VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

VÝPOČET ATRIBUTŮ PRO PŘEDPOVĚĎ DŮSLEDKU MUTACE NA FUNKCI PROTEINU

ATTRIBUTES CALCULATION FOR PREDICTION OF MUTATION EFFECT ON PROTEIN FUNCTION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE JAN ŠINKORA
AUTHOR

VEDOUCÍ PRÁCE Ing. PETR JAŠA
SUPERVISOR

BRNO 2010

Abstrakt

Tato práce se zabývá problematikou z oblasti bioinformatiky, algoritmů a datových typů a strojového učení. Základem práce jsou již existující aplikace Caver a Deleterious, na jejichž vývoji se podíleli studenti Fakulty informatiky Masarykovy univerzity a Fakulty informačních technologií Vysokého učení technického v Brně. Aplikace Deleterious slouží k získávání a výpočtu atributů proteinů důležitých k predikci vlivu mutace proteinu na jeho výslednou funkci a Caver je program pro hledání tunelů v prostorovém modelu proteinu. Výsledkem má být rozšíření těchto aplikací o výpočet nových atributů, které mohou přispět ke zlepšení přesnosti predikce. Přidané atributy souvisí s hledáním a měřením kapes proteinu.

Abstract

This thesis deals with issues of bioinformatics, machine learning, algorithms and data structures. The thesis is based on existing applications, Caver and Deleterious, developed by students from the Faculty of Informatics, Masaryk University and the Faculty of Information Technology, Brno University of Technology. The Deleterious framework calculates protein attributes that are important for the prediction of the effect of protein mutations on its function. Caver is a tool that finds tunnels in the 3-dimensional model of a protein. The goal of the thesis is to extend these applications by adding more attributes to the prediction process that could lead to improved prediction. The added attributes are related to detection and measurement of protein pockets.

Klíčová slova

bioinformatika, strojové učení, DNA, protein, mutace, kapsa, predikce, výpočetní geometrie, Delaunayho triangulace, diskrétní tok

Keywords

bioinformatics, machine learning, DNA, protein, mutation, pocket, prediction, computational geometry, Delaunay triangulation, discrete flow

Citace

Jan Šinkora: Výpočet atributů pro předpověď důsledku mutace na funkci proteinu, bakalářská práce, Brno, FIT VUT v Brně, 2010

Výpočet atributů pro předpověď důsledku mutace na funkci proteinu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Petra Jaši. Uvedl jsem všechny prameny a publikace, ze kterých jsem čerpal.

. . . . . .
Jan Šinkora
14. května 2010

Poděkování

Tímto děkuji vedoucímu práce Ing. Petru Jašovi, Mgr. Petru Benešovi a Mgr. Antonínu Pavelkovi z Loschmidtových Laboratoří za výbornou průpravu do problematiky bioinformatiky a jejich odbornou pomoc při řešení množství problémů, které při tvorbě této práce nastaly.

© Jan Šinkora, 2010.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1 Úvod **3**

2 Biologický základ **4**
2.1 DNA . . . 4
2.1.1 Replikace DNA . . . 4
2.2 RNA . . . 4
2.3 Proteosyntéza . . . 6
2.3.1 Transkripce RNA . . . 6
2.3.2 Translace . . . 7
2.4 Proteiny . . . 7
2.4.1 Struktura proteinů . . . 7
2.4.2 Funkce proteinů . . . 8
2.4.3 Mutace DNA a proteinů . . . 9

3 Odhad vlivu mutace na funkci proteinu **11**
3.1 Bioinformatika . . . 12
3.2 Strojové učení . . . 12
3.2.1 Dělení metod strojového učení . . . 12
3.3 Nástroje pro výpočet atributů proteinů a předpověď vlivu mutace na funkci proteinu . . . 14
3.3.1 Nástroje pro výpočet atributů proteinů . . . 14
3.3.2 Nástroje pro odhad vlivu mutace na základě vypočtených atributů . 15
3.4 Další vývoj v této oblasti . . . 15

4 Analýza **16**
4.1 Vybrané atributy . . . 16
4.2 Hledání kapes v proteinu . . . 16
4.2.1 Delaunayho triangulace . . . 17
4.2.2 Diskrétní tok . . . 17
4.2.3 Měření kapes . . . 19
4.2.4 Caver . . . 19
4.3 Výpočet atributů ze získaných hodnot . . . 20
4.3.1 Deleterious – framework na výpočet atributů . . . 20
4.3.2 WEKA . . . 23

5 Implementace **25**
5.1 Caver . . . 25
5.1.1 API Caveru . . . 25
5.1.2 Implementace algoritmu hledání kapes . . . 26
5.2 Framework Deleterious . . . 28
5.2.1 Přidání obsluhy Caveru do projektu Deleterious . . . 30

6 Testování a výsledky **32**
6.1 Použitý dataset . . . 32
6.2 WEKA - Křížová validace . . . 33
6.2.1 WEKA Experimenter . . . 33
6.2.2 WEKA KnowledgeFlow . . . 34

7 Závěr **36**

A Konfigurace a spuštění **39**
A.1 Konfigurace . . . 39
A.2 Spuštění . . . 39
A.2.1 Caver . . . 39
A.2.2 Deleterious . . . 39
A.3 Dokumentace . . . 39
A.4 Upozornění . . . 40

B Obsah CD **41**

Kapitola 1

Úvod

Pokrok v oblasti výpočetní techniky umožnil vznik nových vědních oborů, které kombinují informatiku s přírodními vědami jako jsou fyzika, biologie a chemie. Jedním z těchto oborů je bioinformatika, která se zabývá shromažďováním, analýzou a zobrazováním biologických dat, primárně na molekulární úrovni. Databáze s biologickými údaji se vyznačují svou rozsáhlostí a bez použití výpočetní techniky by bylo časově neúnosné provést jejich podrobnou analýzu.

Bioinformatika se mimo jiné dotýká *proteinového inženýrství*. To se zabývá návrhem a výrobou nových proteinů s pozměněnou funkcí skrze úpravu sekvence aminokyselin, které protein tvoří. Motivací pro takovou činnost může být například průmyslové využití proteinů, likvidace znečištění životního prostředí nebo syntéza léků. Je časově a finančně neúnosné experimentálně zkoušet všechny možné kombinace mutací. Zatím však nebyl nalezen způsob, jak přesně předpovědět změnu chování proteinu po mutaci. Jedním ze směrů, kterým se bioinformatika v oblasti proteinového inženýrství ubírá, je proto odhad vlivu mutace pomocí metod strojového učení na základě laboratorně zjištěných znalostí. Výsledky odhadů jsou ale zatím stále nepřesné.

Tato práce má za úkol přispět k vývoji a zvýšení správnosti zmíněného odhadu. Ve spolupráci s Loschmidtovými Laboratořemi (viz `http://loschmidt.chemi.muni.cz`) byly v rámci práce do již existujících aplikací přidány algoritmy na vyhledání a měření kapes v bílkovinách a následné určení, zda mutovaná část proteinu v nějaké kapse leží. Kapsy jsou důležité pro vazbu molekul, se kterými proteiny interagují, proto je jejich výpočet důležitý pro predikci chování proteinu. V aplikacích zatím není hledání kapes přímo implementováno a k tomuto účelu slouží externí nástroje, které jsou však často nespolehlivé a nelze je podle potřeby upravovat.

Dokument je rozčleněn na několik kapitol. Po tomto úvodu se druhá kapitola zaměří na biologický základ, který popisuje termíny z biologie použité v této práci. Zahrnuje výklad o DNA, RNA a bílkovinách. Ve třetí kapitole je úvod do bioinformatiky související s proteinovým inženýrstvím, jsou zde uvedeny metody strojového učení, které jsou v bioinformatice široce používány, a nakonec jsou shrnuty různé přístupy a aplikace, které se věnují odhadu vlivu mutace proteinu na jeho funkci. Tvůrčí část práce se dělí na teoretickou analýzu a praktickou aplikaci. Ve čtvrté kapitole se práce zabývá metodami na hledání kapes v proteinu, analyzuje aplikace, které budou touto prací rozšířeny a používány, a uvádí formáty dat, se kterými se bude pracovat. Kapitola pátá pak vysvětluje některé základní principy implementace nových algoritmů. V šesté kapitole jsou uvedeny a popsány výsledky a sedmá kapitola shrnuje poznatky získané z tvorby a výsledků práce a nastiňuje možnosti dalšího pokračování vývoje.

Kapitola 2

Biologický základ

Tato kapitola se zabývá uvedením do problematiky proteinů. Popisuje základní termíny potřebné pro pochopení všech aspektů této práce. Po obecném seznámení s funkcí molekuly DNA a principy dědičnosti je zde vysvětlen proces vzniku proteinu, jeho struktura a význam v organismu.

2.1 DNA

Kyselina deoxyribonukleová (zkráceně DNA z anglického Deoxyribonucleic acid) je nositelka genetické informace. Na DNA se dá pohlížet jako na plán celého konkrétního organismu a je tedy kritická pro rozvoj a funkčnost téměř všech žijících organismů. Informace DNA kóduje pomocí sekvence čtyř nukleových bází: adeninu, guaninu, cytosinu a thyminu. Každá báze má svůj protějšek: adenin se váže na thymin a cytosin na guanin. Z těchto bází jsou spolu s pětiuhlíkatým monosacharidem a jedním nebo více zbytky fosforečné kyseliny sestaveny nukleotidy, které se řetězí do dvou vláken. Tato struktura je zobrazena na ilustraci 2.1.

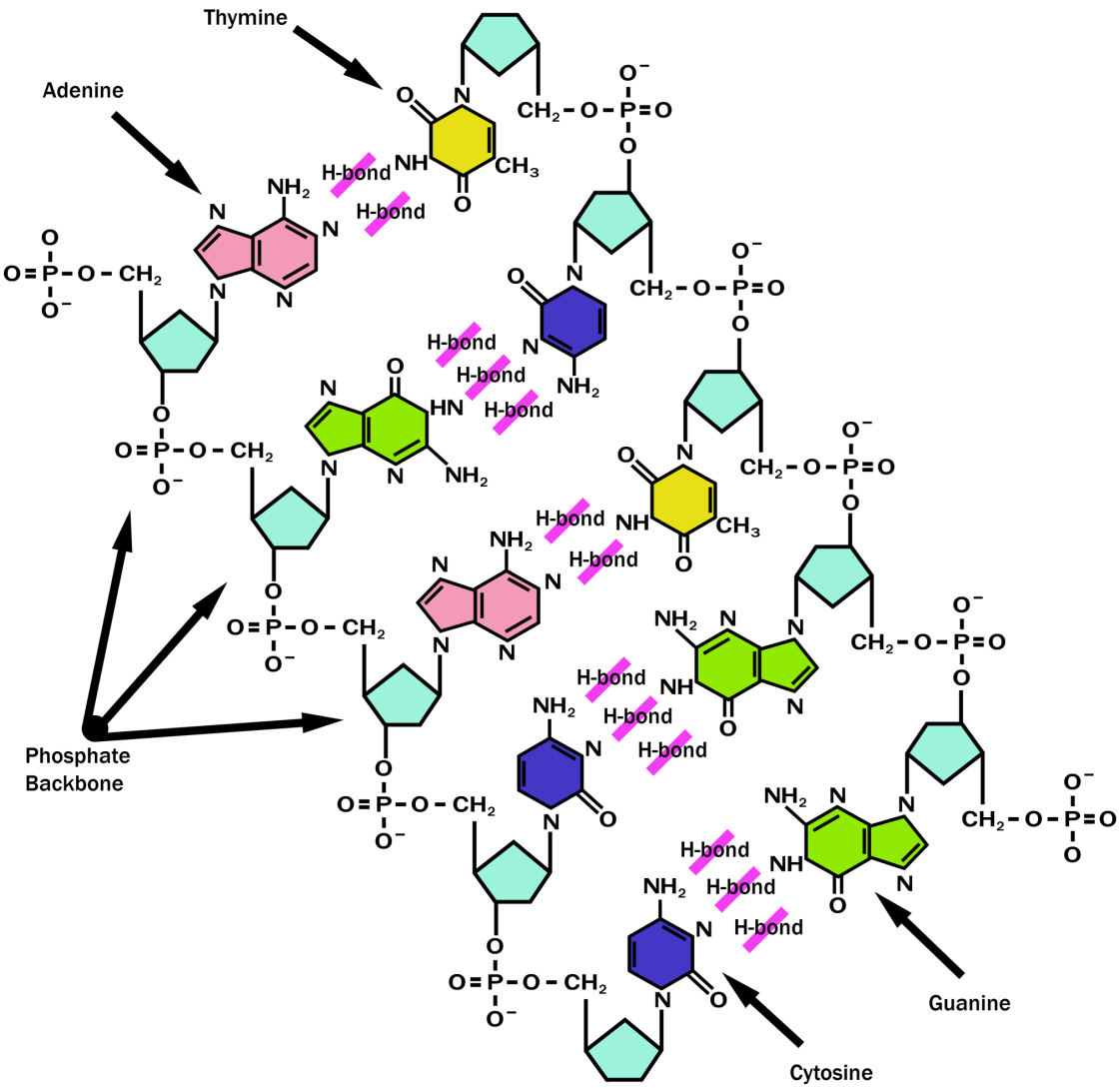
Vlákna DNA jsou komplementární, čili nukleotidy jednoho vlákna jsou řadou komplementárních nukleotidů k těm z vlákna druhého. Nukleotidy jsou spojeny vodíkovými můstky. Dvě vlákna v DNA se stáčejí do charakteristického tvaru pravotočivé dvojšroubovice (odtud pojem "dvojšroubovice DNA") [24]. DNA se u eukaryot, kam spadají rostliny a živočichové, vyskytuje v jádrech buněk, kde probíhá i její replikace.

2.1.1 Replikace DNA

Při dělení buňky se DNA replikuje, čímž dochází k přesnému přenosu genetické informace. Obě buňky vzniklé dělením nesou tedy DNA se stejnou informací jako původní buňka. Díky komplementaritě obou vláken DNA je možné použít obě vlákna na vytvoření přesné kopie původní dvojšroubovice [3]. Proces replikace provádí soubor proteinů zvaný replikační aparát, který DNA na několika místech zároveň začne rozplétat oběma směry a k přečteným nukleotidům přidává jejich komplementární protějšky. Replikace je zobrazena na ilustraci 2.2.

