VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

SHLUKOVÁNÍ BIOLOGICKÝCH SEKVENCÍ

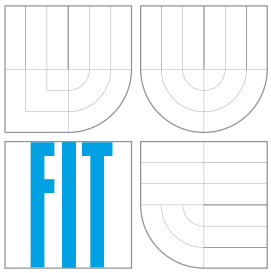
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. RADIM KUBIŠ

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

SHLUKOVÁNÍ BIOLOGICKÝCH SEKVENCÍ

CLUSTERING OF BIOLOGICAL SEQUENCES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. RADIM KUBIŠ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. TOMÁŠ MARTÍNEK, Ph.D.

BRNO 2015

Abstrakt

Jedním z hlavních důvodů, proč se proteinové sekvence shlukují, je predikce jejich struktury, funkce a evoluce. Mnoho současných nástrojů má nevýhodu ve velké výpočetní náročnosti, protože zarovnává každou sekvenci s každou. Pokud některý nástroj pracuje výrazně rychleji, nedosahuje oproti ostatním takové přesnosti. Další z nevýhod je zpracování na vyšších mírách podobnosti, ale homologní proteiny si mohou být podobné i méně. Proces shlukování také bývá ukončen při dosažení určité podmínky, která však nezohledňuje dostatečnou kvalitu shluků. Diplomová práce se zabývá návrhem a implementací nového nástroje pro shlukování proteinových sekvencí. Nový nástroj by měl být výpočetně nenáročný, se zachováním požadované přesnosti, a produkovat kvalitnější shluky. Práce dále popisuje testování navrženého nástroje, zhodnocení dosažených výsledků a možnosti dalšího rozvoje.

Abstract

One of the main reasons for protein clustering is prediction of structure, function and evolution. Many of current tools have disadvantage of high computational complexity due to all-to-all sequence alignment. If any tool works faster, it does not reach accuracy as other tools. Further disadvantage is processing on higher rate of similarity but homologous proteins can be similar with less identity. The process of clustering often ends when reach the condition which does not reflect sufficient quality of clusters. Master's thesis describes the design and implementation of new tool for clustering of protein sequences. New tool should not be computationally demanding but it should preserve required accuracy and produce better clusters. The thesis also describes testing of designed tool, evaluation of results and possibilities of its further development.

Klíčová slova

Shlukování, biologická sekvence, protein, porovnání, podobnost, hierarchie, bioinformatika

Keywords

Clustering, biological sequence, protein, comparison, similarity, hierarchy, bioinformatics

Citace

Radim Kubiš: Shlukování biologických sekvencí, diplomová práce, Brno, FIT VUT v Brně, 2015

Shlukování biologických sekvencí

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Tomáše Martínka, Ph.D.

.....
Radim Kubiš
27. května 2015

Poděkování

Rád bych poděkoval panu Ing. Tomáši Martínkovi, Ph.D., vedoucímu diplomové práce, za odbornou pomoc, ochotu, trpělivost a čas, který mi při tvorbě práce věnoval.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Sekvenční analýza hraje zásadní roli ve výpočetní biologii. Velikosti proteinových databází rychle narůstají díky výstupům z rozsáhlých genomových projektů a objevujícímu se poli metagenomiky. To má za následek stále větší obtížnost a časovou náročnost správy těchto databází a spouštění rutinních vyhledávání v nich. Současně je mnoho nových sekvencí podobných nebo téměř identických s již známými proteiny, mohou být sestřihanými variantami, proteiny příbuzných druhů nebo jednonukleotidovými polymorfismy. Tato informace může být stěžejní pro mnoho účelů, ale nemusí být relevantní v jiných. Účinnou cestou pro řešení tohoto problému je shlukování proteinových sekvencí do skupin a následné použití pouze jediné reprezentativní sekvence nebo konsenzu každé skupiny. Výsledné shluky mohou být buď oddělené, nebo tvořit hierarchickou strukturu. Shlukování výrazně šetří čas při databázovém hledání a zjednodušuje následné uspořádání získaných výsledků.

Detekce homologních proteinových sekvencí je také základem pro predikci struktury, funkce a evoluce dosud neurčených proteinů. Proteiny ve stejných rodinách/podrodinách mají podobnou funkci (např. stejnou enzymatickou aktivitu), ale s jinými parametry (např. rychlost reakce). Současné nástroje vykazují určité nepřesnosti a nedostatky. Nový nástroj pro shlukování proteinových sekvencí, který bude výsledkem této diplomové práce, by měl dosahovat lepších výsledků než dostupné nástroje. Jeho hlavním principem je převod sekvencí na vícedimenzionální vektory, jejichž jednotlivé souřadnice představují podobnosti s ostatními sekvencemi. Podobnost počítá lokálním zarovnáním. Tyto vektory jsou dále shlukovány na základě hustoty.

Kapitola 2 popisuje vybrané existující principy jak standardního, tak hierarchického shlukování, které dávají dobré výsledky a jsou v bioinformatickém oboru hojně využívány. Dále popisuje algoritmy shlukování založené na hustotě. Z těchto principů bude vybrán a případně upraven jeden, nebo kombinace více principů pro implementaci nového nástroje shlukujícího proteinové sekvence s požadovanými vlastnostmi. Výčet proteinových databází, na kterých se běžně testují nově vytvářené shlukovací nástroje, obsahuje kapitola 3. Nový nástroj je také na nejvhodnější z nich testován. Nedostatky současných algoritmů, včetně nástinu jejich možných řešení, jsou podrobněji rozebrány v 4. kapitole. Návrhová kapitola 5 popisuje způsob řešení jednotlivých částí nového nástroje vybranými algoritmy. V kapitole 6 je popis implementace navrženého nástroje a pomocných skriptů využívaných při jeho vývoji. Metodiku testování a porovnání nástrojů popisuje kapitola 7. Ta prezentuje i dosažené výsledky. Závěrečná kapitola 8 posuzuje výsledek práce a obsahuje návrh jejího možného budoucího vývoje.

Kapitola 2

Existující algoritmy pro shlukování

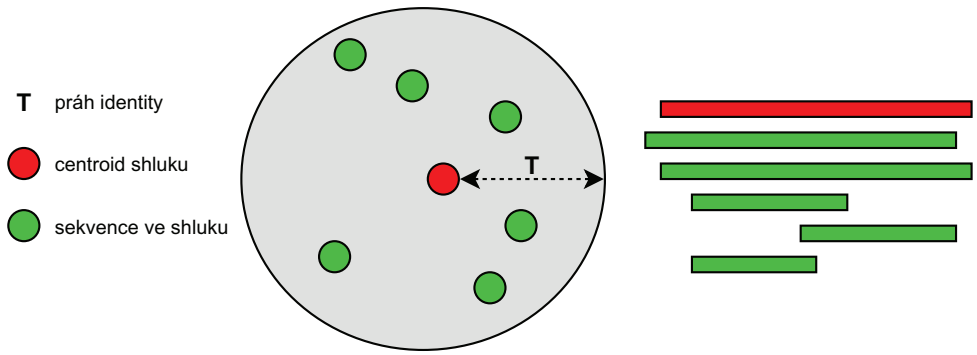
Kapitola popisuje populární a běžné využívané shlukovací algoritmy jak pro biologické sekvence, tak pro použití v mnoha jiných vědeckých disciplinách, které by bylo možné implementovat do nového nástroje.

2.1 Standardní algoritmy

Algoritmy produkující výsledky ve formě oddělených a exkluzivních shluků lze označit jako standardní. Zde jsou uvedeny jednoduché i složitější přístupy pro standardní shlukování.

2.1.1 UCLUST

Algoritmus UCLUST rozděluje sadu sekvencí do shluků. Každý shluk je definován jedinou sekvencí, které se říká centroid nebo reprezentativní sekvence. Všechny sekvence v jednom shluku musí mít podobnost s centroidem vyšší, než je zadaný práh identity T, který si lze představit jako poloměr shluku (obrázek 2.1).



Obrázek 2.1: Shlukování sekvencí algoritmem UCLUST [3].

UCLUST je efektivní pro identitu ∼50 % a více u proteinů a ∼75 % a více u nukleotidů. Efektivita této metody na nižších identitách je diskutabilní, protože se snižuje kvalita zarovnání a nelze z něj spolehlivě určit homologii.

Výstupem jsou shluky splňující kritéria:

1. Všechny centroidy mají vzájemnou podobnost menší než je práh identity T.

2. Všechny sekvence v jednom shluku mají podobnost větší nebo rovnu prahu identity T vzhledem k centroidu.

UCLUST je v základu heuristický a nelze zaručit dodržení 1. kritéria, ale v praxi jsou dva různé centroidy s podobností větší nebo rovnou prahu identity T vzácností. Obecně výše uvedená kritéria splní mnoho různých shlukování. Některá sekvence může být identická, dle nastaveného prahu T, s více centroidy. V ideálním případě bude přiřazena k nejbližšímu možnému centroidu, ale pokud jich existuje více se stejnou podobností, je přiřazení nejednoznačné a musí být provedeno náhodně.

Identity jednotlivých sekvencí jsou určovány na základě globálního zarovnání a jsou akceptovány i fragmenty.

Kroky algoritmu UCLUST:

1. Vezmi následující sekvenci ze vstupu.
2. Porovnej ji s existujícími centroidy na základě krátkých slov o délce k, tzv. k-mery.
3. Zarovnej tuto sekvenci s již existujícími centroidy, se kterými měla dostatečnou podobnost na základě k-merů.
4. Podle výsledku zarovnání:
   a) Pokud se sekvence podobá jednomu centroidu, zařaď ji do jeho shluku.
   b) Pokud se sekvence podobá více centroidům, zařaď ji k nejbližšímu, popř. vyber náhodně jeden z více se stejnou mírou podobnosti.
   c) Pokud se sekvence nepodobá žádnému centroidu, zaveď ji jako nový centroid.
5. Pokračuj bodem 1 pokud nebyl vyčerpán vstup, jinak skonči.

Tento algoritmus patří mezi tzv. „lačné" a jeho výsledek závisí na pořadí vstupních sekvencí. Pokud se následující sekvence shoduje s již existujícím centroidem, je zařazena do jeho shluku, jinak se stává centroidem shluku nového. Nejvhodnější sekvence pro centroidy by tedy měly být ve vstupní sadě co nejdříve (obrázek 2.2), záleží však na účelu použití. Jednou z možností může být seřazení podle délky sekvence [3, 4].

