

[3]Klasifikáciu určitej bielkoviny môžeme nájsť na internetovej stránke databázy SCOP, konkrétne napr na URL: `http://scop.mrc-lmb.cam.ac.uk/scop/data/scop.b.html`.