2.2 RNA

Ribonukleová kyselina (RNA) je, podobně jako DNA, tvořena nukleotidy. Na rozdíl od DNA je většinou pouze jednovláknová a místo thyminu používá jinou bázi: uracil [2]. Některé



Obrázek 2.1: Chemická struktura DNA [26]

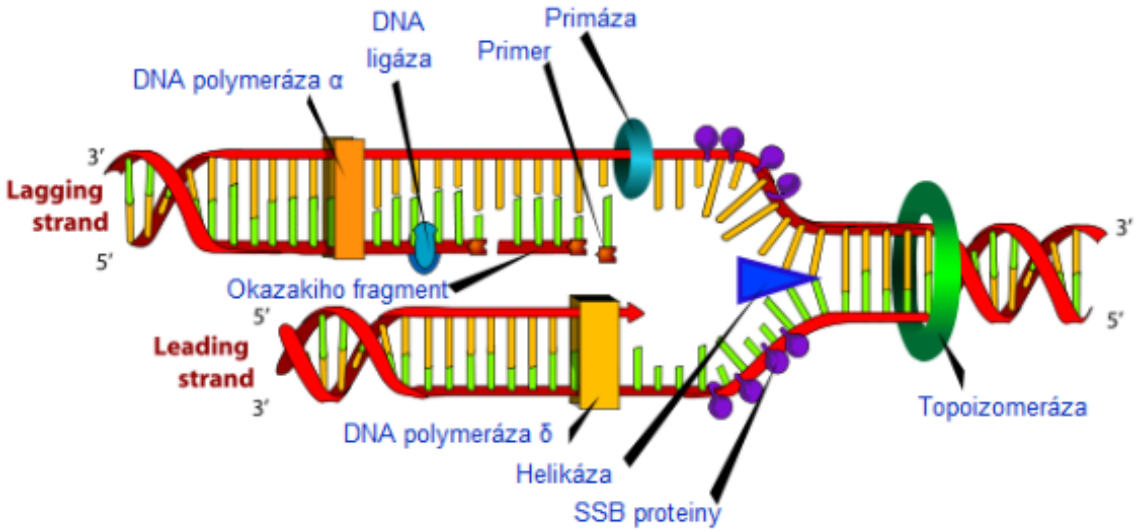
úseky RNA jsou komplementární s jinými, což umožňuje jejich spojení a následné sbalení RNA do trojrozměrné struktury. Známe několik základních typů RNA:

1. Mediátorová RNA (mRNA)
   Tato makromolekula je přepsána přímo z DNA a nese informaci o složení proteinu. Je použita v procesu proteosyntézy, neboli vytváření nového proteinu.

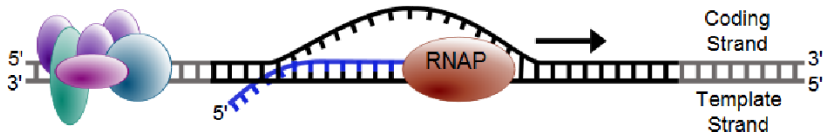
2. Transportní RNA (tRNA)
   Také nazývaná transferová. Slouží k přesunu aminokyselin k ribozomu a napojuje ji do tvořeného proteinu.

3. Ribozomální RNA (rRNA)
   Tvoří spolu se specifickými proteiny ribozomy a podílí se tak na proteosyntéze. Je nejčetnějším typem RNA v buňce.

4. miRNA a siRNA



Obrázek 2.2: Replikace DNA [26]



Obrázek 2.3: Transkripce RNA [26]

Regulují výrobu některých proteinů.

5. snRNA
   Upravuje strukturu mRNA před jejím dokončením.

Zapojení některých typů RNA do procesu tvorby proteinu – proteosyntézy – je popsán v další kapitole.

2.3 Proteosyntéza

Proteosyntéza je proces vytvoření nového proteinu. Sestává ze dvou částí: transkripce (syntéza RNA) a následné translace (sestavení nového proteinu).

2.3.1 Transkripce RNA

RNA je syntetizována v buněčném jádře enzymem zvaným RNA-polymeráza podle vzoru v DNA, nebo vzácně v jiné RNA. Tento proces se nazývá transkripce [3, strana 212 - 218]. RNA-polymeráza najde promotor, což je sekvence nukleotidů určující začátek transkripce. Dvojšroubovice DNA je rozštěpena a podle posloupnosti nukleotidů je vytvářeno vlákno RNA. Po přečtení každého úseku se dvojšroubovice za RNA-polymerázou zase zaplétá. Po dosažení tzv. terminátoru je transkripce ukončena. Ilustrace 2.3 ukazuje transkripci RNA.

Jelikož je v buňce většinou jen jedna kopie genu, který popisuje vznik určitého proteinu, je transkripce velmi důležitá: kdyby byla místo mRNA použita jako vzor přímo DNA, byla

by výroba potřebného množství proteinu mnohem zdlouhavější. Naproti tomu jeden úsek DNA může být vícekrát přepsán v potřebnou RNA a zároveň může být jedna kopie RNA před svojí degradací použita opakovaně k výrobě proteinu [3, strana 121].

2.3.2 Translace

Po transkripci mRNA je ihned možné ji použít k vytvoření nového proteinu. Tato druhá část proteosyntézy se nazývá translace a už neprobíhá v jádře buňky, ale v ribozomech. Sekvence nukleotidů v mRNA je čtena po trojicích, zvaných kodony. Každý kodon kóduje jednu aminokyselinu, která je základním stavebním prvkem proteinů. Množina aminokyselin, kterých je 21, je pak surjektivním zobrazením množiny všech kombinací bází v kodonu (64 možností). Kodony na mRNA samy o sobě nejsou schopny interpretovat aminokyselinu, která k nim náleží. To mají na starost molekuly tRNA. Ty mají na svém těle připojenou aminokyselinu a na jiném místě, zvaném antikodon, je posloupnost tří nukleotidů komplementární ke kodonu, který připojenou aminokyselinu kóduje. Redundance genetického kódu je řešená dvěma způsoby: pro konkrétní aminokyselinu buď existuje mnoho variant tRNA, které pokrývají různé kódující kodony, nebo jedna tRNA doléhá prvními dvěma nukleotidy a třetí, který už není pro rozpoznání aminokyseliny tolik podstatný, je svázán jiným, méně stabilním, typem vodíkového můstku [3, strana 227]. Translace začíná napojením ribozomu na mRNA a následným rozpoznáním místa, odkud má být čtena posloupnost kodonů. Toto místo je určeno speciální trojicí nukleotidů. Poté se opakují tři kroky:

1. tRNA s připojenou aminokyselinou se vazbou kodon-antikodon naváže na aktuální překládané místo v mRNA.
2. Vytvoří se peptidická vazba, která spojí poslední aminokyselinu v už přeložené části proteinu s aminokyselinou na tRNA, čímž dochází zároveň k odpojení předchozí aminokyseliny od její tRNA.
3. tRNA nesoucí předchozí aminokyselinu je odpoutána od mRNA.

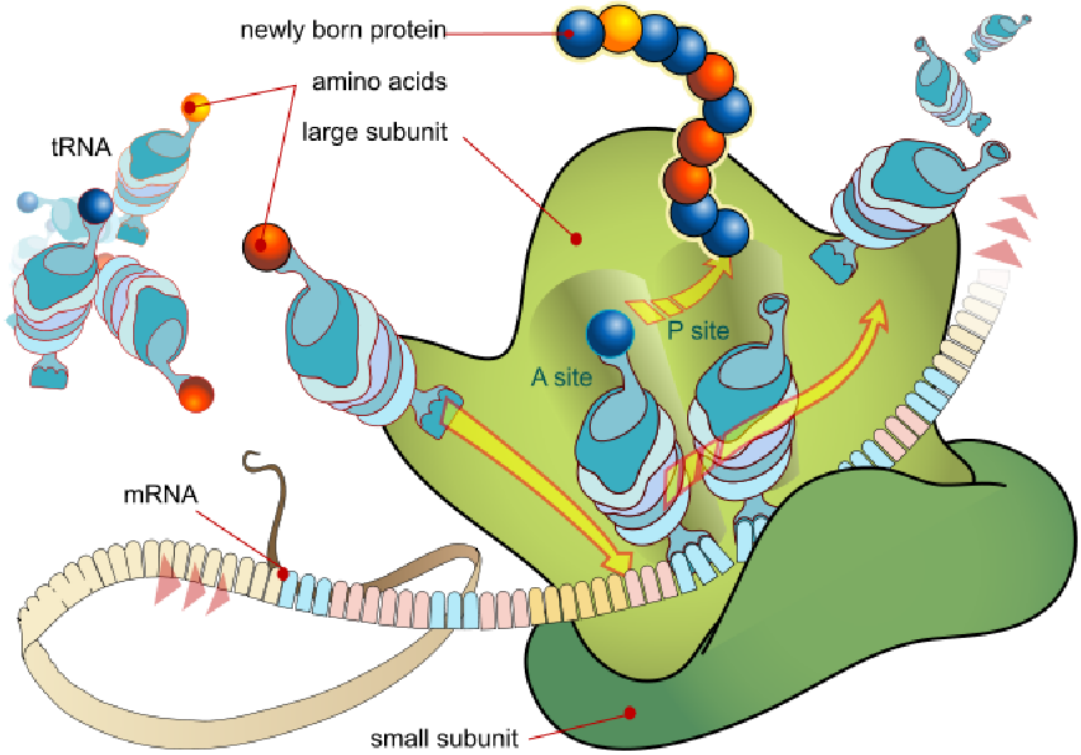
Ribozom se v průběhu každého cyklu posune o tři nukleotidy dále po mRNA. Proces čtení končí, když je rozpoznán jeden ze tří terminačních kodonů. Na ty se neváže žádná tRNA – protein je v takové chvíli odpojen od ribozomu. Translaci zobrazuje ilustrace 2.4.

2.4 Proteiny

Proteiny mohou být také česky nazývané bílkoviny. Některé jsou stavebními kameny buněk, zatímco jiné obstarávají většinu buněčných funkcí [3, strana 133]. Proteiny mohou být enzymy, katalyzující chemické reakce (pepsin), zásobárny na různé látky (ferritin), nebo mohou přenášet molekuly po organismu (hemoglobin) či hrát jednu z mnoha dalších rolí. Jsou tedy velice důležité pro správné fungování prakticky každé buňky.

2.4.1 Struktura proteinů

Proteiny jsou složené z dlouhého řetězce aminokyselin. Tento řetězec se v přímé závislosti na pořadí jednotlivých aminokyselin skládá v prostoru do útvaru s co nejnižší energií. Rozlišujeme čtyři úrovně struktury proteinů, pro které platí, že každá úroveň přímo vychází z předchozí:



Obrázek 2.4: Translace [26]

1. Primární struktura
   sekvence aminokyselin tvořících protein
2. Sekundární struktura
   úseky řetězce aminokyselin složené do útvarů, z nichž jsou hlavní tyto dva typy:
   - α-šroubovice
   - β-pruhy
3. Terciární struktura
   uspořádání celého proteinu v prostoru
4. Kvartérní struktura
   způsob, jakým jsou v prostoru uspořádány proteinové podjednotky ve složitějších proteinových komplexech.

O uspořádání proteinu se někdy mluví jako o jeho konformaci.

2.4.2 Funkce proteinů

Funkce každého proteinu je přímo určena způsobem, jakým je protein schopen navázat spojení s jinými molekulami. To přímo závisí na konformaci proteinu. Proteiny mají povrch s mnoha různými nerovnostmi. Některé z nich jsou vazebnými místy, které svými vlastnostmi určují specifickou látku[1], která se k proteinu v tomto místě může navázat [19]. Vazebná

[1]Připojená molekula je někdy označována jako ligand, u enzymů se mluví o substrátech. V jiných případech se může připojit například další protein.

místa mohou být například:

- Uvnitř proteinu. Cizí molekula se do takového místa dostává například tunely vedoucími skrz protein, které jsou sice mnohokrát užší, než je potřeba, ale konformace proteinu je flexibilní a tak umožňuje protlačení molekuly dovnitř.
- Na povrchu proteinu. Zde může jít například o kapsy, které mají specifický tvar a vlastnosti, komplementární ke tvaru a vlastnostem cizí molekuly. Kapsa je konkrétně dutina (někdy také kavita) v proteinu, která má hrdlo vedoucí na povrch proteinu. Toto hrdlo je užší, než dutina, do které vede.

Cizí molekula je do vazebného místa navázána velkým množstvím nekovalentních[2] vazeb. Takové spojení může mít různé efekty: některé molekuly modifikují funkci proteinu, jiné způsobí aktivaci práce proteinu lehkou změnou konformace proteinu nebo naopak protein deaktivují. Další možností napojovaných molekul jsou třeba látky, které protein odbourává – při rozvazování vazeb si mnohokrát pomáhá i fyzickou silou, kdy používá svoji unikátní konstrukci k ohybu a natahování rozkládané molekuly.

2.4.3 Mutace DNA a proteinů

V průběhu evoluce se jednotlivé formy života od sebe postupně oddělovaly příčinou mutací – z důvodů, popsaných dále, občas dojde u jedince ke změně genetického kódu, což se projeví nějakou změnou v jeho organismu. Občasná změna v DNA je v evoluci vskutku klíčová – organismy, které jsou díky mutacím schopné se přizpůsobit změnám okolí, jsou úspěšné a přežijí. Z pohledu jednotlivce je však náhodná mutace, působící například změnu aminokyseliny v proteinu, většinou škodlivá, ne-li zhoubná.

Příčiny vzniku mutací

Mutace se dají rozdělit na dvě skupiny podle příčiny: jednou jsou mutace vzniklé chybou při replikaci DNA, druhé mají za původce vnější vlivy [23].