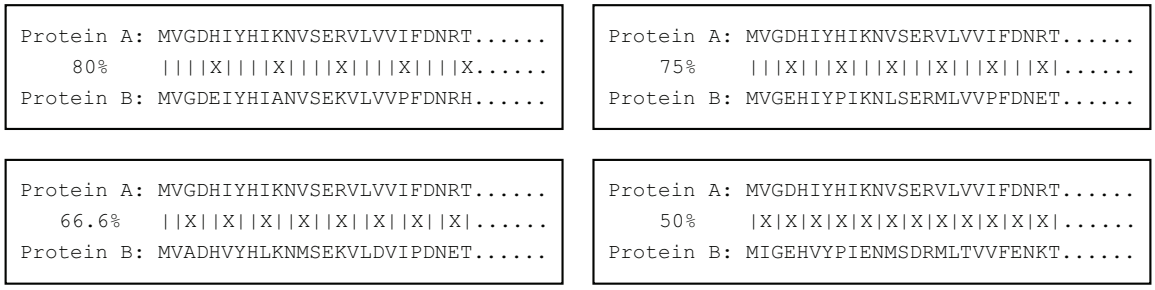
Obrázek 2.2: Výběr nejvhodnějších sekvencí pro centroidy. Centroidy jsou zobrazeny červeně. Nejdelší sekvence jako centroid (vlevo) a fragment jako centroid (vpravo). Fragmenty nejsou vhodné centroidy, protože se sekvence v jeho shluku nemusejí podobat v částech, které k němu nejsou zarovnány (žlutě) [3].

2.1.2 CD-HIT[1]

CD-HIT patří také mezi lačné algoritmy a před samotným shlukováním sekvencí je seřadí podle délky od nejdelší sekvence po nejkratší. Tvorba reprezentativních sekvencí jednotlivých shluků probíhá stejně jako u metody UCLUST. Algoritmus zaručuje, že všechny reprezentativní sekvence jsou vzájemně porovnány a všechny zařazené sekvence jsou porovnány s jejich reprezentanty. Přitom nezáleží na podobnosti dvou sekvencí přiřazených do stejného shluku, například dvě krátké sekvence mohou být zarovnány v jiných oblastech jedné dlouhé reprezentativní sekvence.

Vzhledem k výše uvedené shlukovací strategii je třeba provést vzájemné zarovnání mezi všemi reprezentanty a také zařazených sekvencí s jejich reprezentanty, což je velice časově náročné. Techniky pro urychlení shlukování využívané v tomto algoritmu jsou filtry krátkých slov a tabulka indexů. Filtry krátkých slov mají za účel rozhodnutí, zda identita mezi dvěmi sekvencemi je pod nebo nad prahem identity bez jejich vzájemného zarovnání, čímž se eliminuje zbytečné zarovnávání dvou nepodobných si sekvencí. Tabulka indexů slouží jako slovník, který obsahuje všechny možné kombinace aminokyselin dle délky slova použitého filtru. Například pro filtr používající pentapeptid je počet záznamů v tabulce 21^5 (20 aminokyselin + „X" na pěti pozicích).

Pokud mají mít dvě sekvence určitou podobnost, je nezbytné, aby obsahovaly jistý počet identických krátkých slov. Když nemají, není třeba jejich zarovnávání. Například pro podobnost nad 90 % musí sdílet nejméně jeden identický dekapeptid. Teoreticky lze každý filtr použít pouze pro určité prahy podobnosti. Již zmíněný dekapeptid platí jen pro vyšší než 90% podobnost. Nejnižší práh pentapeptidu je 80 %, protože dvě sekvence mající podobnost 80 % nemusejí obsahovat žádný společný pentapeptid, například se mohou lišit na každé páté aminokyselině podél jejich zarovnání (obrázek 2.3). Bohužel snižováním délky slova filtru výrazně klesá i jeho efektivita. Efektivně lze použít pentapeptid až k 70% podobnosti, tetrapeptid pro 55–60 % a tripeptid na hranici 50 %.



Obrázek 2.3: Rovnoměrně rozložené neshody pro různé délky peptidů [17].

Dvě sekvence se budou jen vzácně lišit na každé páté pozici podél zarovnání, protože obvykle sdílejí více krátkých slov, než je teoreticky požadované číslo a mohou tedy být použita i pro nižší identity. Používání neadekvátních filtrů může způsobit, že výsledek shlukování nebude dost kvalitní. Pokud ovšem požadujeme rychlé shlukování, je třeba zvolit optimální bod, který dostatečně vyhoví zvolenému účelu [14, 15, 16].

Kroky algoritmu CD-HIT:

1. Seřaď vstupní sekvence od nejdelší po nejkratší.

[1]Cluster Database at High Identity with Tolerance, http://cd-hit.org/.

2. Vezmi následující sekvenci ze vstupu.
3. Proveď porovnání sekvence s centroidy pomocí filtru krátkých slov.
4. Zarovnej tuto sekvenci s již existujícími centroidy, které vyhověly filtru krátkých slov.
5. Podle výsledku zarovnání:
   a) Pokud se sekvence podobá jednomu centroidu, zařaď ji do jeho shluku.
   b) Pokud se sekvence podobá více centroidům, zařaď ji k nejbližšímu, popř. vyber náhodně jeden z více se stejnou mírou podobnosti.
   c) Pokud se sekvence nepodobá žádnému centroidu, zaveď ji jako nový centroid.
6. Pokračuj bodem 2 pokud nebyl vyčerpán vstup, jinak skonči.

2.1.3 kClust

Potřebu rychlého, senzitivního a přesného shlukovacího algoritmu sekvencí s podobností pod 30 % vyplňuje metoda kClust. Stejně jako UCLUST a CD-HIT je i kClust lačný algoritmus. Vstupní sadu sekvencí seřadí před zpracováním od nejdelší po nejkratší. Obdobným způsobem jako předchozí metody určuje i reprezentativní sekvence jednotlivých shluků. Algoritmus je rychlý a senzitivní díky použitému předfiltru, který nevyužívá náročné vzájemné zarovnávání všech dvojic sekvencí. Dalšího zrychlení dosahuje dynamickým programováním nad k-ticemi při samotném zarovnávání předfiltrovaných sekvencí.

Předfiltr nehledá počty přesně shodných k-tic ve dvou porovnávaných sekvencích, ale pracuje s podobností těchto k-tic. Takový přístup umožnil zvýšení citlivosti metody. Konkrétně v případě kClust-u jsou počítány podobnosti 6-tic (obrázek 2.4). Pokud součet skóre z matice podobnosti přesáhne jistý práh S_{min}, sekvence předfiltru vyhoví a jsou vzájemně zarovnány.

Sekvence, které prošly krokem předfiltru, se zarovnají rychlou heuristikou k-ticového dynamického programování (kDP). Tato heuristika hledá optimální lokální zarovnání podobných si čtveřic. Optimální zarovnání je takové, které má největší součet skóre včetně penalizace za výskyt mezer. Celkové optimální zarovnání dvou sekvencí se poté hodnotí maticí substitucí BLOSUM62 opět nad čtveřicemi.

Pravidla pro zařazení vstupní sekvence do shluku jsou:

1. Podobnost sekvencí dynamického programování nad čtveřicemi je větší, než minimální skóre v matici substitucí BLOSUM62 po sloupcích.
2. Zarovnání dosahuje E-hodnoty menší, než je zadaný práh (standardně 1e-3).
3. Zarovnání pokrývá alespoň 80 % residuí reprezentativní sekvence.

Tato pravidla zaručují, že shluky obsahují sekvence s téměř identickou doménovou kompozicí. Protože kClust sleduje rozmístění čtveřic a šestic, redukuje se šum způsobený korelací mezi sousedícími k-ticemi [8, 19, 22].

Jednotlivé kroky algoritmu:

1. Seřazení vstupních sekvencí podle délky od nejdelší po nejkratší.
2. Porovnání následující vstupní sekvence pomocí 6-ticového předfiltru s již existujícími reprezentativními sekvencemi shluků.

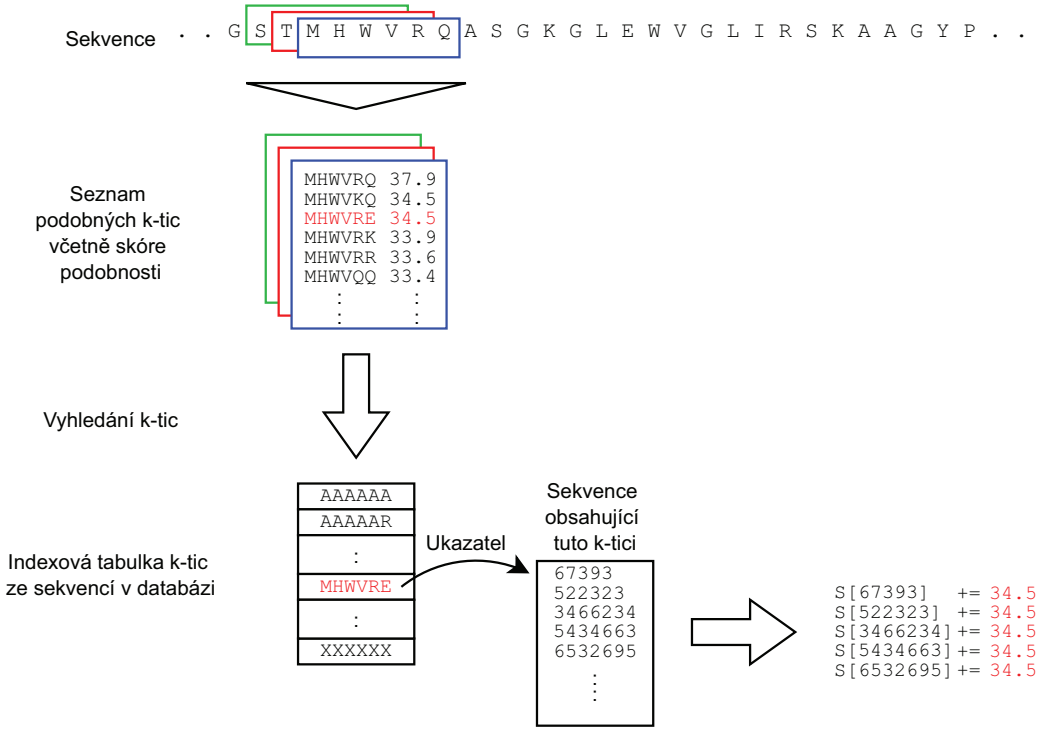
3. Zarovnání sekvence s již existujícími reprezentativními sekvencemi, které vyhověly předfiltru, algoritmem k-ticového dynamického programování čtveřic.

4. Podle výsledku zarovnání:

   a) Pokud se sekvence dostatečně shoduje s reprezentativní sekvencí některého shluku (podle výše uvedených tří pravidel), je do tohoto shluku zařazena.

   b) Pokud se sekvence neshoduje s žádnou reprezentativní sekvencí, zavede se jako reprezentativní sekvence nového shluku.

5. Pokračování od bodu 2 pokud nebyl vyčerpán vstup, jinak konec algoritmu shlukování.



Obrázek 2.4: Zjednodušený algoritmus skórování k-tic [19].

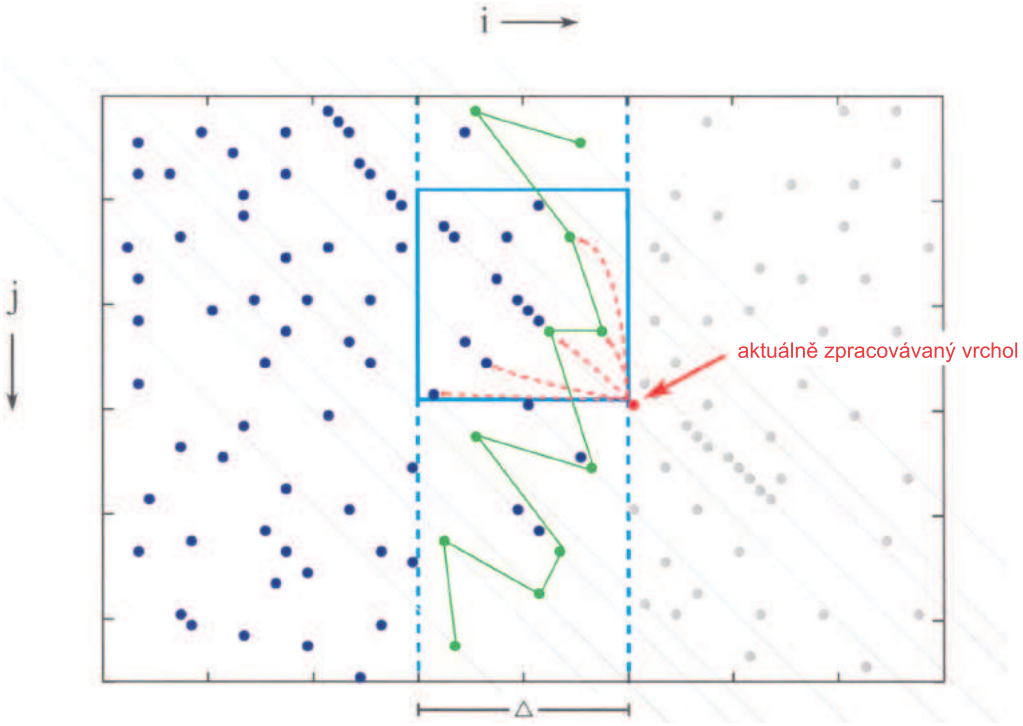
kDP

První krok algoritmu určuje optimální cestu na k-ticových shodách. Provádí dynamické programování pro maximalizaci skóre zarovnání k-tic včetně heuristických penalt za mezery.

Druhý krok využívá vytvořenou cestu v prvním kroku pro výpočet kompletního zarovnání páru sekvencí. Zarovnání residuí mezi shodami k-tic počítá se standardními afinními mezerovými penaltami a hodnocením substitucí aminokyselin.

Ve fázi dynamického programování je počítáno maximální zarovnání k-tice pro každý vrchol. Toto skóre koresponduje s nejlepším zarovnáním shody k-tic končících ve vrcholu. Algoritmus zpracovává vrcholy vzestupně dle pozice i v sekvenci x. Dochází k maximalizaci skóre pro každý vrchol (obrázek 2.5, červený bod) kontrolou pouze blízkých shodných k-tic uvnitř čtvercového okna s délkou strany Δ residuí (obrázek 2.5, modrý čtverec). Aby byl přístup ke shodám k-tic v Δ-okně rychlý, nedávno zpracované vrcholy uvnitř Δ-pásma

(obrázek 2.5, přerušovaný modrý obdélník) jsou udržovány. Nejbližší vrcholy k přední linii Δ-pásma na různých diagonálách jsou označovány jako bázové body (obrázek 2.5, zelené body). K-ticový graf k nim poskytuje přístup dle pořadí jejich diagonál (obrázek 2.5, zelené čáry). Navíc, každý bázový bod má hranu ke koncovým vrcholům na stejné diagonále. Červené přerušované čáry (obrázek 2.5) indikují vrcholy, které budou kontrolovány pro dynamické optimalizování skóre. Pokud existuje více k-ticových shod na stejné diagonále, je použit pouze nejbližší vrchol, což ničemu nevadí, protože více shodných k-tic na stejné diagonále často ukazuje na homologní regiony. Je tedy velmi nepravděpodobné, že by cesta zarovnání pokrývala pouze zlomek těchto shod. Z tohoto důvodu postačuje kontrola nejbližšího vrcholu k nalezení optimální cesty. V každém vrcholu musí být provedeny následující kroky pro výpočet skóre k-ticového zarovnání a pro uchování aktualizovaného k-ticového grafu.



Obrázek 2.5: k-ticové dynamické programování [19].

1. Nalezení vrcholů v Δ-okně pomocí bázových bodů aktuálního Δ-pásma. Bázové body mimo okno můžou mít vrcholy na jejich diagonále, které jsou uvnitř okna. Ty jsou přístupné z bázových bodů díky hranám, které ukazují na další vrchol stejné diagonály.

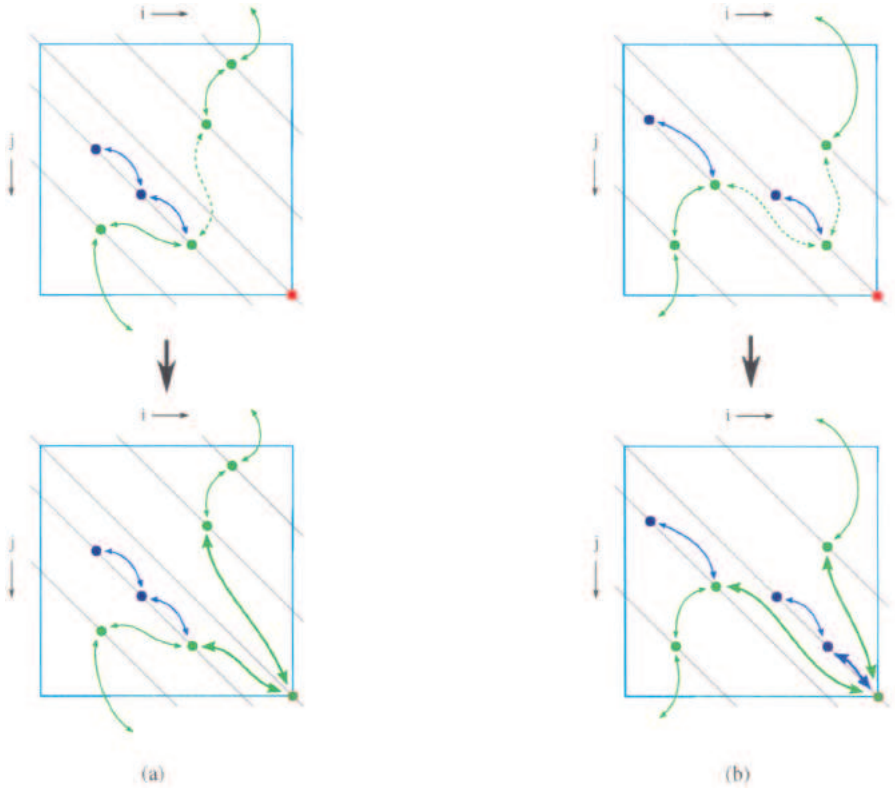
2. Výpočet skóre dynamického programování maximalizací přes cesty (červené bodované čáry) přes všechny možné předchůdce vrcholů nalezených v modrém Δ-okně. Uložení maximálního skóre dynamického programování a odpovídajícího zpátečního vrcholu v aktuálně zpracovávaném vrcholu (červený bod).

3. Aktualizace pozice přední linie Δ-pásma a přidání aktuálního vrcholu jako bázového do k-ticového grafu (obrázek 2.6).

4. Aktualizace zadní linie Δ-pásma osekáním vrcholů, které již nejsou uvnitř pásma a změna relevantních hran.

5. Posun na následující vrchol podle pozice i v sekvenci S_x a návrat na bod 1.

Existují dva různé způsoby pro zahrnutí aktuálně zpracovaného vrcholu do k-ticového grafu (krok 3). Buď je obsazena nová diagonála (obrázek 2.6a) nebo je existující bázový bod nahrazen (obrázek 2.6b). Vrcholy, které vypadnou na levé straně Δ-pásma (krok 4) jsou zpracované obdobně. Pokud byl vrchol bázovým bodem, přilehlé bázové body musí být znovu připojeny (obrázek 2.6, zelená hrana). Jinak, vrchol je třeba odpojit od jeho souseda na stejné diagonále [19].



Obrázek 2.6: Způsoby zahrnutí aktuálně zpracovávaného vrcholu do k-ticového grafu [19].