1. Chyba při replikaci DNA
   I přes vysokou spolehlivost replikace DNA (jedna chyba na zhruba 10^9 zkopírovaných nukleotidů[3] [3, strana 200]) občas nějaká chyba zůstane neopravená a zafixuje se – při dalších replikacích DNA se v systému zachovává.
2. Vnější vlivy mutací
   Vnějším vlivem rozumíme radiaci nebo některé chemikálie, které mohou způsobit rozpad DNA. Buňka sice má mechanismy pro opravu DNA, ale tento proces rozhodně není vždy úspěšný.

[2] Vazby, při kterých spojené atomy nesdílí elektronové páry. Tyto vazby jsou obecně slabší než vazby kovalentní.

[3] Replikační aparát má ve skutečnosti zhruba o dva řády menší spolehlivost, ale po samotné replikace je DNA ještě kontrolována za účelem odhaleni neshody parity nukleotidů. Tento postup zajišťuje výslednou spolehlivost.

Typy mutace

Na úrovni nukleotidů v DNA rozlišujeme tři typy mutací [23]:

1. Substituce
   Nejdůležitější mutace z pohledu jak evoluce, tak proteinového inženýrství. Jeden nukleotid je nahrazen nukleotidem s jinou nukleovou bází. Tento typ mutace má nejmenší šanci na změnu funkce proteinu – díky redundantnímu genetickému kódu (kodony → aminokyseliny) je možné, že substituční mutace dokonce ani nezpůsobí žádnou změnu cílového proteinu. Možný vliv takové mutace na protein je výměna aminokyseliny na určitém místě. I to může však mít radikální vliv na funkci či stabilitu výsledného proteinu. Takováto mutace se často označuje anglickým termínem non-synonymous nucleotide polymorphism (nsSNP) a je dnes nejsledovanějším typem mutace z hlediska proteinového inženýrství.

2. Vložení
   Méně obvyklá, ale zato nebezpečná mutace, která přidává do sekvence v DNA jeden či více nových nukleotidů. V případě, že jsou vloženy tři nukleotidy, je do proteinu přidána nová aminokyselina. Už to samo o sobě může být pro protein ničivá změna – mnohem větší dopad má však přidání jiného počtu nukleotidů. V takových případech ztrácí celá sekvence za místem mutace svůj původní význam, protože jsou při dekódování čteny úplně jiné trojice.

   Příklad

   Sekvence CTG-GAG dostane nový nukleotid A na čtvrtém místě. Výsledná sekvence bude čtená jako CTG-AGA-G.

3. Odstranění
   Pro tuto mutaci platí to samé co pro předchozí s tím rozdílem, že je nukleotid z řetězce vyňat.

Efekty mutací

Mutace může mít různý vliv na funkci organismu. Jsou místa DNA, kde mutace nevede k žádnému efektu, naopak jsou místa, která svou mutací znemožní správné fungování proteinu například úpravou tvaru vazebného místa. Z pohledu proteinů může mutace v DNA a následná záměna aminokyseliny mít na organismus různé efekty – na některých místech v proteinu mohou dvě podobné aminokyseliny fungovat stejně dobře, v aktivních místech však jakákoli záměna vede většinou k výrazné změně fungování proteinu. Pokud je na místě, kde aminokyselina v proteinu plní čistě strukturní roli, dosazena aminokyselina s jinou velikosti, může toto vést například k uzavření nějaké kapsy, čímž je znemožněno připoutání cizí molekuly a tím pádem protein ztrácí funkci.

Kapitola 3

Odhad vlivu mutace na funkci proteinu

Přesná predikce funkce proteinu je dnes nemožná – neexistuje žádný model, který by byl schopný přesně určit, jak bude protein s libovolnou konfigurací aminokyselin fungovat či jestli bude vůbec stabilní. Dokonce ani nebyl nalezen způsob, jak předpovědět finální konformaci proteinu v prostoru v závislosti na jeho primární struktuře. Máme však k dispozici poměrně velké množství údajů, které pomáhají vliv mutace na funkci proteinu s určitou pravděpodobností odhadnout. Tyto údaje, spojené s konkrétním proteinem, budu v rámci této práce nazývat atributy proteinů. Existují tři základní typy atributů:

Sekvenční atributy
: Sekvenčními atributy se myslí atributy získané z primární struktury proteinu. Mohou to být atributy vycházející přímo ze sekvence aminokyselin jednoho proteinu, nebo z porovnání sekvence studovaného proteinu se sekvencemi jiných, známých proteinů. Existují rozsáhlé databáze, které obsahují experimentálně získané informace o proteinových mutacích a jejich efektu. Některé jsou i veřejně přístupné, viz strana 20. Při shodě delší sekvence aminokyselin sledovaného proteinu s proteiny z těchto databází lze tvrdit, že protein se při mutaci v této části své konfigurace zachová podobně nebo stejně jako referenční proteiny. Konkrétními atributy mohou být například evoluční konzervovanost[1] nukleotidu na určité pozici, náboj a hydrofóbnost aminokyseliny atd. Tyto vlastnosti mohou mít významný vliv na okolí aminokyseliny.

Anotace v databázích
: Některé databáze poskytují již změřené nebo spočítané atributy k proteinům a jejich variantám. Jako příklad lze uvést například označení aktivních nebo vazebných míst proteinu.

Strukturní atributy
: Stejně jako v případě sekvencí jsou k dispozici i údaje o 3D konformaci proteinů a jejich variant. Z toho přímo plyne možnost využití takovýchto údajů pro získání strukturních atributů. Sem spadají atributy jako povrch proteinu přístupný rozpouštědlu (angl. accessible solvent area), nebo vlastnosti a přístupnost aktivních míst.

[1] Aminokyselina na určité pozici je *evolučně konzervovaná*, pokud zde mutace v přírodě nevyskytuje. To totiž ukazuje na to, že jedinec s takovouto mutací pravděpodobně nepřežil.

3.1 Bioinformatika

Bioinformatika je rapidně se rozvíjející obor, který kombinuje výpočetní technologie s biologií [13]. Dnes se bioinformatika využívá hlavně v oblasti molekulární biologie. Základem bioinformatiky jsou velká množství experimentálně získaných dat, u jejichž získávání se klade důraz na co nejpřesnější měření. Čím větší množství přesných dat je k dispozici, tím větší možnosti bioinformatika nabízí. Jako příklad lze uvést následující srovnání:

- Z jedné sekvence aminokyselin, která určuje jeden protein, jsme schopni získat relativně málo užitečných údajů pro další výzkum.
- Pokud však známe i strukturu proteinu v prostoru, lze protein zobrazit pro lepší představu výzkumníka, nebo jej lze analyzovat a například takto určit, které aminokyseliny jsou přístupné rozpouštědlu.
- Pokud jsou k dispozici data o více proteinech i s jejich mutacemi a experimentálně zjištěným dopadem těchto mutací na jejich funkci, je možné pomocí strojového učení s určitou pravděpodobností odhadnout, jaký vliv bude mít nová, ještě nevyzkoušená mutace.

Hlavním zaměřením bioinformatiky je vývoj a analýza algoritmů, které jsou použitelné při práci s biologickými daty. Jedná se například o porovnávání totožnosti řetězců, optimalizované prohledávání rozsáhlých databází nebo třeba práce se strojovým učením. V oblasti proteinového inženýrství se bioinformatika zabývá mimo jiné právě odhady vlivu mutace proteinu na jeho funkci na základě souhrnu výše zmíněných atributů. Výsledkem takového odhadu může být například informace, zda bude funkce proteinu po mutaci změněna, či nebude mít na funkci žádny vliv. Často se pracuje s anglickými termíny *disease-asociated mutation* (mutace spojená s nemocí z důvodu nesprávné funkce proteinu) a *neutral mutation* (mutace, u které nebyl zjištěn vliv na funkci proteinu).

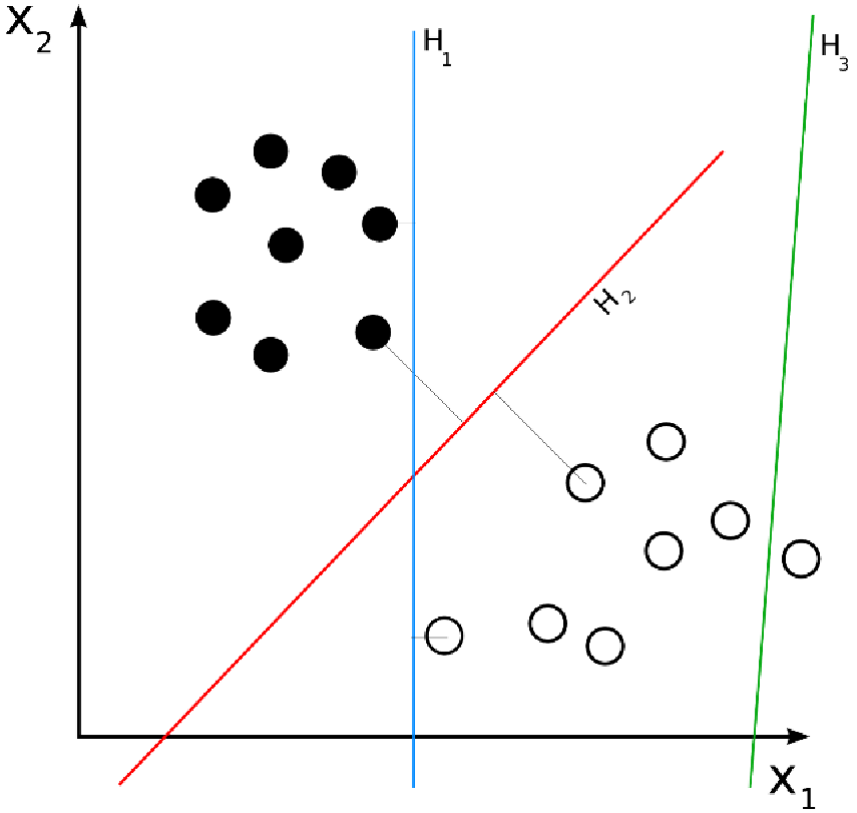
3.2 Strojové učení

Strojové učení je vědecká oblast zabývající se vývojem algoritmů, které počítačům umožňují nacházet v datech vzory. Tyto vzory pak mohou být využity pro rozhodování nebo porozumění datům.

3.2.1 Dělení metod strojového učení

Základními třemi typy strojového učení jsou: [16]

1. Řízené (supervised)
   Učící se systém dostane vzorek dat ve formátu vstup → výstup a podle nich vytváří mapovací funkci, která je poté schopna pro jakýkoli validní vstup generovat odhadovaný výstup
2. Neřízené (unsupervised)
   V tomto případě nejsou data v učícím vzorku ohodnocena. Cílem algoritmů neřízeného učení je zařadit data podle společných znaků – zjistit, jak jsou data organizována.



Obrázek 3.1: Ukázka výsledku práce SVM [26]. Prostor vzorků je zde dvojrozměrný. Přímky H_{1-3} ukazují možné rozdělení tohoto prostoru, H_2 je ideálním výsledkem SVM.

3. Polořízené (semi-supervised)
   Kombinuje oba přístupy: učící se stroj dostane na vstupu jak označená, tak neoznačená data. Tento postup je výhodný v případech, kdy je získávání označených dat závislé na zdlouhavém procesu ohodnocování ze strany člověka. Obvykle se při tomto způsobu používá malý vzorek ohodnocených vstupu a velký vzorek neohodnocených.

Příklady algoritmů strojového učení

Následují dva příklady algoritmů strojového učení, které se využívají v bioinformatických výpočtech [27].

Support vector machine[2] (SVM)

SVM je stroj patřící do skupiny řízených typů strojového učení. Cílem stroje typu SVM je najít nadrovinu, která prostor vzorků rozděluje tak, že její vzdálenost od nejbližších vzorků je co největší. SVM jsou v základní verzi binární (rozdělují vzorky do dvou skupin), ale existují i variace pro použití na vícerozměrných prostorech. Ukázka výsledku SVM nad dvojrozměrným prostorem je na ilustraci 3.1.

[2] Český název se zatím neustálil.

Neuronové sítě

Neuronová síť je jedním z výpočetních modelů používaných v umělé inteligenci. Vzorem pro vznik tohoto typu modelu byly právě neuronové sítě v biologickém slova smyslu. Neuronová síť v informatice slouží jako nástroj pro provádění paralelních výpočtů. Síť se skládá z množiny neuronů (jednotek, uzlů), které jsou mezi sebou spojeny a předávají si informace transformované přenosovými funkcemi. Neuronová síť má schopnost učení – výsledkem takového učení je nastavení sítě tak, aby dávala co nejoptimálnější výsledky. Rozlišujeme zde dvě varianty již zmíněných typů učení:

1. s učitelem
   Neuronové síti jsou předávány vstupy a na základě jejích výstupů, porovnaných se skutečnými výsledky, jsou upravovány vlastnosti sítě.

2. bez učitele
   Síť se v tomto případě učí sama: jedna možnost učení bez učitele je třídění vzorů do určitých skupin a následné reagování na typického zástupce skupiny, druhou možností je přizpůsobení své topologie vlastnostem vzorů.

3.3 Nástroje pro výpočet atributů proteinů a předpověď vlivu mutace na funkci proteinu

Dnes je již známo mnoho metod, použitelných k výpočtu atributů varianty proteinu. Některé tyto metody používají údaje o sekvenci nukleotidů/aminokyselin v proteinu, některé využívají údaje o struktuře a jiné tyto dvě skupiny kombinují. V této části shrnu některé ze současných aplikací, které tyto metody implementují, nebo využívají.

3.3.1 Nástroje pro výpočet atributů proteinů

FoldX

FoldX je metoda používající empirické silové pole, vyvinuté pro rychlé vyhodnocení efektu mutace na stabilitu a dynamiku proteinů a aminokyselin a jejich skládání v prostoru [22]. FoldX nejlépe pracuje při srovnání proteinu, který se běžně vyskytuje v přírodě, a jeho zmutované varianty. Nástroj vyhodnocuje celkovou volnou energii proteinu za použití atributů jako: elektrostatická interakce, síla vodíkových můstků, nebo Van der Waalsových sil mezi atomy.

CASTp (Compute Atlas of Surface Topology of proteins)

CASTp je nástroj, který hledá v 3D struktuře proteinu konkávní povrchové regiony, což zahrnuje kapsy a dutiny „pohřbené“ uvnitř molekuly [4]. Pro analýzu molekuly využívá moderní poznatky z oblasti výpočetní geometrie jako jsou alfa tvary a duální komplexy (alpha shapes, dual complexes [9]).