2.1.4 Tribe-MCL

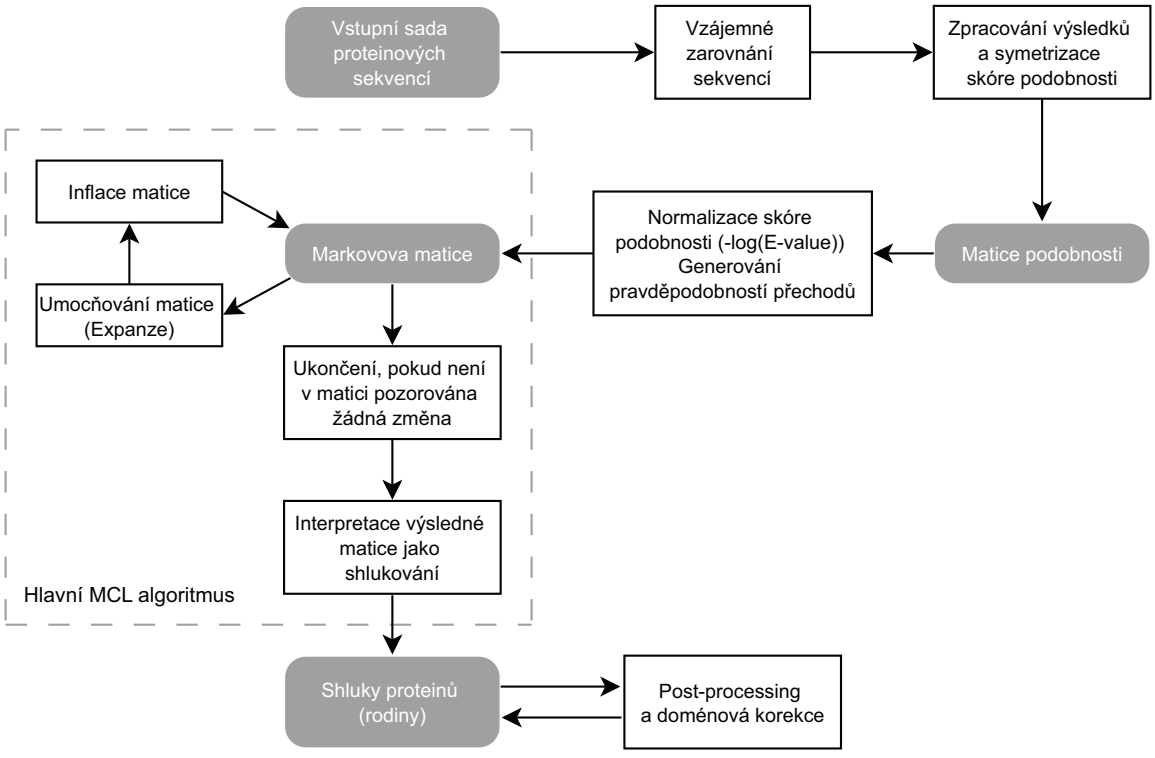
Metody pro shlukování proteinových sekvencí do rodin se obecně spoléhají na jejich míru podobnosti. Jedním problémem, kterému tyto metody čelí v mnoha proteinových rodinách, je detekce vícedoménových struktur. Proteiny obsahující vícedomény mohou metody zmást a výsledek obsahuje špatné zařazení do proteinové rodiny. Výskyt sdílených domén ve skupině proteinů nemusí nutně znamenat, že provádějí stejnou biochemickou funkci. V ideálním případě jsou tyto typy proteinů zařazeny do jedné rodiny pouze pokud vykazují vysokou podobnost stavby domén. Kromě těchto relativně velkých, nezávisle složených proteinových domén, bylo zjištěno, že menší, poměrně rozšířené proteinové moduly, zhoršují problém ještě více. Mnoho proteinů sdílí tzv. „promiskuitní domény" a je známo, že zastávají velmi roz-

dílné funkce. Proteiny přiřazené do rodiny čistě na základě těchto domén pravděpodobně nemají společnou evoluční historii s ostatními členy rodiny.

Mnoho eukaryotických proteinů obsahuje velké množství proteinových domén, z nichž každá musí být detekována a vyřešena efektivním shlukovacím algoritmem. Iterativní algoritmy automaticky detekující domény trpí nadměrným a nepředvídatelným množstvím dodatečných kroků pro porovnání sekvencí, které je činí nepraktickými. Řešením by byla spíše detekce proteinů s velmi podobnou stavbou domén, než detekce každé individuální domény. Předpokladem je, že proteiny s téměř identickou sadou domén můžou mít velmi podobnou biochemickou roli.

Tradiční metody pracují s podobnostním vztahem mezi párem sekvencí, ale teorie grafů umožňuje klasifikaci proteinů do rodin založenou na globálním zpracování všech vztahů v podobnostním prostoru současně. Tribe-MCL je efektivní a spolehlivá shlukovací metoda založená na Markovově shlukovacím (MCL) algoritmu, který byl vytvořen pro shlukování grafů využívaných v simulaci toků. Tento přístup je neuvěřitelně rychlý, velice přesný a netrpí mnoha výše uvedenými problémy.

Algoritmus vyžaduje předpočítanou matici podobnosti, která reprezentuje spojení grafu. Jednotlivé uzly reprezentují proteiny, hrany reprezentují podobnosti mezi proteiny. Váha je hranám přiřazována jako párový průměr $-log_{10}(E - hodnoty)$, výsledkem je symetrická matice. Váhy jsou dále transformovány na pravděpodobnosti mezi proteiny uvnitř grafu. Matice prochází iterativně expanzí a inflací, dokud neobsahuje velmi malou nebo žádnou změnu (ekvilibrium). Finální matice se pak interpretuje jako shlukování proteinů do rodin. Kroky algoritmu jsou popsány na obrázku 2.7.



Obrázek 2.7: Schéma algoritmu Tribe-MCL [5].

MCL algoritmus vyhledává shluky v grafu pomocí matematických procedur. Proces deterministicky vypočítává náhodné průchody grafem podobnosti a používá dva operátory

transformující jednu pravděpodobnost na druhou. Využívá k tomu jazyk stochastických matic (zvaných Markovovy matice), které zachycují matematický koncept náhodného procházení grafem. Střídá při tom operátory zvané expanze a inflace. Expanze se shoduje s umocňováním matice. Inflace odpovídá umocňování matice po složkách, následovanému změnou měřítka, čímž se matice stává opět stochastickou, tj. buňky matice odpovídají pravděpodobnostem. Sloupcová stochastická matice je matice s vlastností, že součet hodnot ve sloupci je roven 1. Každý sloupec odpovídá jednomu uzlu, každá buňka ve sloupci potom hraně s pravděpodobností k jinému uzlu. Parametr inflace MCL algoritmu je používán ke kontrole zrnitosti (těsnosti) jednotlivých shluků. Inflace zvyšuje pravděpodobnost uvnitř shluku a snižuje pravděpodobnost mezi shluky.

Tento algoritmus je rychlý, škálovatelný a má přirozený parametr zrnitosti. Je postaven na velmi odlišném paradigmatu než ostatní shlukovací metody [5].

2.1.5 Hobohm and Sander

Uwe Hobohm a Chris Sander vytvořili dva různé algoritmy pro výběr reprezentativních sekvencí ze vstupní datové sady, které jsou určeny především pro minimalizaci redundance a maximalizaci pokrytí redukované datové sady [11, 12].

Select until done („vyber dokud není hotovo")

Algoritmus probíhá současným zpracováváním tří seznamů proteinových identifikátorů:

- testovací seznam všech kandidátních proteinů (test list),
- seznam přeskočení (skip list),
- seznam výběru (select list).

Testovací seznam může být seřazen podle uživatelem definovaných kritérií, takže určité typy proteinů mají větší pravděpodobnost výběru. Seznam přeskočení obsahuje proteiny, které jsou podobné předchozím zpracovaným proteinům z testovacího seznamu a může také obsahovat předem nežádoucí proteiny. Seznam výběru (počátečně prázdný) obsahuje proteiny vybrané jako reprezentativní.

Postup jednotlivých kroků algoritmu:

1. Načti identifikátor jednoho proteinu z testovacího seznamu.
2. Zkontroluj, jestli je tento protein v seznamu přeskočení:
   (a) Pokud ano, opakuj bod 1.
   (b) Pokud ne, pokračuj na další bod.
3. Zkontroluj, jestli protein splňuje uživatelem specifikované požadavky (např. minimální délka sekvence):
   (a) Pokud ano, přidej protein do seznamu výběru.
   (b) Pokud ne, jdi na bod 1.
4. Proveď FASTA vyhledávání proteinu oproti zbývajícím sekvencím v testovacím seznamu.

5. Projdi výsledný FASTA soubor a přidej do seznamu přeskočení proteiny s větší podobností, než je nastavený práh.

6. Pokud ještě nebyl zpracován celý vstup, vrať se na bod 1, jinak skonči.

Remove until done („odstraň dokud není hotovo")

Tento algoritmus je výpočetně náročnější, protože vyžaduje kompletní matici párových vztahů mezi všemi proteiny v seznamu kandidátů. Cílem je odstranění, v co nejmenším počtu kroků, jednoho proteinu současně s jeho párovými vztahy, dokud není matice všech zbývajících proteinů bez jakýchkoliv vztahů. Tento globální optimalizační problém není triviální a je řešen procedurou lačného typu: odstranění proteinu s nejvyšším počtem párových vztahů, což směřuje k minimalizaci celkového počtu potřebných kroků pro odstranění všech párových vztahů v matici.

Matice vztahů je generována ze vzájemného zarovnání každé sekvence se všemi ostatními za pomoci dynamického zarovnávacího algoritmu Smith&Waterman. Po aplikaci zadaného prahu podobnosti obsahuje matice pouze jednobitovou informaci pro každý proteinový pár:

- 1, pokud si jsou proteiny podobné,
- 0 jinak.

V každé iteraci je odstraněn protein tak, že jsou všechny jeho bitové vztahy vůči ostatním nastaveny na 0. Pokud má nejvyšší počet párových vztahů více proteinů, je jeden z nich vybrán náhodně. Algoritmus končí, když jsou všechny bity párových vztahů vynulovány (všechny zbývající proteiny jsou vzájemně odlišné). Zbývající proteiny se stávají vybranými reprezentanty. V závěrečném průchodu všemi odstraněnými proteiny jsou obnoveny ty, které nemají žádného souseda mezi vybranými proteiny, ale tento krok jen zřídkakdy rozšíří skupinu reprezentantů.

Rozdíl mezi oběma metodami je graficky znázorněn na obrázku 2.8.

2.2 Hierarchické algoritmy

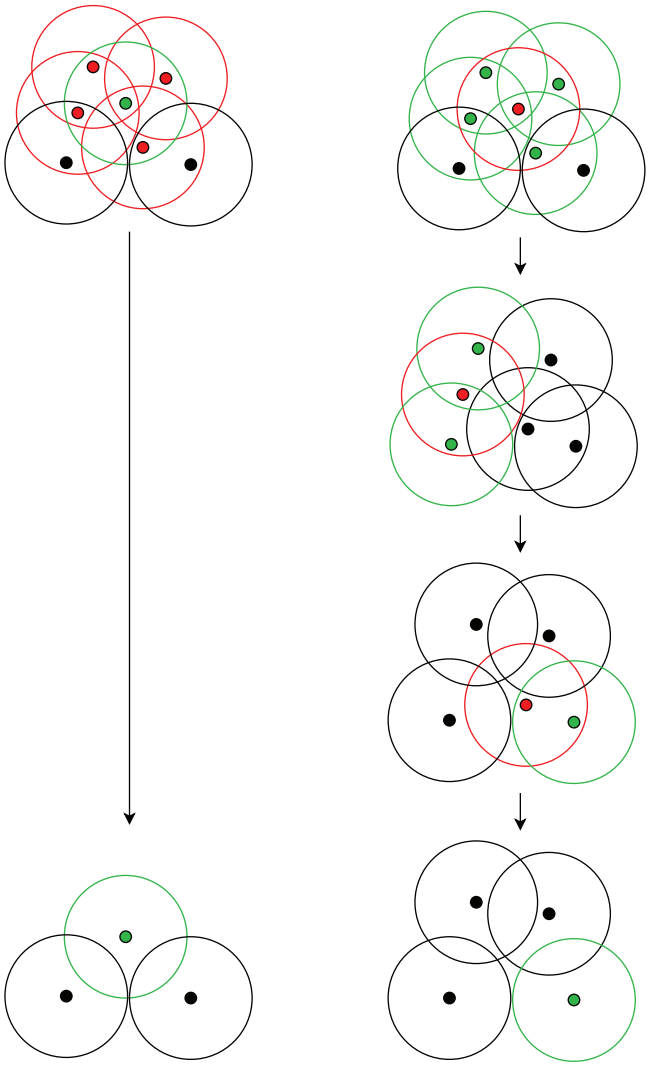
Výstupem těchto algoritmů je binární stromová hierarchie mezi vstupními daty. V každém kroku se vybere dvojice shluků k jejich sloučení. N listů odpovídá jednotlivým vstupním položkám. N-1 vnitřních uzlů (shluků) odpovídá seskupení hrubší zrnitosti na vyšším stupni ve stromu a nejvyšší úroveň představuje jeho kořen. Skóre sloučení představuje výšku dendrogramu.

Pro definování podobnosti párů napříč shluky jsou používány různé formulace. Metody s jednoduchou vazbou spojují vždy nejbližší dvojice. Pečlivé metody dokáží shlukovat celou vstupní sadu, čtou jednu hranu současně. Z tohoto důvodu lze implementace s jednoduchou vazbou škálovat na rozsáhlé datové sady. Jsou ale citlivé na odlehlé hodnoty, protože v jednotlivých krocích uvažují pouze minimální hranu.

2.2.1 UPGMA[2]

Pravděpodobně nejpopulárnější algoritmus pro hierarchické shlukování zejména ve výpočetní biologii. Vytváří kořenový strom podobností zdola nahoru. K určení podobnosti sekvencí

[2]Unweighted Pair Group Method using arithmetic Averages.



Obrázek 2.8: Odstranění proteinů v jedné iteraci algoritmu Select until done (vlevo) a během tří iterací algoritmu Remove until done (vpravo). Červené body představují sekvence k odstranění, červené kružnice jejich okolí. Zelenou barvou jsou reprezentovány sousedící body a jejich okolí. Černé sekvence znázorňují ostatní sekvence v blízkosti předchozích, ale nejsou algoritmy nijak dotčeny [12].

používá aritmetický průměr napříč všemi shluky, díky jehož stabilitě je velmi praktický, a je tedy robustnější než metody s jednoduchou vazbou.

Shlukování metodou UPGMA vyžaduje předpočítání zarovnání všech kombinací dvojic sekvencí. Hodnoty jsou uloženy do trojúhelníkové matice podobnosti. Spojují se vždy dvě sekvence nebo shluky s nejvyšší mírou identity. Po každém sloučení dochází ke zmenšení matice tak, že se dva řádky, resp. dva sloupce spojených sekvencí nahradí jedním řádkem, resp. jedním sloupcem. Dále je nutný přepočet spojením dotčených hodnot v matici s využitím aritmetického průměru.

Nevýhodou tohoto algoritmu je nutnost uchovávání celé matice podobnosti v paměti. Počet párových vztahů narůstá kvadraticky s počtem shlukovaných sekvencí a matice začíná být mimořádně velká již při středně rozsáhlém vstupu. Algoritmus se tak stává nepraktický

pro mnoho různých úloh [18, 21].

Kroky algoritmu UPGMA:

1. Vypočítej zarovnání všech kombinací dvojic sekvencí.
2. Vyber z tabulky podobnosti dvě sekvence S_1 a S_2 s nejvyšší mírou identity.
3. Dvojici spoj do jedné sekvence S_{1+2} a přepočítej podobnosti v tabulce dle vztahu 2.1

$$Identita\ S_{1+2,\,X} = \frac{Identita\ S_{1,\,X} + Identita\ S_{2,\,X}}{2}, \tag{2.1}$$

   kde X označuje sekvenci, ke které se vztahuje výpočet nové míry identity.
4. Opakuj od bodu 2, dokud nejsou spojeny všechny dvojice.

Příklad spojení tří sekvencí algoritmem UPGMA:

	S_1	S_2	S_3
S_1	–		
S_2	0,71	–	
S_3	0,57	0,63	–

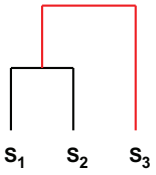
Tabulka 2.1: Výběr sekvencí S_1 a S_2.

	S_{1+2}	S_3
S_{1+2}	–	
S_3	0,60	–

Tabulka 2.2: Výběr sekvencí S_{1+2} a S_3.

$$Identita\ S_{1+2,\,3} = \frac{Identita\ S_{1,\,3} + Identita\ S_{2,\,3}}{2} = \frac{0,57 + 0,63}{2} = \frac{1,20}{2} = \underline{\underline{0,60}}$$

Obrázek 2.9: Spojení sekvencí S_1 a S_2.



Obrázek 2.10: Spojení sekvencí S_{1+2} a S_3.

2.2.2 Neighbour-joining

Neighbour-joining je shlukovací heuristická metoda, která generuje obecně nekořenové stromy. Stejně jako UPGMA pracuje ve směru zdola nahoru. Využívá se predevším pro tvorbu fylogenetických stromů. Jejím cílem je nalézt dva uzly ke spojení splňující kritéria:

1. Uzly jsou nejblíže k sobě.
2. Uzly jsou nejvzdáleněji od všech ostatních uzlů.

Vzdálenosti dvojice uzlů od sebe obsahuje předpočítaná matice podobnosti. Vzdálenost jednoho uzlu od všech ostatních musí být vypočtena podle vztahu 2.2

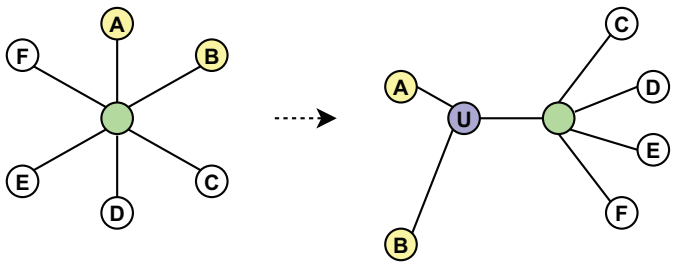
$$u(A) = \frac{1}{N-2} \sum_{i=1}^{N} D(A, X_i), \tag{2.2}$$

kde číslo N je počet všech analyzovaných uzlů a $D(A, X_i)$ je vzdálenost uzlu A od uzlu X_i z matice podobnosti.

Určit kořen stromu lze tak, že se do skupiny sekvencí vloží tzv. „outgroup", nebo-li sekvence, která je velmi vzdálená od skupiny ostatních analyzovaných sekvencí (ingroups). Taková sekvence bude připojena ke všem ostatním nejdelší hranou. Pokud nelze jednoduše outgroup sekvenci zvolit, vybere se nejdelší hrana stromu a kořen je vložen do středu této hrany [21].

Kroky metody Neighbour-joining:

1. Vypočítej matici vzdáleností.
2. Vytvoř hvězdicový strom s jedním společným středovým uzlem a listovými uzly odpovídajícími vstupním sekvencím.
3. Najdi dva uzly A a B, které splňují výše uvedená kritéria.
4. Spoj tyto uzly do jednoho uzlu U (obrázek 2.11).
5. Vypočítej délky nově vzniklých hran.
6. Vypočítej vzdálenost uzlu U od všech ostatních uzlů.
7. Opakuj od bodu 3, dokud nezůstane pouze jediný uzel.



Obrázek 2.11: Počáteční strom (vlevo) a strom po prvním spojení (vpravo) metodou Neighbour-joining.

2.2.3 mBed

Klíčem k vytvoření pokrokového přístupu ke shlukování sekvencí je použitá metoda generující řídící strom. Obvykle se předpokládá zarovnání každé sekvence s každou při složitosti $O(N^2)$. Algoritmus mBed pracuje se složitostí $O(N \log N)$ a produkuje stejně přesné stromy jako konvenční metody.

Sekvence vstupní datové sady jsou převedeny na n-elementové vektory v n-rozměrném prostoru, kde n je úměrné k $log\, N$. Každý element vektoru představuje jednoduše vzdálenost k jedné z n referenčních sekvencí. Takové vektory pak mohou být shlukovány extrémně rychle standardními metodami jako k-means nebo UPGMA.