Metoda vyhledávání kapes, kterou používá CASTp, je zahrnuta v praktické části této práce a bude více popsána v další kapitole.

Framework Deleterious

V této práci bude použit pro kompilaci atributů framework Deleterious, o kterém se blíže zmíním v sekci 4.3.1 na straně 20.

3.3.2 Nástroje pro odhad vlivu mutace na základě vypočtených atributů

SIFT (Sorting Intolerant From Tolerant)

SIFT je nástroj, který k výpočtu předpovědi využívá údaje o sekvenci aminokyselin v proteinu [17]. Předpověď se zakládá hlavně na velmi důležitém atributu, kterým je konzervovanost aminokyseliny na určité pozici. Předpoklad výpočtu je takový, že aminokyseliny na důležitých pozicích budou evolučně zakonzervované: budou stejné ve více proteinech stejného typu. Mutace evolučně konzervované aminokyseliny tím pádem nejspíš povede k výrazné změně funkce proteinu, což v důsledku vede k ovlivnění jeho funkce natolik, že se stává změna pro organismus zhoubnou.

SNAP (Screening for Non-Acceptable Polymorphisms)

SNAP je nástroj založený na učení neuronových sítí, který umožňuje předpověď vlivu nsSNP na funkci proteinu [6]. Používá sekvenční údaje, ale lepších výsledků dosahuje, pokud jsou k dispozici i strukturní nebo funkcionální data o proteinu. K odhadu vlivu mutace na funkci proteinu používá mimo jiné přístupnost residua rozpouštědlu (mutované proteiny jsou rozděleny podle přístupnosti rozpouštědla do tří skupin: povrchové, středně zanořené (intermediate) a „pohřbené“ (buried)) a algoritmus BLAST pro srovnání sekvencí více proteinů.

3.4 Další vývoj v této oblasti

Čím více relevantních atributů je použito v systému strojového učení, tím větší je šance, že bude výsledný odhad správný. Je tedy žádoucí získávat o proteinech co nejvíce údajů a následnými experimenty zjišťovat, které z těchto atributů přispívají ke správnosti odhadů.

Kapitola 4

Analýza

Po dohodě s vedoucím práce byly vybrány atributy, které by měly pomoci zlepšit výsledky predikce. Tato kapitola je představí, uvede čtenáře do problematiky jejich výpočtu a popíše již existující aplikace, které jsou základem pro tyto výpočty a jejich následné použití.

4.1 Vybrané atributy

Cílem této práce je vyhledat a změřit kapsy ve struktuře proteinu a následně určit, zda se v některých mutované residuum[1] nachází. Praktickou část této práce lze rozdělit na algoritmus vyhledání kapes v proteinu a následné vypočítání různých atributů, které s kapsami souvisí. Výsledné atributy jsou kombinací naměřených hodnot a jiných vlastností proteinu, spočítaných pomocí knihoven přístupných ve frameworku. Vybranými atributy jsou:

- **Příslušnost residua v kapse** – binární atribut. Kladná hodnota znamená, že residuum se účastní struktury nějaké kapsy.
- **Objem kapsy** – číslo vyjadřující přibližný objem největší kapsy, ve které se residuum nachází (pokud taková kapsa existuje).
- **Příslušnost residua v katalytické kapse** – binární atribut. Kladná hodnota znamená, že v kapse, označené předchozími atributy, je přítomno katalytické residuum[2] a zároveň je relativně blízko zkoumanému residuu.
- **Vzdálenost k nejbližšímu katalytickému residuu v kapse**
- **Katalytické residuum** – binární hodnota, která je kladná, pokud je zkoumané residuum přímo označené jako katalytické

4.2 Hledání kapes v proteinu

Existuje mnoho přístupů k problematice vyhledávání kapes. Souhrn několika významných lze nalézt například ve článku [25]. V zásadě jde o dva přístupy:

[1] Aminokyselina v proteinu

[2] Residuum, které se účastní katalytické činnosti proteinu, získáno přes jiné nástroje přítomné ve frameworku

1. Rychlé hledání za použití virtuální výplně (dále *diskrétní řešení*). Výplní může být myšlena například trojrozměrná mřížka, kterou se struktura proteinu protkne. Následně se jako kapsa označí sousedící buňky, které uvnitř sebe neobsahují žádné atomy proteinu.

2. Pomalejší, ale velice přesné hledání kapes pomocí triangulace v prostoru (dále *analytické řešení*).

Metoda, použitá v této práci, vychází z výpočtů použitých v programu CASTp 3.3.1. Algoritmus využívá vlastností Delaunayho triangulace a algoritmu diskrétního toku. Výhody této metody analýzy jsou například:

- analytické řešení neposkytuje tolik prostoru pro chybu jako diskrétní řešení
- hranice mezi atomy molekuly a okolním rozpouštědlem je určená přesně, nikoli odhadem
- všechny vypočítané parametry jsou nezávislé na otočení molekuly v prostoru – výsledek analýzy je vždy stejný

Pro identifikaci kapes po provedení triangulace je použita metoda diskrétního toku (z angl. discrete flow). Stejnou metodu využívá aplikace CASTp, ale existuje pouze jako webový server, který je značně nestabilní a často je pomalý, nebo úplně nedostupný. Navíc jde o uzavřenou aplikaci a tím pádem zde není prostor pro případné rozšíření.

4.2.1 Delaunayho triangulace

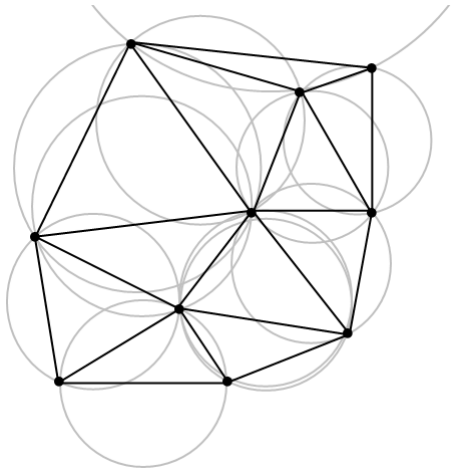
Delaunayho triangulace nad množinou bodů M na ploše vytvoří soubor trojúhelníků s následujícími vlastnostmi:

- každý trojúhelník má jako vrcholy body z M a každý bod je vrcholem nějakého trojúhelníku
- každé dva trojúhelníky sdílejí maximálně jednu hranu
- sjednocení všech trojúhelníků tvoří souvislou plochu
- pro každý trojúhelník platí, že uvnitř kružnice opsané tomuto trojúhelníku neleží žádný bod z M kromě bodů tvořících tento trojúhelník

Analogická pravidla platí i v trojrozměrném prostoru s několika úpravami: místo trojúhelníků jsou použity čtyřstěny, místo opsaných kružnic se používají opsané koule. Příklad Delaunayho triangulace na ploše je na ilustraci 4.1 a výpočet této triangulace je například na [12].

4.2.2 Diskrétní tok

Diskrétní tok je definován ve článku [7] jako relace mezi dvěma čtyřstěny Delaunayho triangulace. Pro každý čtyřstěn je potřeba nejdříve nalézt jeho *ortogonální centrum*.



Obrázek 4.1: Ukázka Delaunayho triangulace nad množinou bodů v dvojrozměrném prostoru [26]. Jsou zobrazeny i kružnice opsané každému trojúhelníku.

Ortogonální centrum čtyřstěnu

Jako *ortogonální centrum* označujeme střed koule čtyřstěnu opsané. Pojem *ortogonální* zde označuje to, že pokud vytvoříme v jakémkoli bodě umístěném na kružnici vzniklé průnikem dvou koulí tečné plochy k těmto koulím, budou na sebe kolmé. Pokud označíme vrcholy čtyřstěnu T jako body a, b, c, d a střed koule opsané jako m, získáme m podle vzorce 4.1.

$$m = a + \frac{|d-a|^2 * [(b-a) \times (c-a)] + |c-a|^2 * [(d-a) \times (b-a)] + |b-a|^2 * [(c-a) \times (d-a)]}{2 * (b-a) \cdot [(c-a) \times (d-a)]} \tag{4.1}$$

Relaci diskrétního toku označujeme $\tau \prec \sigma$, pokud σ, τ jsou čtyřstěny, pro které platí následující:

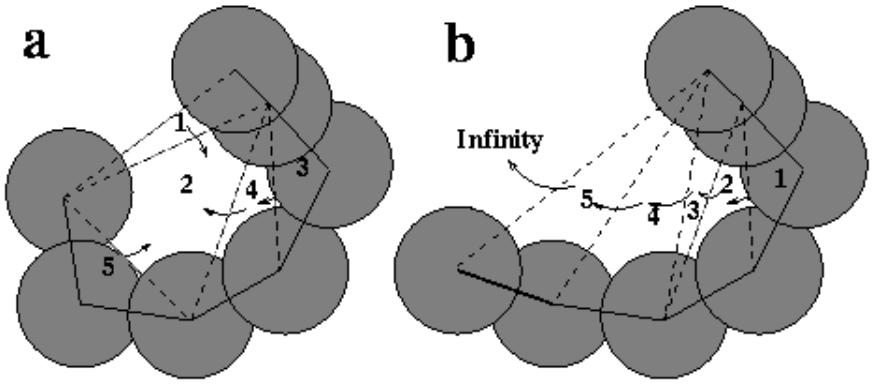
- σ a τ jsou sousedy – sdílí trojúhelník φ
- vrchol τ, který netvoří trojúhelník φ, a ortogonální centrum τ leží na opačných stranách roviny, která obsahuje všechny vrcholy φ

Zjištění, zda dva body leží na opačných stranách či na stejné straně plochy, se provádí pomocí upraveného vzorce na výpočet vzdálenosti bodu od roviny (viz vzorec 4.2). Pokud je výsledek pro oba zkoumané body různý, leží na opačných stranách plochy.

$$sign = sgn\left(\frac{ax_0 + by_0 + cz_0 + d}{\sqrt{a^2 + b^2 + c^2}}\right) \tag{4.2}$$

Pokud čtyřstěn obsahuje své ortogonální centrum, nazýváme tento čtyřstěn *výpustí (angl. sink)*. Z každého čtyřstěnu, který není výpustí, relace přechází do maximálně tří dalších čtyřstěnů, protože ortogonální centrum může být na odvrácené straně maximálně tří stěn čtyřstěnu. Každá dutina molekuly obsahuje výpusť.

Algoritmus nenachází vždy kompletní dutiny, ale pouze jejich části - každá taková část má svoji výpusť. Z definice relace totiž vyplývá, že velikost poloměru koule čtyřstěnu opsané se zvětšuje se směrem relace. Tím pádem počáteční čtyřstěny každého diskrétního toku jsou buď u stěn dutiny, nebo u úzkého hrdla. Principy diskrétního toku jsou zobrazeny na ilustracích 4.2 a 4.3.



Obrázek 4.2: Princip fungování diskrétního toku ve 2D. Obrázek A ukazuje situaci, která je vyhodnocená jako kapsa, kdežto B kapsou není. Trojúhelníky, které neobsahují své ortogonální centrum (v tomto případě střed kružnice opsané) "přetékají"ve směru tohoto centra. Obrázek získán na [1].

4.2.3 Měření kapes

U vyhledaných kapes je vypočítán jejich přibližný objem tak, že se vypočítá objem všech čtyřstěnů, které kapsu tvoří a následně se odečtou části atomů, které do čtyřstěnů zasahují. U některých uspořádání atomů ve čtyřstěnu vznikají situace, kdy je algoritmicky náročné vypočítat přesné objemy vzájemných průniků atomů a čtyřstěnu. Tyto složitější situace nejsou brány v potaz, protože pro strojové učení není relevantní přesný obsah kapsy. Důležité je identifikovat poměry mezi menší a větší kapsou. Rozdíl v poměrech přesných objemů kapes a přibližných jsou při vyjádření důležitosti kapes zanedbatelné.

Výpočet objemu čtyřstěnu se počítá podle vzorce 4.3

$$V = \frac{|(a-d)\cdot((b-d)\times(c-d))|}{6} \tag{4.3}$$

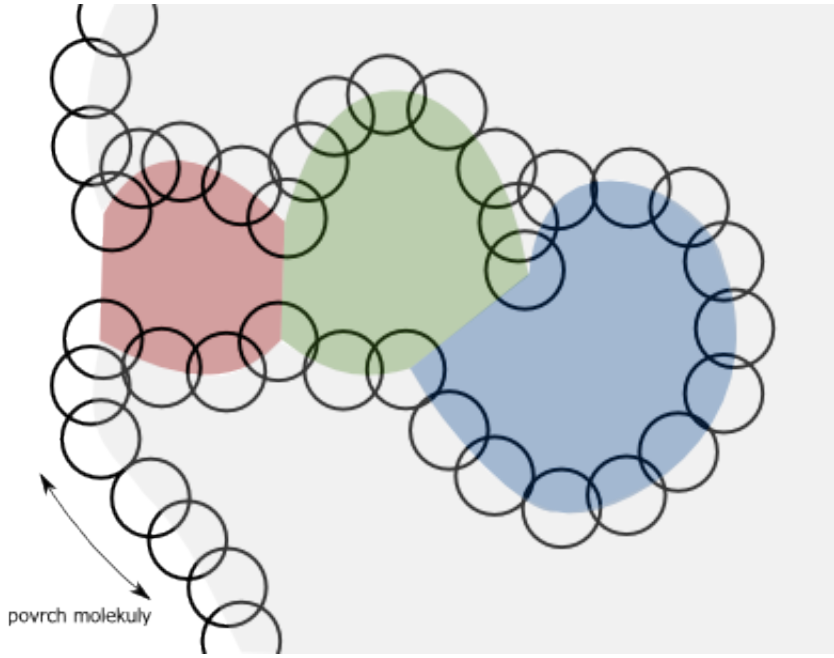
Průniky atomů s čtyřstěnem jsou vypočítány na základě některých algoritmů uvedených ve článku [8], kde jsou označené anglickými termíny *sector* a *wedge*.

4.2.4 Caver

Caver je aplikace používaná na analýzu 3D struktury proteinu a následné vyhledání tunelů v této struktuře. Tunelem je myšlena co nejširší souvislá cesta od povrchu proteinu k danému místu uvnitř proteinu [15]. Caver nad 3D strukturou proteinu vypočítá Delaunayho triangulaci a následně používá Dijkstrův algoritmus na průchod grafu k nalezení optimální cesty. Aplikace jako taková neobsahuje mechanismy pro zobrazení nalezeného tunelu, ale poskytuje výsledky použitelné ve specializovaných programech vytvořených pro tyto účely (například PyMOL, viz `http://www.pymol.org`).

Caver zatím nezahrnuje algoritmy pro hledání kapes, ale poskytuje struktury, které se dají k tomuto účelu použít. Proto bude v rámci této práce rozšířen o mechanismy výpočtu diskrétního toku.

Strukturu proteinu načítá Caver ze souborů ve formátu PDB.



Obrázek 4.3: Ukázka kapsy, která vzniká sjednocením několika dutin, které postupně našel algoritmus diskrétního toku. Kruhy znázorňují atomy, původní tři dutiny jsou označené postupně červenou, zelenou a modrou barvou. Pro jednoduchost je ukázka zobrazena ve 2D.

Formát PDB

Soubory PDB používají textový formát. Každý řádek popisuje jeden atom, o kterém uvádí mimo jiné jeho příslušnost ke konkrétní aminokyselině a jeho pozici v prostoru. Prostorové souřadnice atomů jsou definovány v kartézské soustavě souřadnic (viz obrázek 4.4). Hodnoty na řádku jsou odděleny mezerami a zarovnány tak, aby začínaly na stejných pozicích v řádku. Tabulka 4.1 ukazuje příklad souboru ve formátu PDB. Databáze obsahuje velké množství struktur známých proteinů. Proteiny jsou v databázi značené zkratkami ze čtyř znaků (tzv. PDB ID). Grafické znázornění proteinu z PDB databáze je na ilustraci 4.5.

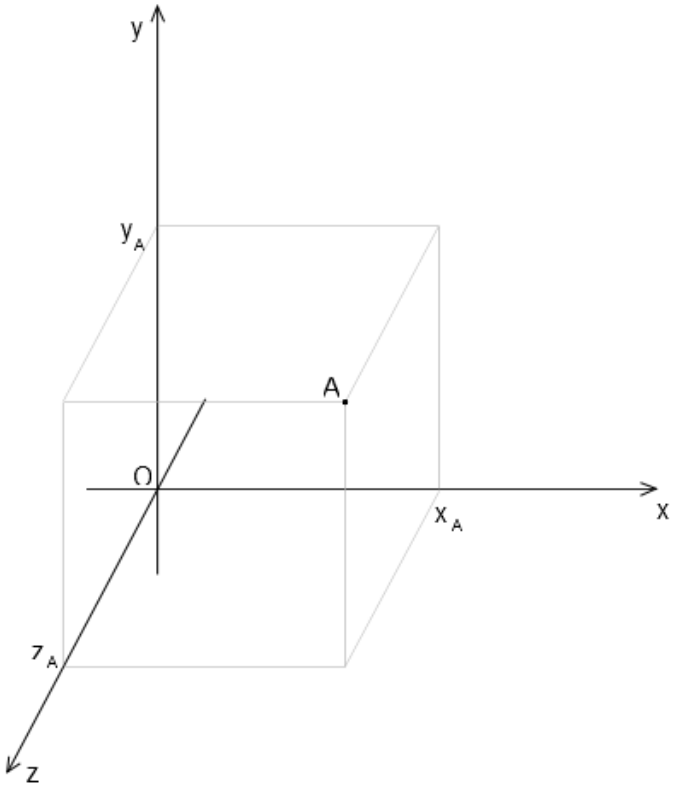
Přístup k databázi je na webovém rozhraní umístěném na adrese `http://www.pdb.org`. Na stránce `http://www.wwpdb.org/docs.html` lze nalézt podrobnější dokumentaci k formátu souborů PDB.

4.3 Výpočet atributů ze získaných hodnot

Nové funkce Caveru umí vyhledat kapsy a změřit jejich objem. Tyto údaje jsou následně zpracovány ve frameworku Deleterious, kde jsou z nich vytvořeny konkrétní atributy použitelné pro následné strojové učení.

4.3.1 Deleterious – framework na výpočet atributů

Aplikace s názvem Deleterious je určena pro spojení výpočtů z různých zdrojů za účelem jejich hromadného použití v odhadu vlivu mutace na protein. Jak již napovídá označení *framework*, nejde o koncovou aplikaci, ale spíše o sadu nástrojů pro vývojáře. Od doby, kdy



Obrázek 4.4: Kartézská soustava souřadnic je taková soustava souřadnic, u které jsou souřadné osy vzájemně kolmé a protínají se v jednom bodě - počátku soustavy souřadnic [26].

se frameworkem zabýval ve své práci (viz [14]) Bc. Jiří Matějíček, byla struktura částečně změněna a aktuálně se nepoužívá uživatelské rozhraní, které bylo jedním z výsledků jeho práce.

Pro tuto práci byly použité nástroje frameworku, které mají jako vstup seznam mutací proteinů s jejich různými atributy – může jít například o binární nebo číselné hodnoty vyjadřující nějakou změřenou, odhadnutou nebo vypočítanou vlastnost mutovaného proteinu. Výstupem frameworku je soubor typu ARFF (viz strana 23), který obsahuje vstupní data pro aplikaci WEKA (viz strana 23).

V aplikaci se používají dva hlavní typy modulů, které přímo pracují s vzorky pro strojové učení:

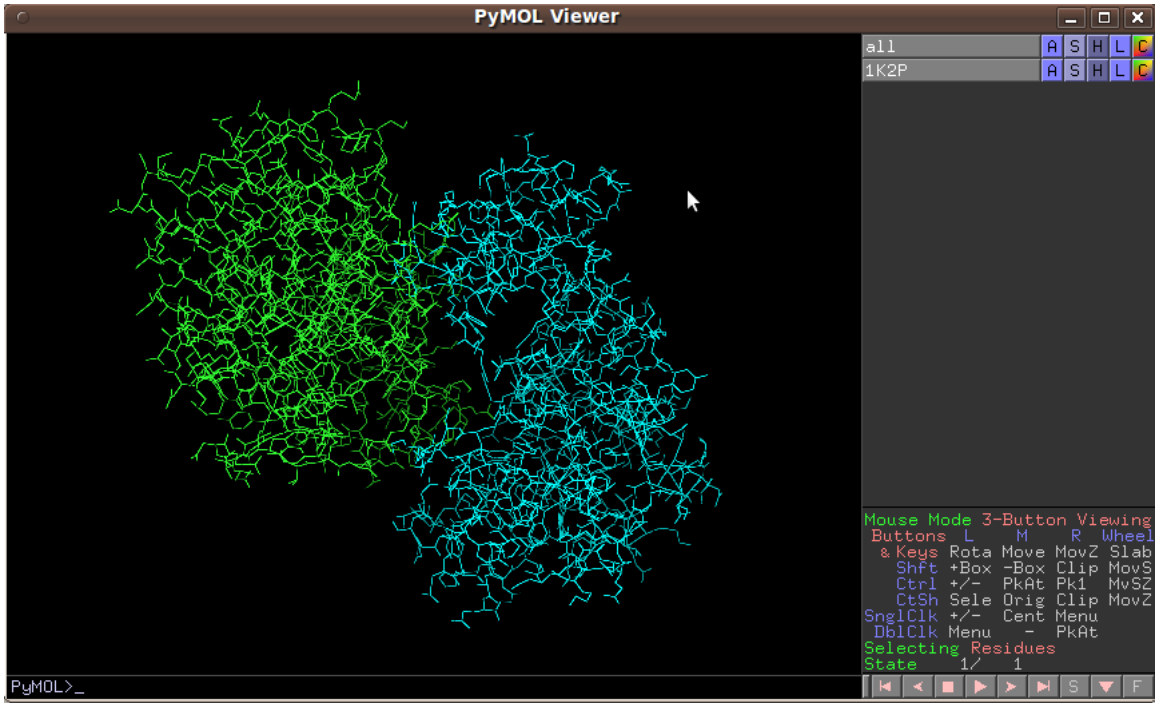
1. Překládací moduly, které mají za úkol formátování struktury různých vstupních dat na ARFF soubory

2. Výpočetní moduly, které pracují se soubory formátu ARFF, do kterého přidávají další hodnoty.

V aktuální verzi jsou dostupné mimo jiné moduly druhého typu vypsané v tabulce 4.2.

ATOM	51	CE2	PHE	A	21	4,371	85,533	38,534	0,83	13,62	C
ATOM	52	CZ	PHE	A	21	5,458	85,265	39,362	1,00	13,60	C
ATOM	53	N	ASP	A	22	3,655	81,902	33,110	1,00	13,57	N

Tabulka 4.1: Ukázka formátu souborů PDB. Jde o několik atomů z proteinu 1cqw. První řádek popisuje atom s pořadovým číslem 51, který je součástí aminokyseliny fenylalaninu s ID 21. Souřadnice X, Y a Z jsou první tři číselné hodnoty na řádku a poslední sloupec udává typ prvku (v tomto případě uhlík).



Obrázek 4.5: 3D struktura proteinu 1K2P zobrazená v aplikaci PyMOL. Tento protein je složený ze dvou řetězců, které jsou znázorněny odlišnými barvami.

Modul	Příklad přidaného atributu
MSAExpander[3]	evoluční konzervovanost
NeighbourExpander	počet atomů v blízkosti residua
FoldXExpander	energie aminokyseliny

Tabulka 4.2: Příklady výpočetních modulů dostupných ve frameworku Deleterious

```
@relation 'deleterious'
@attribute PDB string % pdb code
@attribute IN_POCKET {TRUE,FALSE} % true if the residue is in a pocket
@attribute POCKET_VOLUME numeric % only relevant if IN_POCKET is TRUE

@data
1B2X,TRUE,248.83691
1LBI,FALSE,0.0
```

Obrázek 4.6: Ukázka souboru formátu ARFF. Jde o výňatek z výsledného souboru po nalezení kapes. Jsou zde ukázány tři typy atributů: řetězec znaků, výčtový typ a číselná hodnota. Znak % označuje začátek komentáře.

Formát ARFF

ARFF je zkratkou pro Attribute-Relation File Format. Soubory tohoto typu popisují relaci strojového učení, typy atributů, které se v relaci vyskytují a pak soupis hodnot těchto atributů. Při definici typů atributů se klíčová slova oddělují mezerami, každý vzorek je na novém řádku a hodnoty atributů jsou pak oddělovány čárkami. Ukázka souboru ve formátu ARFF je na ilustraci 4.6.

4.3.2 WEKA

Zkratka WEKA pochází z anglického Waikato Environment for Knowledge Analysis [10]. Jde o aplikaci, která má cíl otevřít odborníkům z nejrůznějších oborů možnosti strojového učení. Součástí aplikace je několik grafických uživatelských rozhraní, které usnadňují práci se systémem. Aktuální verze je z řady 3.X a je psaná kompletně v Javě. WEKA bere jako vstupní data soubory s různými formáty jako například CSV[4], ale nativním formátem je ARFF. WEKA umí se zdrojovými daty pracovat nejrůznějšími způsoby. Použití aplikace WEKA pro účely této práce je podrobně vysvětleno v kapitole 6.

Klasifikátory

Klasifikátor je funkce, která zařadí každý vzorek dat podle jeho atributů do jedné z výsledných tříd. Výroba klasifikátoru je procesem strojového učení na skupině vzorků.

Při strojovém učení byly použity tyto klasifikátory:

Naivní Bayesův klasifikátor

Tento jednoduchý klasifikátor je založený na Bayesově teorému. Klasifikátoru byl odvozen z předpokladu, že jednotlivé atributy jsou na sobě nezávislé. Více o Bayesovských klasifiká-

[4] Čárkou oddělené hodnoty, z angl. Comma-Separated Values

torech lze nalézt v publikaci [11].

Náhodný les (Random Forest)

Náhodný les je metoda kombinující náhodně generované stromy, které o výsledném rozhodnutí hlasují. Více v [5].

SMO[5]

Tato metoda patří do skupiny SVM (viz strana 13) a implementuje algoritmy popsané v publikaci [20].

J48

J48 vytváří rozhodovací stromy typu C4.5, více v práci [21].

[5] Sekvenční minimální optimalizace, z anglického Sequential Minimal Optimalization

Kapitola 5

Implementace

Implementace navržených algoritmů sestává ze dvou hlavních částí:

1. Rozšíření aplikace Caver o hledání a měření kapes
2. Vytvoření modulu do frameworku Deleterious, který přidá údaje získané z Caveru k existujícím atributům

V této kapitole budou tyto dva body podrobně popsány. Pro přehlednost uvádím vždy nejprve přehled API dané aplikace a pak teprve popis samotné implementace

5.1 Caver

5.1.1 API Caveru

Aplikace Caver je napsaná v programovacím jazyce Java. Páteří celého projektu je balík tříd nazvaný AnBeKoM[1], který obsahuje pár základních tříd, na kterých je hledání kapes založeno. Dalším důležitým balíkem je `mymath`. Oba tyto balíky zde budou popsány detailněji.

Balík `AnBeKoM`

`Molecule` Třída obsahuje veškeré údaje o proteinu (mimo jiné soupis residuí a atomů a jejich názvů) a po výpočtu Delaunayho triangulace také poskytuje přístup k vypočteným čtyřstěnům přes pole s objekty třídy `Tetra`. Také obsahuje metody, které umožňují různým objektům v systému spolupracovat.

`InputLoader` Třída zpracovává vstupní PDB soubor a načítá z něj potřebné struktury pro běh aplikace.

`Tetra` Objekty této třídy jsou čtyřstěny vzniklé při Delaunayho triangulaci. Každá instance tedy obsahuje identifikaci čtyř atomů (objekt třídy `Atom`), které ji tvoří. Dále skrze metody této třídy lze zjistit, zda dané souřadnice leží uvnitř čtyřstěnu, nebo získat poloměr otvoru mezi atomy některé ze stěn čtyřstěnu.