Algoritmus mBed se skládá ze tří kroků:

1. Počáteční výběr referenčních sekvencí nazývaných „semínka".

   Nejdříve je vybrán vzorek t sekvencí ze vstupní datové sady X, kde $t = (log_2 N)^2$. Tato počáteční množina se značí jako R a je zvolena z podle délky seřazené vstupní sady X s konstantním krokem.

   Vybraná semínka jsou vzájemně porovnána. Pokud si jsou některá velice podobná (s určitou prahovou vzdáleností), kratší z nich se považuje za nadbytečné a je vyřazeno. Tento práh se standardně nastavuje na nulu, aby byla vyřazena pouze identická semínka.

2. Analýza potenciálních semínek.

   Množina referenčních sekvencí R teď může být použita k vektorizaci vstupních sekvencí. Alternativně lze každé semínko použít pro nalezení dalších semínek, které lépe charakterizují vstupní datovou sadu. Existují dva způsoby:

   (a) Heuristika používající středové objekty.

   Každé semínko je použito k nalezení poteciálních odlehlých hodnot. Nejdříve se identifikuje sekvence, která se nachází nejdále od semínka. Sekvence ležící nejdále od takovéto sekvence se stává novým semínkem.

   Pro každou sekvenci s v R:

   i. Nechť l je sekvence v X s maximální vzdáleností $d(l, s)$.
   ii. Nechť m je sekvence v X s maximální vzdáleností $d(m, l)$.
   iii. m je nové semínko.

   (b) Heuristika používající středové skupiny.

   Heuristika středových skupin pracuje stejně jako heuristika středových objektů, jen místo sekvencí hledá jejich skupiny. Nejdříve najde sekvenci, která je nejdále od semínka. Potom iterativně hledá sekvenci nalézající se nejdále od skupiny již nalezených sekvencí.

   Pro každou sekvenci s v R:

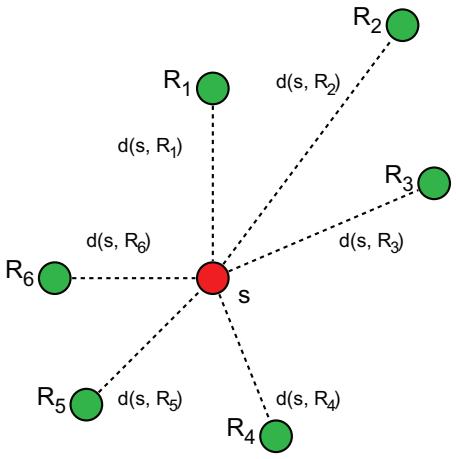
   i. Nechť l je sekvence v X s maximální vzdáleností $d(l, s)$.
   ii. Nechť m je sekvence v X s maximální vzdáleností $d(m, s) + d(m, l)$.
   iii. Nechť n je sekvence v X s maximální $d(n, s) + d(n, l) + d(n, m) + \ldots$

   Smyčka je ukončena, pokud se identifikuje stejná sekvence více než jednou, nebo pokud skupina dosáhne maximální nastavené velikosti. Každý z členů skupiny je pak novým semínkem. Stejně jako v kroku 1, než se některá sekvence stane novým semínkem, je porovnána s již existujícími semínky. Pokud je některému velice podobná, zamítne se jako nové semínko.

3. Vektorizace vstupních sekvencí.

   Jakmile je vybrána množina semínek R, všechny sekvence vstupní datové sady jsou asociovány s jim odpovídajícím t-dimenzionálním vektorem. To se provede jednoduše jako výpočet vzdálenosti sekvence od každého z t semínek. Získané hodnoty jsou souřadnice vektoru.

   Pro každou sekvenci s v X: $F(s) = [d(s, R_1), d(s, R_2), \ldots, d(s, R_t)]$ (obrázek 2.12).

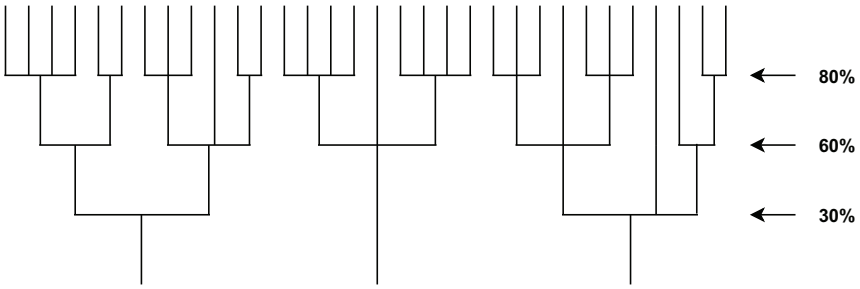


Obrázek 2.12: Vektorizace vstupní sekvence s od t semínek metodou mBed.

Proces převodu biologických sekvencí na jim odpovídající vektory probíhá takovým způsobem, že vzdálenosti mezi těmito vektory se blíží vzájemným vzdálenostem mezi sekvencemi. Vektory v tomto tvaru jsou řazeny rychleji, než vzdálenosti sekvencí, což umožňuje rychlejší generování matice vzdáleností ze vstupní datové sady [2].

2.2.4 Hierarchický CD-HIT

Standardní algoritmus CD-HIT může být využitý i pro proces hierarchického shlukování. První běh proběhne standardně nad originální vstupní datovou sadou, kdy je nastaven vysoký práh podobnosti. V každém dalším běhu vstupují do shlukovacího procesu pouze reprezentativní sekvence jednotlivých shluků z předchozího běhu, ale práh podobnosti je nastaven na menší hodnotu (obrázek 2.13). Tento proces iterativně spojuje podobné sekvence a produkuje tím hierarchickou strukturu [13].



Obrázek 2.13: Tvorba hierarchie při využití algoritmu CD-HIT [17].

2.3 Shlukování na základě hustoty

Standardní algoritmy vytvářejí rozdělení databáze D s n objekty do k shluků. k je vstupním parametrem, tj. požaduje nějakou znalost oblasti, která není bohužel pro mnoho aplikací dostupná. Algoritmus typicky začíná s iniciálním rozdělením D a pak iterativně hledá optimální rozdělení. Každý shluk je reprezentován středním objektem shluku nebo jedním z objektů shluku ležícím blízko jeho středu. Standardní algoritmy tedy používají dva kroky.

V prvním určí k reprezentativních objektů. Ve druhém přidělují každý objekt k nejbližšímu reprezentativnímu objektu. Shluky nalezené těmito algoritmy jsou konvexní, což je velice omezující.

Hierarchické algoritmy vytvářejí hierarchickou dekompozici D. Dekompozice je reprezentována dendrogramem, stromem iterativně rozdělujícím D na menší podmnožiny, dokud každá podmnožina neobsahuje pouze jeden prvek. V takové hierarchii představuje každý uzel stromu jeden shluk z D. Dendrogram může být tvořen od listů ke kořenu (aglomerativně), resp. od kořene k listům (členěním) shluků v každém kroku. Oproti rozdělujícím algoritmům nepotřebují k jako vstup, ale potřebují mít definovánu ukončující podmínku slučování, resp. dělení.

Shlukovací algoritmy jsou atraktivní pro úlohu klasifikace, ale s jejich aplikací na rozsáhlé databáze rostou následující požadavky:

1. Minimální znalost dané oblasti ke stanovení vstupních parametrů, protože vhodné hodnoty nejsou často předem známé, pokud se jedná o rozsáhlé databáze.

2. Nalezení shluků libovolných tvarů, protože tvar shluků v databázích může být lineární, kulový atd.

3. Dobrá efektivita na rozsáhlých databázích, tj. na databázích podstatně větších, než jen pár tisíc objektů.

Při pohledu na ukázkové sady bodů znázorněných na obrázku 2.14, můžeme snadno a jednoznačně detekovat shluky bodů a šumové body nepatřící do žádného ze shluků. Hlavní důvod, proč rozeznáme shluky je ten, že uvnitř každého shluku je hustota bodů typicky značně vyšší, než mimo shluk. Kromě toho, hustota v oblastech šumu je menší, než hustota jakéhokoliv ze shluků.

Obrázek 2.14: Ukázkové sady bodů pro shlukování [6].

2.3.1 DBSCAN[3]

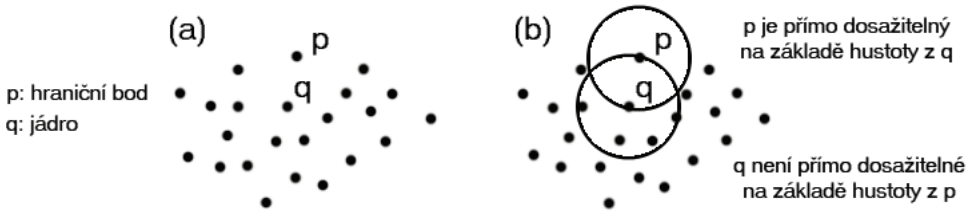
Známé algoritmy neposkytují řešení pro kombinaci výše uvedených požadavků. Algoritmus DBSCAN vyžaduje pouze jeden vstupní parametr a umožňuje uživateli určit jeho nejvhodnější hodnotu. Nalezne shluky libovolných tvarů a je efektivní i pro rozsáhlé databáze.

Klíčovou myšlenkou je, že pro každý bod shluku musí jeho okolí daného poloměru obsahovat alespoň minimum bodů, tj. hustota okolí musí překročit určitý práh. Tvar okolí

[3]Density Based Spatial Clustering of Applications with Noise.

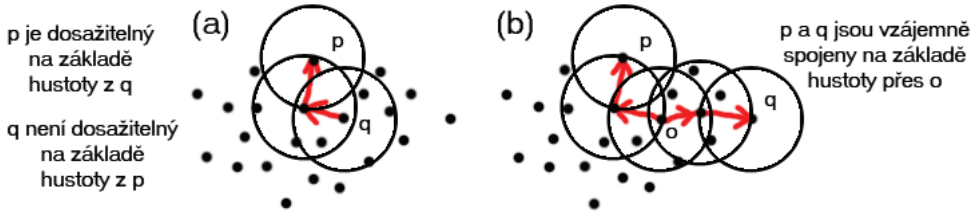
je určen vybranou vzdálenostní funkcí pro dva body P a Q, označovanou jako *dist(P, Q)*. Například pro vzdálenostní funkci Manhattan ve 2D prostoru je tvar okolí obdélníkový. Přístup v DBSCAN pracuje s jakoukoliv vzdálenostní funkcí, takže pro každou aplikaci je možné zvolit odpovídající funkci.