[1] Název je zkratkou jmen původních autorů aplikace

`Atom` Instance třídy zastupuje atom molekuly a obsahuje mimo jiné informaci o příslušnosti ke konkrétnímu residuu.

`DelaunayTriangulation` Tato abstraktní třída je základem pro několik různých přístupů k výpočtu Delaunayho triangulace. Tyto metody se liší rychlostí výpočtu a také poskytují poněkud jiné výstupy. Rozdílné přístupy k triangulaci ale vedou k velice podobným výsledkům.

balík `myMath`

Funkce spojené s umístěním objektů Caveru do prostoru jsou implementovány v balíku myMath. Mezi významné třídy zde patří:

`Vertex3f` Každá instance této třídy identifikuje bod nebo vektor v prostoru. Každý objekt je definován třemi souřadnicemi (kartézská soustava souřadnic, viz obrázek 4.4). Třída dále poskytuje metody pro provádění základních vektorových operací - velikost vektoru, vzdálenost dvou bodů, součet a násobení vektorů, vektorový a skalární součin atp.

`Vertex4f` Tato třída je podobná třídě `Vertex3f` s tím rozdílem, že místo bodu popisuje kouli v prostoru. Kromě tří souřadnic, které zde vyjadřují pozici středu koule, obsahuje instance této třídy ještě poloměr koule.

5.1.2 Implementace algoritmu hledání kapes

Pro účely hledání kapes byly přidány nové metody do stávajících tříd `Tetra` a `Molecule`. Zároveň byl vytvořen balík `pockets` obsahující nové třídy. Ve většině objektů se u výpočetně náročných operací používá *líne inicializace (angl. lazy initialization)*, všechny výpočty tedy probíhají až ve chvíli, kdy jsou výsledné hodnoty potřeba. Všechny vypočtené hodnoty jsou v objektech uloženy pro opakované použití, protože při sběru atributů je potřeba získat údaje o kapsách jednoho proteinu vícekrát.

Dutiny v molekule se hledají od výpustí, čili proti směru relace diskrétního toku. Kromě podmínek relace se ještě ověřuje šířka průchodu mezi dvěma čtyřstěny, která musí být natolik velká, aby mezi čtyřstěny prošel atom jiné molekuly. Tato šířka je nastavena na 1,4 Angströmů[2].

Úpravy ve stávajících třídách

Třída `Tetra` První metodou přidanou do této třídy je `initCircumsphere`, která vypočítává souřadnice středu koule čtyřstěnu opsané podle vzorce 4.1.

Další podstatnou metodou je `initFlowUnit`, která podle pozice středu opsané koule zjišťuje, do kterých sousedních čtyřstěnu tento čtyřstěn teče (viz sekce 4.2.2). Algoritmus výpočtu je na 5.1.

[2] Angström je mezinárodně uznávaná jednotka používaná v souvislosti s molekulární biologií. 1 Ang = 10^{-10} metrů

Algoritmus 5.1 Identifikace čtyřstěnů, do kterých směřuje relace diskrétního toku. Metoda *sign* používá vzorec 4.2 pro zjištění, zda je bod na jedné, či druhé straně plochy. *opposite* je vrchol čtyřstěnu, který neleží na ploše zkoumané stěny.

```
i = 0;
for (side in tetra.sides){
    if (side.sign(center) != side.sign(opposite)) {
        flowTo[i] = tetra.getNeighbour(i);
        i++;
    }
}
```

Třída `Molecule` Inicializaci struktur potřebných pro výpočet kapes zařizují po řadě metody `initializeFlow` a dále pak `findAllVoids`. První prochází všechny objekty třídy `Tetra` a zjišťuje, zda jde o výpusti, nebo ne. Pokud jde o výpusť, tak se ještě ověří, zda vede do čtyřstěnu alespoň z jedné strany cesta o minimálním poloměru (viz výše). Tato podmínka filtruje výpusti, které nejsou přístupné pro atomy jiných molekul. Pokud výpusť projde podmínkou, je zařazena do lokálního seznamu `sinks`. Druhá metoda, `findAllVoids`, prochází tento seznam a nad každou výpustí vytvoří instanci objektu třídy `DiscreteFlow` a volá její metodu `flow`, čímž je vytvořena nová kapsa. Po nalezení kapes proběhne ještě jejich spojování, o což se stará metoda `uniteVoids` voláním příslušných metod třídy `MVoid`.

Nalezené kapsy lze získat voláním `getPockets()` bez parametrů, což vrátí všechny nalezené kapsy. Druhou možností je předání parametrů `chain` a `position`, které definují residuum, jehož atomy mají v kapse ležet.

Nově přidané třídy (balík `pockets`)

Třída `IDGenerator` V systému je vytvářeno mnoho objektů, které je potřeba od sebe odlišit a zároveň nemají žádné vlastnosti, ze kterých by šel jednoduše vypočítat unikátní identifikátor. Proto vznikla statická třída `IDGenerator`, která vnitřně mapuje třídu na číslo. Toto číslo vyjadřuje aktuální čítač objektů klíčové třídy. `IDGenerator` poskytuje metody pro získání dalšího volného ID pro danou třídu, takže lze inicializovat objekty, aniž by jim bylo nutné nějaké ID předávat při konstrukci.

Třída `MVoid` Instance této třídy identifikují kapsy. Obsahují seznam čtyřstěnů a atomů, které kapsu tvoří. Kromě metod sloužících k přístupu k vlastnostem kapsy je zde několik dalších důležitých metod: `calculateVolume` provede součet objemů všech čtyřstěnů kapsy, `connectTo` slouží ke spojování více kapes (pokud navazují, viz obrázek 4.3) a `savePyMol` uloží na cestu definovanou Caverem soubor, který je použitelný jako vstup do zobrazovacího programu PyMOL.

Třída `DiscreteFlow` Objekt diskrétního toku je určen výpustí, do které stéká $0 - n$ čtyřstěnů. Metoda `flow` spouští proces prohledávání sady čtyřstěnů zavoláním metody `flowStep` nad touto výpustí. Bližší popis je v algoritmu 5.2. Metoda `getVoid` sestaví z nalezených čtyřstěnů objekt třídy `MVoid` a tento objekt vrátí.

Algoritmus 5.2 Diskrétní tok - algoritmus. Projde všechny stěny (`sides`) čtyřstěnu `tetra`. Následně, pokud je průchod mezi čtyřstěny dostatečně velký a pokud je aktuální čtyřstěn cílovým v relaci diskrétního toku, rekurzivně se volá funkce `flowStep` na sousední čtyřstěn. Proces se spouští na výpusti `sink`.

```
flowStep(sink);
function flowStep(tetra) {
   pocketTetras.add(tetra);
   flow = false;
   for (side in tetra.sides) {
      if (side.hole ≥ minimalRadius){
         destinations = tetra.getNeighbour(side).getFlowDestinations();
         for (destination in destinations) {
            if (destination == tetra) {
               flowStep(tetra.getNeighbour(side);
               flow = true; break;
            }
         }
         if (flow == false) {
            pocketOpenings.add(new PocketOpening(side));
         }
      }
   }
}
```

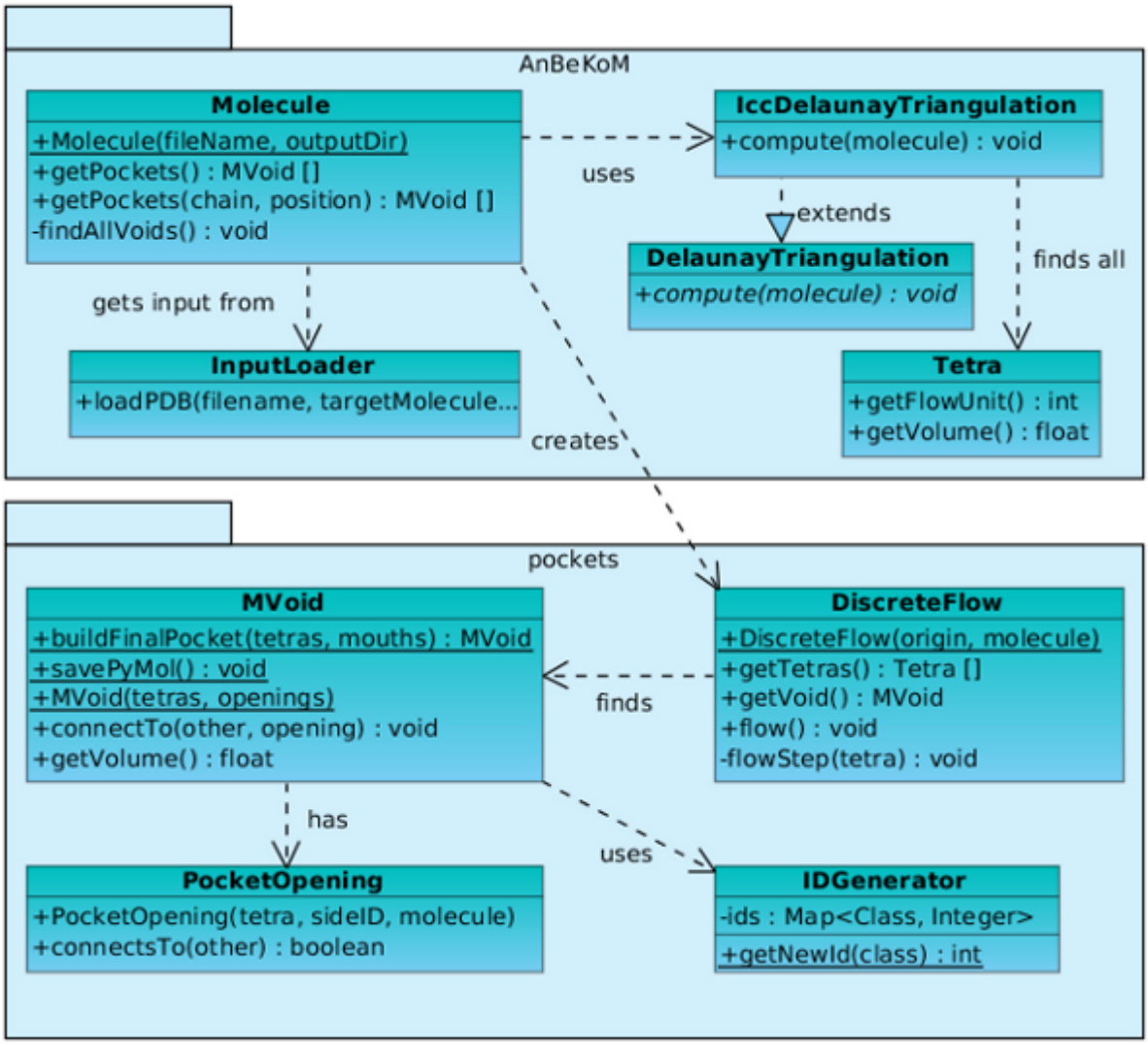
Třída `PocketOpening` Instance této třídy reprezentují ústí kapsy. Ústí je stěna čtyřstěnu, která je sice dostatečně široká pro vstup atomu cizí molekuly, ale neúčastní se relace diskrétního toku spojené s touto dutinou. Takové ústí je pak použito při spojování více kapes. Pokud má dutina nějaké ústí i po spojování kapes, jde o ústí vedoucí na povrch molekuly a tím pádem je dutina kapsou. Třída poskytuje metodu `connectsTo`, která vyhodnotí pravdivost napojení tohoto ústí na jiné. Objekty jsou inicializovány během diskrétního toku (algoritmus 5.2).

Třída `PocketException` Nestandardní situace jsou v přidaném systému kapes řešeny objekty této výjimkové třídy. Inicializují se řetězcem vysvětlujícím podstatu výjimky.

Diagram tříd zobrazuje dva důležité balíčky aplikace Caver použité při hledání kapes spolu s třídami a jejich metodami. Diagram je na obrázku 5.1.

5.2 Framework Deleterious

Práce frameworku sestává ze dvou částí: nejprve jsou napočítány a shromážděny základní atributy, potřebné pro další zpracování. Jde například o identifikaci proteinu nebo o údaje o aminokyselině, která je měněna. Tuto část je možné přeskočit, pokud je jako vstup poskytnut soubor s připravenými základními atributy. Druhou fází je spouštění jednotlivých modulů, které do souhrnu atributů přidávají nové na základě buď přímo implementovaných



Obrázek 5.1: Diagram významných balíčků, tříd a metod aplikace Caver, které jsou použity při hledání kapes

výpočtů, nebo voláním externích aplikací. Framework je napsaný v programovacím jazyce Java.

Framework obsahuje mimo jiné knihovnu pro práci s aplikací *HotSpot Wizard*. Jde o webový server, který slouží k identifikaci aktivních míst (angl. Hot Spots) v proteinu [18]. Tato knihovna se používá pro výpočet atributů souvisejících s katalytickými kapsami. Navíc v rámci svého výpočtu knihovna stahuje PDB soubor k aktuálnímu proteinu, který se využije k inicializaci objektu molekuly z projektu Caver.

V současné verzi frameworku sice není žádné uživatelské rozhraní, ale pro účely testování a získávání výsledků je zde třída `deleterious.Main`, ve které jsou připravené základní moduly a ukázky jejich použití. Pro účely demonstrace této práce byla třída upravena tak, aby byla aplikace spustitelná z příkazové řádky. Použití je vysvětleno v příloze A.

Název	Možné hodnoty	Význam
IN_POCKET	TRUE,FALSE	příslušnost residua k nějaké kapse
POCKET_VOLUME	desetinné číslo	objem kapsy
IN_CAT_POCKET	TRUE,FALSE	příslušnost residua ke katalytické kapse a blízkost ke katalytickému residuu
DISTANCE_TO_POCKET_AS	desetinné číslo	vzdálenost k nejbližšímu katalytickému residuu v kapse
CATALYTIC	TRUE,FALSE	TRUE, pokud je zkoumané residuum katalytické

Tabulka 5.1: Typy atributů počítaných modulem `PocketExpander`

5.2.1 Přidání obsluhy Caveru do projektu Deleterious

Pro přidávání dalších modulů do frameworku slouží systém sestávající z abstraktní třídy `deleterious.attributes.AbstractAttributeExpander` (dále `Expander`) a následné konfigurace spuštění modulu ve třídě `deleterious.Main`. Abstraktní metody, které je třeba při rozšiřování `Expanderu` implementovat, jsou tyto tři: `newPdbId`, `getAttributeTypes` a `calculateAttributes`. Hlavní funkcí Expanderu je načtení dat z modulů, které už svou práci ukončily a následné přidání nových atributů. Tato akce se spouští na instanci konkrétního Expanderu voláním metody `expand`.

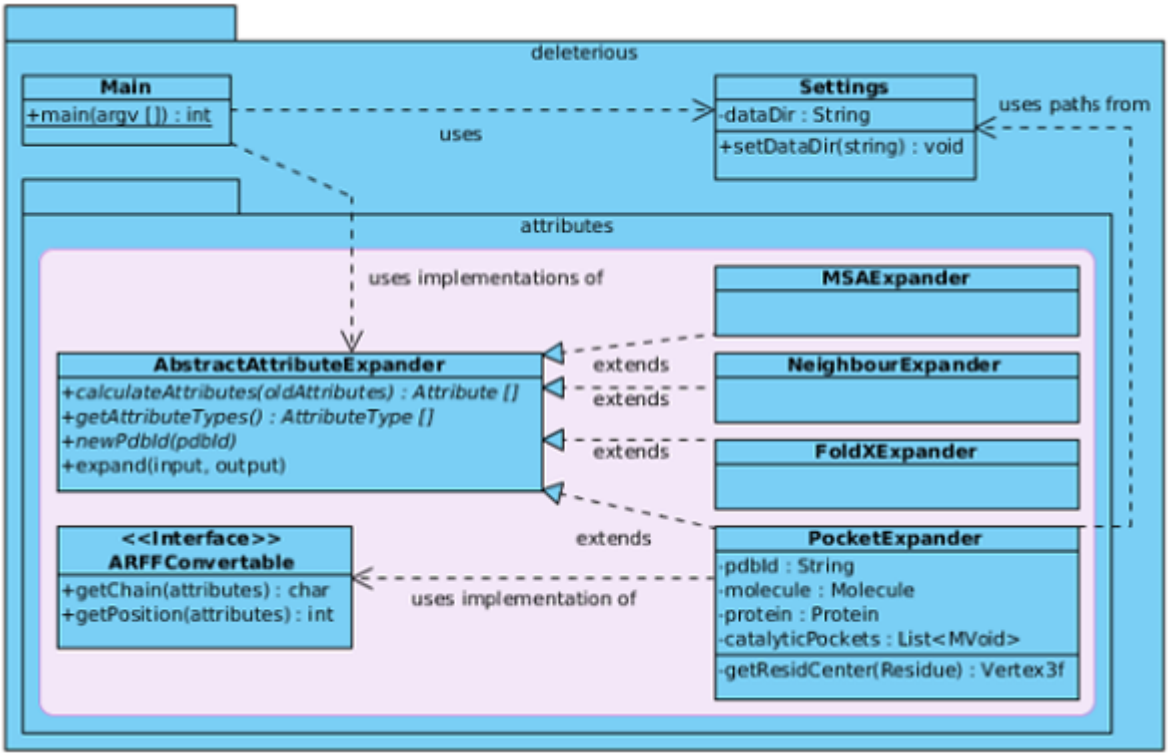
Třída `PocketExpander`

Metoda `newPdbId` slouží k inicializaci struktur pro konkrétní protein. V `PocketExpanderu` se tato metoda používá na vytvoření instance třídy `Molecule` z balíku `Caver.AnBeKoM`, čímž se napočítá triangulace atomů molekuly. Také se zde vytvoří objekt proteinu z knihovny HotSpot Wizardu, který obsahuje informace o pozici a složení aktivních míst proteinu. HotSpot Wizard umí vrátit ID residuí, která se podílí na aktivních místech. Pro identifikaci, zda se residuum podílí na struktuře kapsy, je potřeba ho přes toto ID spojit s příslušnými atomy ve struktuře molekuly z Caveru.

Nakonec se zde naplní struktury, které slouží k urychlení dalšího výpočtu atributů. Jde o dva seznamy: seznam katalytických kapes, seznam katalytických residuí a tři mapy: ke každé kapse se přiřadí jeho aktivní místa a na aktivní residuum se mapuje jeho vypočítané těžiště a seznam jeho atomů.

Účelem metody `getAttributeTypes` je poskytnout informace o atributech, které Expander počítá. Vracenou strukturou je pole instancí třídy `AttributeType`, která je sestavena s parametry *název atributu*, *datový typ hodnoty (popřípadě výčet hodnot)* a *komentář k atributu*. Atributy poskytované PocketExpanderem jsou uvedeny v tabulce 5.1.

Metoda `calculateAttributes` nejdříve z předchozích atributů vyčte hodnoty *chain* a *position*, které následně předá jako parametry metodě `getPockets` privátní instance třídy `Molecule`. Metoda vrací pole instancí třídy `Attribute`, která se vytváří s parametry určujícími typ atributu a jeho hodnotu. V případě, že residuum v kapse není, je objem kapsy nastavený na hodnotu 0. U atributu *vzdálenost k nejbližšímu katalytickému residuu v kapse* je v případě, že residuum není v katalytické kapse, nastavena vzdálenost na 88. Tato hodnota je tak velká, že ji algoritmy strojového učení nepřiřazují k negativnímu efektu mutace.



Obrázek 5.2: Diagram významných balíčků, tříd a metod frameworku Deleterious, které jsou použity při vytváření nových atributů (externí knihovny nejsou v diagramu zmíněny)

Diagram tříd zobrazuje dva důležité balíčky frameworku Deleterious použité při vytváření nových atributů společně s jejich třídami a metodami. Diagram je na obrázku 5.2.

Kapitola 6

Testování a výsledky

V této kapitole budou popsány principy vyhodnocení výsledků poskytnutých aplikací Deleterious spolu s novými atributy. Jedná se hlavně o zjištění poměru správných a špatných výsledků predikce při použití dat s novými atributy. Také budou uvedeny některé významné hodnoty, které z těchto testů vyplynuly.

6.1 Použitý dataset

Atributy byly vygenerovány na datasetu[1] z projektu AUTO–MUTE (viz http://proteins.gmu.edu/automute). Tento dataset obsahuje mutace lidských proteinů, které byly původně získány z databáze Swiss-Prot (viz http://www.expasy.org/sprot). Dataset obsahuje mimo již napočítaných atributů i vše potřebné pro PocketExpander - PDB ID a pozici aminokyseliny, která byla mutována. Každá mutace je také označena jednou ze tří tříd:

- *neutral (nt)* je mutace, která neměla vliv na efekt proteinu
- *disease-associated (da)* má souvislost s nemocí, tedy mutace, která způsobuje změnu v chování nebo stabilitě proteinu
- *undefined (un)*, čili nedefinovaná, je mutace, u které nebyl vliv na funkci nebo stabilitu popsán

Tento dataset byl nejprve přeformátován na soubor ARFF a následně byl předán aplikaci s frameworkem Deleterious, kde byly vypočítány všechny atributy. Protože bylo cílem otestovat přínosnost nových atributů pro strojové učení, byl v Deleterious použit pouze nový modul, PocketExpander.

Pro získání správných výsledků bylo potřeba nejdříve určit správně hodnotu, která limituje atribut IN_CAT_POCKET (minimální vzdálenost ke katalytickému residuu v kapse). Proto byl pro vzorek nejdřív napočítán samotný atribut DISTANCE_TO_POCKET_AS. Po analýze výsledků byly jako vhodné limity vybrány hodnoty 8, 16, 24, 32 a neomezeně Angströmů a ke každému tomuto atributu byl tedy vygenerován zvláštní atribut.

Originální dataset obsahuje skupiny mutací téhož proteinu a přitom všechny mutace jedné skupiny často patří do jediné třídy. To by mohlo vést k objevení zcela nežádoucího vzoru "patří-li mutace do jistého proteinu, pak patří do jisté třídy". I přes vymazání atributu s identifikátorem proteinu nelze identifikaci proteinu z ostatních atributů vyloučit. Proto

[1] Termín často používaný pro označení sady dat, na které se bude aplikovat strojové učení

Dataset	J48	NaiveBayes	RandomForest	SMO
A	58,81	63,26	58,14	60,13
B	61,45	**70,78**	59,39	64,77
C	60,51	**69,75**	58,00	64,89

Tabulka 6.1: Porovnání křížové validace datasetů. Číselné hodnoty vyjadřují procenta správných odhadů. Tučně označené hodnoty jsou o více jak 10% vyšší. A je původní dataset bez nových atributů, B obsahuje atributy IN_CAT_POCKET a CATALYTIC a C má i zbylé počítané atributy.

byla v datasetu ponechána právě jedna mutace z každého proteinu. Přitom byly upřednostněny mutace nacházející se v katytických kapsách. Nakonec byly pro zjednodušení analýz odstraněny da mutace tak, aby dataset obsahoval stejný počet da a nt mutací.

Při porovnání vždy pracuji s datasetem A, který neobsahuje žádné nové atributy. Dále použiji dataset označený jako B, který má nové atributy CATALYTIC a všechny varianty IN_CAT_POCKET (viz tabulka 5.1). Poslední dataset je označen C a obsahuje oproti B navíc ještě atributy IN_POCKET, POCKET_VOLUME a DISTANCE_TO_POCKET_AS.

6.2 WEKA - Křížová validace

Po napočítání všech atributů byly z výsledného ARFF souboru odstraněny značkovací atributy, které nevypovídají o vlastnostech mutací, ale jen je popisují. Jde o PDB ID, pozici residua a jaká aminokyselina byla na této pozici mutována. Následně bylo nutné určit, zda a o kolik se zlepšil poměr správných předpovědí oproti špatným. Na toto zjištění bylo použito *křížové ověření* (také *křížová validace*[2]).

Nejprve se dataset rozdělí náhodně na X složek. Klasifikátory se učí na $X-1$ *trénovacích* skupinách a poté se zjišťuje počet správných a špatných výsledků predikce na údajích z *testovací* skupiny, která ve trénování nebyla. Toto se poté opakuje tak, aby postupně každá skupina byla jednou testovací a $(X-1)$krát učící. Po každém učení a testování se zjistí, v kolika procentech případů naučený algoritmus správně odhadl výslednou hodnotu. Výsledná správnost se vypočítá jako průměr dílčích výsledků.

Byly vybrány čtyři různé metody klasifikace: J48, Naive Bayes, Random Forest a SMO. Počet složek pro křížovou validaci (Ve WEKA se nazývají anglicky *folds*) byl nastaven na 10.

6.2.1 WEKA Experimenter

Experimenter umožňuje porovnání výsledků různých datasetů a různých algoritmů strojového učení. Byly použity všechny tři uvedené datasety. Pro porovnání výsledků byly použity dvě výsledné hodnoty: procentuální vyjádření správnosti odhadu a plocha pod křivkou Receiver Operating Characteristic[3] (ROC). Pro náhodný klasifikátor vychází hodnota ROC plochy 0,5 a pro dokonalý klasifikátor vyjde 1 - čím vyšší je hodnota, tím je lepší predikce.

Výsledky křížové validace jsou zobrazeny v tabulkách 6.1 a 6.2.

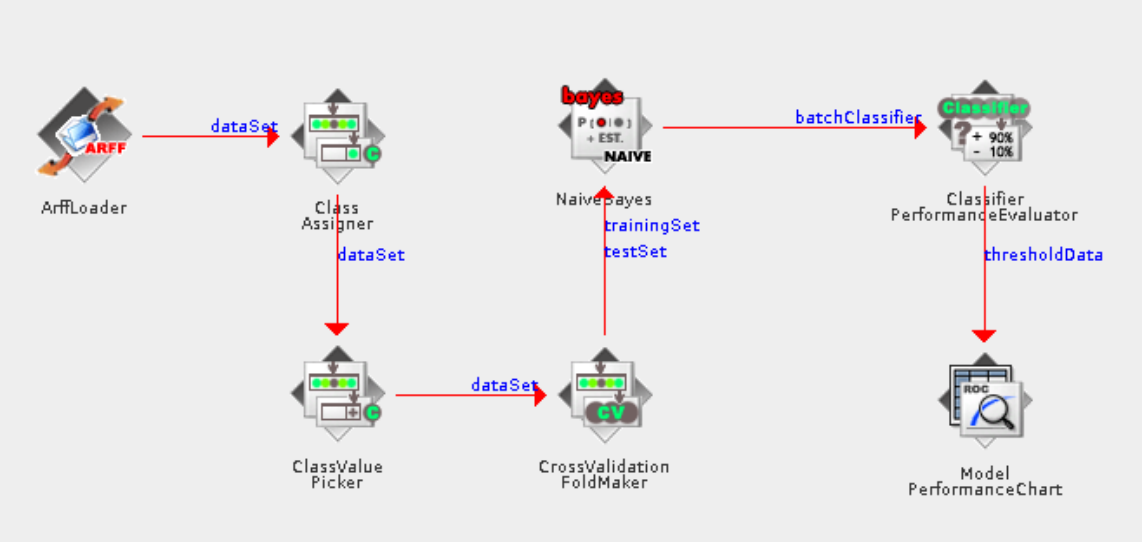
[2] Často se používá anglický termín cross-validation
[3] český termín se zatím neustálil

Dataset	J48	NaiveBayes	RandomForest	SMO
A	0,58	0,69	0,61	0,64
B	0,62	**0,76**	0,64	0,69
C	0,61	**0,76**	0,62	0,69