Naivní přístup by mohl vyžadovat pro každý bod ve shluku alespoň minimální počet (*MinPts*) bodů v ε-okolí bodu (A.1.1). Ovšem tento přístup selhává, protože existují dva druhy bodů ve shluku: body uvnitř shluku (jádra) a body na hranici shluku (hraniční body) (obrázek 2.15). Obecně ε-okolí hraničního bodu obsahuje výrazně méně bodů než ε-okolí jádra. Je tedy třeba nastavit minimální počet bodů na relativně nízkou hodnotu za účelem zahrnutí všech bodů patřících do stejného shluku. Tato hodnota ovšem nebude charakteristická pro příslušný shluk – zvláště za přítomnosti šumu. Proto je třeba, aby pro každý bod P ve shluku C existoval takový bod Q v C, že P je uvnitř ε-okolí Q a $N_\varepsilon(P)$ obsahuje alespoň*MinPts* bodů.



Obrázek 2.15: Jádra a hraniční body algoritmu DBSCAN [6].

Shluk (A.1.5) je definován jako množina bodů spojených na základě hustoty (A.1.4), který je maximální vzhledem k dosažitelnosti na základě hustoty (A.1.2, A.1.3) (obrázek 2.16). Šum (A.1.6) je definován relativně k dané množině shluků jednoduše, jako množina bodů v *D*, které nepatří do žádného ze shluků.



Obrázek 2.16: Dosažitelnost na základě hustoty a spojení na základě hustoty u metody DBSCAN [6].

Je třeba znát ideální parametry ε a *MinPts* každého shluku a alespoň jeden bod příslušného shluku. Pak lze získat všechny body, které jsou dosažitelné na základě hustoty z daného bodu. Neexistuje ovšem snadný způsob, jak získat tyto parametry pro všechny shluky předem. K tomuto účelu lze jednoduchou a efektivní heuristikou stanovit ε a *MinPts* nejřidšího shluku databáze. DBSCAN tak používá globální hodnotu těchto parametrů (stejné hodnoty pro všechny shluky). Parametry hustoty nejřidšího shluku jsou dobrými kandidáty, protože specifikují nejnižší hustotu, která není považována za šum.

Algoritmus

DBSCAN začne s libovolným bodem P a hledá všechny body dosažitelné na základě hustoty od P vzhledem k ε a *MinPts*. Pokud je P jádrem, dostaneme shluk vzhledem k ε a *MinPts* (Lemma A.1.2). Pokud je P hraniční bod, žádné body z něj nejsou dosažitelné na základě hustoty a DBSCAN pokračuje dalším bodem databáze.

Při používání globální hodnoty ε a *MinPts* může DBSCAN sloučit (dle definice A.1.5) dva shluky s rozdílnou hustotou do jednoho, pokud jsou blízko sebe. Nechť je *vzdálenost dvou množin bodů* S_1 a S_2 definována jako $dist(S_1, S_2) = min\{dist(P, Q) \mid P \in S_1, Q \in S_2\}$. Potom budou dvě množiny bodů s hustotou alespoň takovou, jakou má nejřidší shluk, od sebe odděleny pouze pokud vzdálenost mezi nimi je větší než ε. Pak může být nutné rekurzivní volání DBSCAN pro nalezené shluky s vyšší hodnotou *MinPts*. Toto není nevýhodou, protože rekuzivní aplikace DBSCAN z něj dělá elegantní a velmi efektivní algoritmus. Kromě toho, rekurzivní shlukování bodů shluku je nezbytné pouze za podmínek, které mohou být snadno detekovány.

Pseudokód algoritmu DBSCAN:

```
DBSCAN (SetOfPoints, Eps, MinPts)

  // SetOfPoints is UNCLASSIFIED
  ClusterId := nextId(NOISE);
  FOR i FROM 1 TO SetOfPoints.size DO
    Point := SetOfPoints.get(i);
    IF Point.ClId = UNCLASSIFIED THEN
      IF ExpandCluster(SetOfPoints, Point, ClusterId, Eps, MinPts) THEN
        ClusterId = nextId(ClusterId);
      END IF
    END IF
  END FOR
END; // DBSCAN
```

Nejdůležitější funkcí algoritmu DBSCAN je:

```
ExpandCluster(SetOfPoints, Point, ClId, Eps, MinPts) : Boolean;
  seeds := SetOfPoints.regionQuery(Point, Eps);
  IF seeds.size < MinPts THEN // no core point
    SetOfPoints.changeClId(Point, NOISE);
    RETURN False;
  ELSE  // all points in seeds are density-reachable from Point
    SetOfPoints.changeClIds(seeds, ClId);
    seeds.delete(Point);
    WHILE seeds <> Empty DO
      currentP := seeds.first();
      result := SetOfPoints.regionQuery(currentP, Eps);

      IF result.size >= MinPts THEN
        FOR i FROM 1 TO result.size DO
          resultP := result.get(i);
          IF resultP.ClId IN {UNCLASSIFIED, NOISE} THEN
```

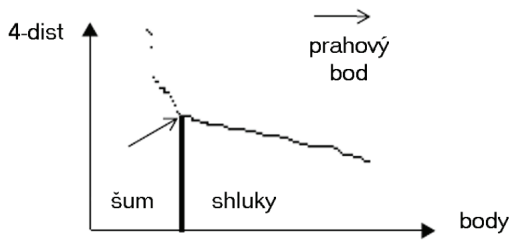
```
            IF resultP.ClId = UNCLASSIFIED THEN
              seeds.append(resultP);
            END IF;
            SetOfPoints.changeClId(resultP, ClId);
          END IF; // UNCLASSIFIED or NOISE
        END FOR;
      END IF; // result.size >= MinPts
      seeds.delete(currentP);
    END WHILE; // seeds <> Empty
    RETURN True;
  END IF;
END; // ExpandCluster
```

Pokud jsou dva shluky C_1 a C_2 velmi blízko, může se stát, že nějaký bod P patří do obou shluků. Potom musí být bod P hraničním bodem v obou shlucích, protože by jinak bylo C_1 totožné s C_2 (kvůli používání globálních parametrů). V takovém případě bude bod P přiřazen ke shluku, který byl nalezen dříve. Až na tyto vyjímečné situace, výsledek DBSCAN je nezávislý na pořadí vstupních bodů (důsledek Lemma **??**).

Určení parametrů ε a *MinPts*

Heuristika pro určení parametrů ε a *MinPts* „nejřidšího" shluku databáze je založena na následujícím pozorování. Nechť d je vzdálenost bodu P k jeho k-tému nejbližšímu sousedovi, potom d-okolí bodu P obsahuje právě $k+1$ bodů pro téměř všechny body P. d-okolí bodu P obsahuje více než $k+1$ bodů pouze tehdy, pokud několik bodů má přesně stejnou vzdálenost d od P, což je velmi nepravděpodobné. Kromě toho, změna k pro bod ve shluku nevede k velkým změnám d. K tomuto dochází pouze pokud k-tí nejbližší sousedé, pro $k = 1, 2, 3, \ldots$, leží přibližně na jedné přímce, ale to obecně neplatí pro bod ve shluku.

Pro dané k lze definovat funkci *k-dist* z databáze D do oboru reálných čísel, která mapuje každý bod na vzdálenost od jeho k-tého nejbližšího souseda. Při sestupném seřazení bodů databáze podle jejich *k-dist* hodnoty poskytuje graf této funkce několik nápověd týkajících se distribuce hustoty v databázi. Pokud je vybrán libovolný bod P, ε je nastaveno na *k-dist(P)* a parametr *MinPts* na k, všechny body s hodnotou menší nebo rovnou hodnotě *k-dist* budou jádra. Pokud lze nalézt *prahový bod* s maximální hodnotou *k-dist* v nejřidším shluku databáze D, získáme požadované hodnoty parametrů. Prahový bod je první bod v prvním „údolí" seřazeného k-dist grafu (obrázek 2.17). Všechny body s vyšší hodnotou *k-dist* (vlevo od prahu) jsou považovány za šum, všechny ostatní body (vpravo od prahu) jsou přiřazeny do některého shluku.



Obrázek 2.17: Seřazený 4-dist graf pro databázi 3 metodou DBSCAN [6].

Obecně je velice těžké detekovat první „údolí" automaticky, ale je to relativně jednoduché pro uživatele, aby viděl toto údolí v grafické reprezentaci. Proto je navržen interaktivní přístup pro určení prahového bodu.

DBSCAN potřebuje dva parametry, ε a *MinPts*. Ale z experimentů vyplynulo, že *k-dist* graf pro $k > 4$ se výrazně neliší od *4-dist* grafu (navíc potřebují mnohem více výpočtů). Tímto byl parametr *MinPts* eliminován a pro všechny 2D databáze se nastavuje na hodnotu 4. K interaktivnímu určení parametru ε je navržen následující přístup:

- Systém vypočítá a zobrazí 4-dist graf databáze.
- Pokud může uživatel odhadnout procento šumu, je toto procento zadáno a systém z něj odvodí návrh prahového bodu.
- Uživatel buď přijme navrhnutý práh nebo vybere jiný bod jako prahový. 4-dist hodnota prahového bodu je použita jako ε pro DBSCAN [6].

2.3.2 DENCLUE[4]

Data ukládaná v databázích jsou stále více složitější a pro efektivnější získávání bývají obvykle transformována na vysocedimenzionální vektory rysů. V mnoha aplikacích jsou tyto databáze velmi rozsáhlé a obsahují miliony datových objektů s desítkami až stovkami dimenzí. Hlavní problém existujících přístupů v souvislosti se shlukováním je ten, že mnoho algoritmů není navrženo pro shlukování vysocedimenzionálních vektorů rysů a tak jejich výkonnost rychle degeneruje s narůstajícím počtem dimenzí.