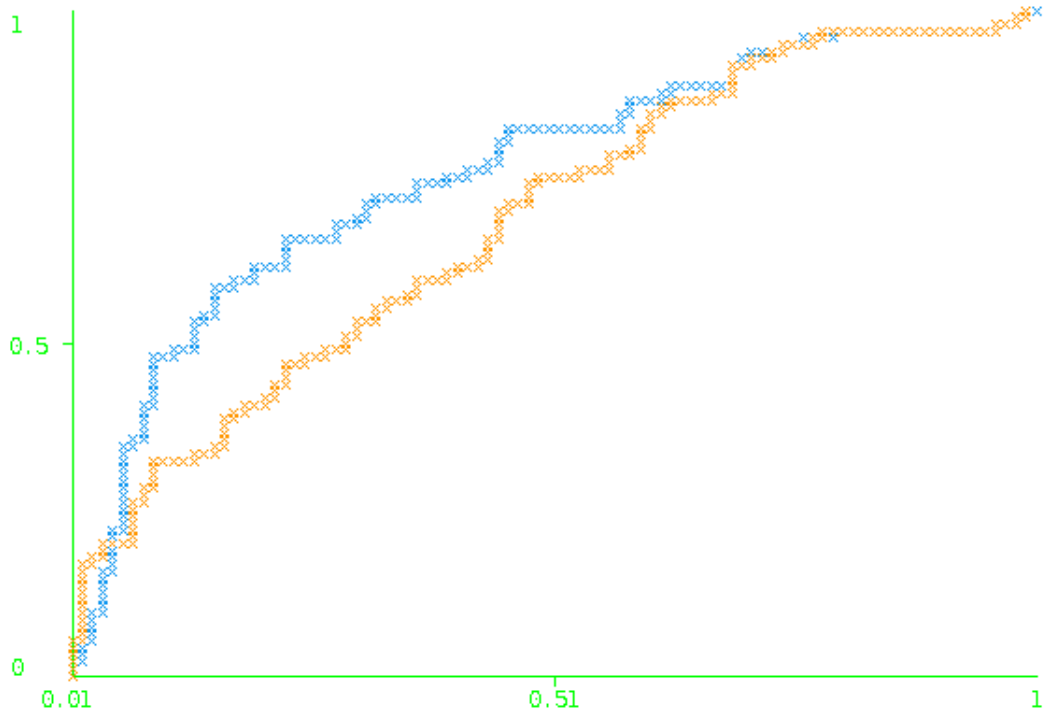
Tabulka 6.2: Porovnání křížové validace datasetů. Číselné hodnoty ukazují plochu pod ROC křivkou. Tučně označené hodnoty jsou o více jak 10% vyšší. A je původní dataset bez nových atributů, B obsahuje atributy IN_CAT_POCKET a CATALYTIC a C má i zbylé počítané atributy.

6.2.2 WEKA KnowledgeFlow

KnowledgeFlow je grafické rozhraní aplikace WEKA. Poskytuje možnost naplánovat testování pomocí orientovaného grafu, kde jsou jednotlivé uzly kroky experimentu, které mají každý své vstupy a výstupy, jimiž jsou propojeny. Pro lepší porovnání výsledků s kapsami a bez nich byly naplánovány dva experimenty, jejichž cílem bylo vytvořit ROC křivky. Z tohoto testu byl vyřazen dataset C, protože podle předchozích testů vykazoval stejné nebo horší výsledky jako B a neposkytoval tedy atributy, které by odhad zlepšovaly. Experiment je zobrazený na ilustraci 6.1 a výsledný graf, zobrazující obě ROC křivky je na ilustraci 6.2.



Obrázek 6.1: WEKA KnowledgeFlow diagram pro generování ROC křivky. ArffLoader nahrává vstupní dataset, ClassAssigner vybírá třídu, která bude sloužit ke klasifikaci (da, nt) a ClassValuePicker vybere, která hodnota z této třídy bude označena jako TRUE pro účely ROC křivky. CrossValidationFoldMaker vytváří jednotlivé skupiny pro křížovou validaci a předává jak trénovací i testovací množinu dat NaiveBayes modulu, který se na trénovací sadě naučí a na testovací sadě se otestuje. ClassifierPerformanceEvaluator z výsledků křížové validace sestaví data pro graf ROC křivky a ModelPerformanceChart graf zobrazí.



Obrázek 6.2: ROC křivky datasetů. Jde o spojené grafy pro oba datasety. Oranžová křivka je původní dataset, modrá je dataset nový. Čím více se body křivky blíží bodu (0, 1), tím přesnější výsledky metoda na datasetu poskytuje.

Kapitola 7

Závěr

Cílem této práce bylo najít atributy proteinů, které mají potenciál zlepšit odhad vlivu mutace na funkci proteinu. Následně měly být vytvořeny algoritmy, které tyto atributy získávají. Na doporučení vedoucího práce jsem analyzoval metody, které jsou používané pro vyhledávání kapes ve struktuře proteinů. Významným atributem mělo být zjištění, zda mutované residuum leží v kapse důležité pro funkci proteinu. Do aplikace Caver byl poté implementován modul na hledání a měření kapes a Caver byl následně připojen jako knihovna do frameworku Deleterious. Pomocí Deleterious byly nové atributy napočítány pro sadu lidských mutací, o nichž je známo, zda mají negativní vliv na lidské zdraví. Použitím strojového učení bylo ukázáno, že informace o přítomnosti residua v kapse aktivního místa a o katalytických residuích vedou ke kvalitnější předpovědi vlivu mutace proteinu na lidské zdraví. Nejlepších výsledků dosahoval Naivní Bayesův klasifikátor.

Modul pro výpočet kapes je rozšířením aplikace Caver, která byla doposud zaměřena na vyhledávání tunelů ve struktuře proteinu. Nyní Caver umožňuje zobrazení nalezených kapes v programu PyMOL a poskytuje informace o objemu kapes a o atomech, které tvoří stěny těchto kapes. Tyto údaje jsou důležité pro studium vztahu struktury a funkce proteinů a pro návrh mutací v proteinovém inženýrství.

Další vývoj programu Caver by mohl vést k podrobnější analýze struktury proteinu. Přesný objem kapes by bylo možné získat díky vylepšení algoritmů výpočetní geometrie, které se zatím používají pouze k přibližnému výpočtu objemu. Framework Deleterious má velký prostor k vývoji. Možná by bylo dobré zavést paralelizaci výpočtu nezávislých proteinů pro zrychlení výpočtu na procesorech s více jádry. Hlavní přínos však bude mít přidávání nových modulů. Není jisté, nakolik bude každý další atribut prospěšný pro účely odhadu. I odhalení neužitečných atributů je však prospěšné, protože vede k bližšímu pochopení vztahů mezi atributy popisujícími mutaci a funkcí proteinů.

Literatura

[1] CASTp background. http://sts.bioengr.uic.edu/castp/background.php, Květen 2010. URL http://sts.bioengr.uic.edu/castp/background.php

[2] Genetika, DNA & RNA. 2010, [Online; cit. 2009-12-20]. URL http://genetika.wz.cz/dnarna.htm

[3] Alberts; Bray; Johnson; aj.: *Základy buněčné biologie – Úvod do molekulární biologie buňky*. Espero Publishing s.r.o., druhé vydání, 2005, ISBN 80-902906-2-0.

[4] Binkowski, T. A.; Naghibzadeh, S.; Liang, J.: Computed Atlas of Surface Topography of proteins. *Nucleic Acids Research*, ročník 31, č. 13 3352-3355, 2003.

[5] Breiman, L.: Random Forests. *Machine Learning*, ročník 45, č. 1, 2001: s. 5–32.

[6] Bromberg, Y.; Rost, B.: Predict effect of non-synonymous polymorphisms on function. *Nucleic Acids Research*, 2007. URL http://nar.oxfordjournals.org/cgi/content/full/gkm238v1

[7] Edelsbrunner, H.; Facello, M.; Liang, J.: On the definition and the construction of pockets in macromolecules. *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing*, 1996: s. 272–287, ISSN 1793-5091.

[8] Edelsbrunner, H.; Fu, P.: Measuring space filling diagrams and voids. Technická Zpráva UIUC-BI-MB-94-01, Beckman Institute, University of Illinois, Urbana-Champaign, 1994.

[9] Edelsbrunner, H.; Mücke, E. P.: Three-dimensional alpha shapes. *ACM Trans. Graph.*, ročník 13, č. 1, 1994: s. 43–72.

[10] Hall, M.; Frank, E.; Holmes, G.; aj.: The WEKA data mining software: an update. *SIGKDD Explorations Newsletter*, ročník 11, č. 1, 2009: s. 10–18.

[11] John, G. H.; Langley, P.: Estimating Continuous Distributions in Bayesian Classifiers. In *Eleventh Conference on Uncertainty in Artificial Intelligence*, San Mateo: Morgan Kaufmann, 1995, s. 338–345.

[12] Lee, D. T.; Schachter, B. J.: Two algorithms for constructing a Delaunay triangulation. *International Journal of Parallel Programming*, ročník 9, č. 3, Červen 1980: s. 219–242.

[13] Lesk, A. M.: *Introduction to Bioinformatics*. Oxford University Press, třetí vydání, 2008, ISBN 978-0-19-920804-3.

[14] Matějíček, J.: *Výpočet atributů pro předpověď důsledku mutace na funkci proteinu.* Diplomová práce, FIT VUT v Brně, 2009.

[15] Medek, P.; Beneš, P.; Sochor, J.: Computation of tunnels in protein molecules using Delaunay triangulation. *Journal of WSCG*, 2007: s. 107–114.

[16] Mitchell, T.: *Machine Learning.* McGraw Hill Higher Education, první vydání, Říjen 1997, ISBN 0071154671.

[17] Pauline, C. N.; Henikoff, S.: Predicting amino acid changes that affect protein function. *Nucleic Acids Research*, ročník 31, č. 13 3812-3814, 2003.

[18] Pavelka, A.; Chovancova, E.; Damborsky, J.: HotSpot Wizard: a web server for identification of hot spots in protein engineering. *Nucleic Acids Research*, ročník 37, č. suppl_2, Červenec 2009: s. W376–383.

[19] Petsko, G. A.; Ringe, D.: *Protein structure and function.* New Science Press, 2004, ISBN 1405119225, 9781405119221, 220 s.

[20] Platt, J.; Schoelkopf, B.; Burges, C.; aj.: Machines using Sequential Minimal Optimization. In *Advances in Kernel Methods - Support Vector Learning*, MIT Press, 1998.

[21] Salzberg, S. L.: C4.5: Programs for Machine Learning by J. Ross Quinlan. Morgan Kaufmann Publishers, Inc., 1993. *Machine Learning*, ročník 16, č. 3, 1994: s. 235–240.

[22] Schymkowitz, J.; Borg, J.; Stricher, F.; aj.: The FoldX web server: an online force field. *Nucleic Acids Res*, 2005, [Online].
URL http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1160148/

[23] University of California Museum of Paleontology: "DNA and Mutations" – Understanding Evolution. Prosinec 2009, [Online].
URL http://evolution.berkeley.edu/evolibrary/article/mutations_01

[24] Watson; Crick: Molecular structure of nucleic acids; a structure for deoxyribose nucleic acid. *Nature*, 1953, [online].
URL http://www.nature.com/nature/dna50/watsoncrick.pdf

[25] Weisel, M.; Proschak, E.; Schneider, G.: PocketPicker: analysis of ligand binding-sites with shape descriptors. *Chemistry Central Journal*, ročník 1, č. 1, 2007: str. 7, ISSN 1752-153X.
URL http://journal.chemistrycentral.com/content/1/1/7

[26] Wikipedia: Wikipedia, The Free Encyclopedia. 2010, [Online].
URL http://en.wikipedia.org/

[27] Zhang, Y.; Rajapakse, J. C.: *Machine Learning in Bioinformatics.* John Wiley and Sons, Listopad 2008, ISBN 0470116625, 9780470116623, 476 s.

Příloha A

Konfigurace a spuštění

Příloha popisuje požadavky na systém, pokyny ke konfiguraci a spuštění a poznámky k běhu aplikace Caver a frameworku Deleterious.

A.1 Konfigurace

Aplikace pro své spuštění potřebují nainstalovanou javu verze 1.6 a vyšší. Obě aplikace jsou NetBeans projekty. Projekt Caveru je nazván AnBeKoM.

A.2 Spuštění

Ve složce, která obsahuje přeložený soubor .jar a složku lib s potřebnými knihovnami použít následující dávky:

A.2.1 Caver

```
java -jar AnBeKoM.jar
```

A.2.2 Deleterious

```
java -jar Deleterious.jar
```

Na výstup bude vytištěna nápověda, ze které lze vyčíst parametry aplikace. Lze také použít dávkové soubory demo.bat nebo demo.sh (podle systému) na spuštění programu s demonstračními vstupy.

NetBeans projekty se dají v NetBeans IDE otevřít a spustit přímo z přeložených .class souborů.

A.3 Dokumentace

Obě aplikace mají ve složce PROJEKT/dist/javadoc vygenerovanou programovou dokumentaci javadoc.

A.4 Upozornění

Caver při zpracování velkých molekul zabere až 1GB, Deleterious se blíží 1.5GB. Obě hodnoty jsou pouze orientační - více se při testování nezabralo. Pokud selže výpočet na nedostatku paměti, je možné virtuálnímu stroji Javy povolit více při spuštění s parametrem -Xmx, například -Xmx1500M povolí až 1500 MB paměti.

Deleterious při prvním běhu nad kompletním AUTO–MUTE datasetem běží zhruba 2.5 hodiny.

Příloha B

Obsah CD

Adresářová struktura obsahu přiloženého CD:

```
/
├── aplikace
│   ├── Caver ..... Spustitelná aplikace
│   ├── Deleterious ..... Spustitelná aplikace
│   └── zdrojovy_kod
│       ├── AnBeKoM ..... NetBeans projekt aplikace Caver
│       │   └── dist
│       │       └── javadoc ..... Programová dokumentace Caveru
│       └── Deleterious ..... NetBeans projekt aplikace Deleterious
│           └── dist
│               └── javadoc ..... Programová dokumentace Deleterious
├── technicka_zprava ..... PDF sobor s technickou zprávou
│   └── tex ..... LaTeX zdroje technické zprávy
└── vysledky
    ├── dataset_A.arff ..... Minimální dataset bez nových atributů
    ├── dataset_B.arff ..... Atributy IN_CAT_POCKET a CATALYTIC
    ├── dataset_C.arff ..... Všechny nové atributy
    ├── experiment_spravnost ..... Výstup experimentu - správnost
    └── experiment_roc ..... Výstup experimentu - plocha pod ROC
```