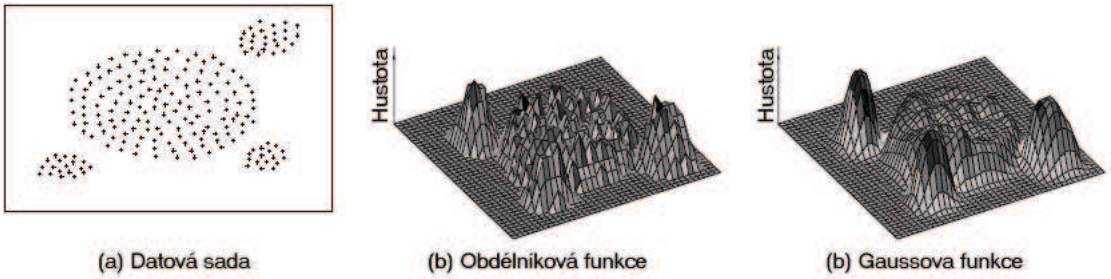
Základní myšlenkou přístupu tohoto algoritmu je modelování celkové hustoty bodu analyticky jako součtu funkcí vlivu datových bodů. Shluky tak mohou být identifikovány stanovením atraktorů hustoty. Libovolné tvary shluků mohou být snadno popsány jednoduchou rovnicí založenou na funkci celkové hustoty. Výhody tohoto přístupu jsou:

- Pevný matematický základ.
- Dobré shlukovací vlastnosti pro datové sady s velkým množstvím šumu.
- Umožňuje kompaktní matematický popis shluků libovolných tvarů ve vysocedimenzionálních datových sadách.
- Je výrazně rychlejší než existující algoritmy.

Funkci vlivu (A.2.1) lze vidět jako funkci, která popisuje dopad datového bodu na jeho sousedství. Příklady funkcí vlivu jsou parabolické funkce, obdélníková funkce nebo Gaussova funkce. Funkce vlivu je aplikována na každý datový bod. Celkovou hustotu (A.2.2) datového prostoru lze vypočítat jako součet funkcí vlivu všech datových bodů. Shluky jsou určeny matematicky identifikací atraktorů hustoty (A.2.4), což jsou lokální maxima celkové funkce hustoty. Pokud je celková funkce hustoty spojitá a má derivaci v každém bodě, určení atraktorů hustoty může být efektivně provedeno pomocí algoritmu *Hill-climbing* řízeného stoupáním (A.2.3) celkové funkce hustoty.

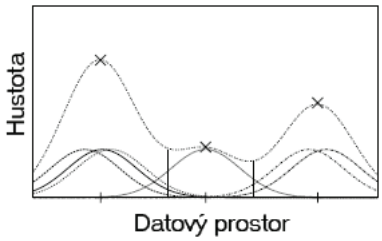
Obrázek 2.18 zobrazuje příklad sady datových bodů ve 2D prostoru společně s odpovídajícími funkcemi hustoty pro obdélníkovou a Gaussovu funkci vlivu.

[4] DENsity-based CLUstEring.



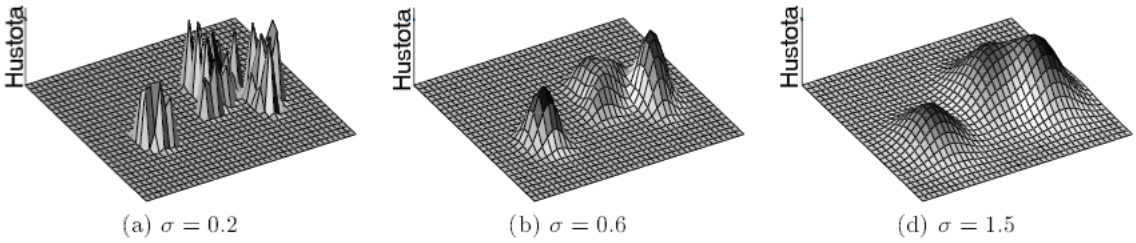
Obrázek 2.18: Příklad funkcí hustoty [10].

Lze definovat dvě různé pojetí shluků – shluky definované středem (A.2.5) a shluky libovolného tvaru (A.2.6). Pro definice je třeba pojem *atraktor hustoty*. Atraktory hustoty jsou neformálně lokální maxima celkové funkce hustoty a proto je také třeba definovat stoupání funkce hustoty.



Obrázek 2.19: Příklad atraktorů hustoty u algoritmu DENCLUE [10].

Obrázek 2.19 zobrazuje příklad atraktorů hustoty v jednodimenzionálním prostoru. Pro spojitou a derivovatelnou funkci vlivu může být použit jednoduchý gradientní algoritmus (hill-climbing) k nalezení atraktorů hustoty datového bodu $x \in D$. Algoritmus je řízen stoupáním f_B^D.

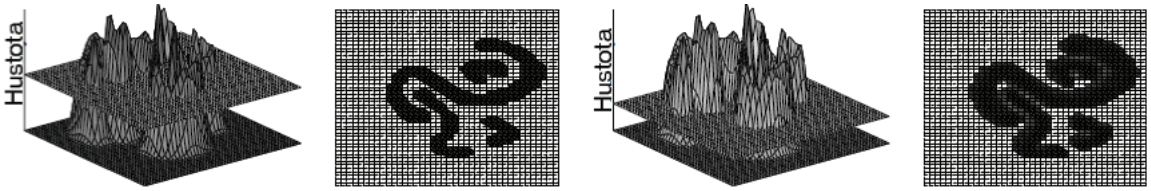


Obrázek 2.20: Příklad shluků definovaných středem pro různé σ algoritmu DENCLUE [10].

Obrázek 2.20 zobrazuje příklady shluků definovaných středem pro různé σ. Počet nalezených shluků je závislý na σ. Na obrázku 2.21a a 2.21c jsou příklady funkcí hustoty zároveň s rovinou pro různá ξ. Obrázky 2.21b a 2.21d zobrazují výsledné shluky libovolného tvaru. Parametr σ popisuje vliv datového bodu v datovém prostoru a ξ popisuje, kdy je atraktor hustoty významný.

Všeobecnost

Přístup metody DENCLUE zobecňuje ostatní shlukovací přístupy, zejména:



(a) $\xi = 2$ (b) $\xi = 2$ (c) $\xi = 1$ (d) $\xi = 1$

Obrázek 2.21: Příklad shluků libovolného tvaru pro různá ξ algoritmu DENCLUE [10].

- metody založené na rozdělování,
- hierarchické metody,
- metody založené na lokalitě.

Pokud je použita obdélníková funkce vlivu se $\sigma = \varepsilon$ a mez odlehlých hodnot $\xi = MinPts$, definované shluky libovolných tvarů jsou stejné jako shluky nalezené metodou DBSCAN.

Použitím různých hodnot parametru σ a pojetím shluků definovaných středem podle definice 11 je možné generovat hierarchii shluků. S velmi malou počáteční hodnotou σ se získá N^* shluků. Zvýšením σ se začnou atraktory hustoty slučovat a tak se získá další úroveň hierarchie. Každým dalším zvýšením hodnoty σ se sloučí více a více atraktorů hustoty, až nakonec vznikne pouze jeden atraktor hustoty reprezentující kořen hierarchie.

Volba parametrů

DENCLUE má dva důležité parametry: σ a ξ. Parametr σ určuje vliv bodu v jeho okolí a ξ popisuje, zda je atraktor hustoty významný, umožňuje redukci počtu atraktorů hustoty a zlepšuje výkonnost.

Výběr dobrého σ může být provedeno zvážením různých σ a stanovením nejširšího intervalu mezi σ_{max} a σ_{min}, kde počet atraktorů hustoty $m(\sigma)$ zůstává konstantní. Shlukování vzniklé z tohoto přístupu lze považovat za přirozeně adaptované na datovou sadu.

Parametr ξ je minimální úroveň hustoty pro atraktor hustoty, aby byl významným. Pokud je ξ nastaveno na nulu, všechny atraktory hustoty, dohromady s jejich datovými body atrakovanými na základě hustoty, budou určeny jako shluky. Toto není ovšem často žádoucí, protože se každý bod může stát svým vlastním shlukem (zvláště v oblastech s nízkou hustotou). Dobrá volba ξ pomůže algoritmu zaměřit se na hustě obsazené oblasti a ušetřit výpočetní čas.

Pro efektivní určení shluků je třeba nalézt možnost efektivního výpočtu funkce hustoty a atraktorů hustoty. Důležité je zjištění, že výpočet hustoty bodu $x \in F^d$ závisí pouze na těch bodech datové sady, které jsou v blízkosti x, a přispívají tak k hustotě. Všechny ostatní body mohou být zanedbány bez podstatné chyby.

Lokální funkce hustoty (A.2.7) je aproximací celkové funkce hustoty. Myšlenkou je brát v úvahu pouze vliv blízkých bodů, zatímco je zanedbán vliv bodů vzdálených. Vzniká tak chyba, která je ovšem zaručeně v úzkých mezích. K definování lokální funkce hustoty je třeba funkce $near(x)$, kde $x_1 \in near(x) : d(x_1, x) \leq \sigma_{near}$.

Algoritmus

DENCLUE pracuje ve dvou krocích. První je předshlukovací krok, ve kterém je vytvořena mapa relevantní části datového prostoru. Mapa je použita ke zrychlení výpočtu funkce hustoty, která vyžaduje efektivní přístup k sousedním částem datového prostoru. Druhý krok je shlukovací, v němž algoritmus identifikuje atraktory hustoty a odpovídající atrakované body na základě hustoty.

Krok 1

Minimální ohraničující (hyper-)obdélník datové sady je rozdělen na d-dimenzionální hyperkrychle s délkou hrany 2σ. Uvažují se pouze hyperkrychle, které obsahují datové body. Počet vzniklých krychlí je v rozsahu 1 až N^* v závislosti na zvoleném σ, ale nezávisí na dimenzionalitě datového prostoru. Hyperkrychle jsou číslovány podle jejich relativní pozice od počátku. Tímto způsobem mohou být hyperkrychle mapovány na jednodimenzionální klíče, které lze efektivně uložit do vyhledávacího nebo B^+ stromu. Pro každou krychli $c \in C_p$ je uložen počet jejich datových bodů (N_c), ukazatele na tyto body a lineární součet $\sum_{x \in c} x$. Tyto informace jsou využity ve shlukovacím kroku pro rychlý výpočet průměru krychle $mean(c)$. Protože může být shluk rozprostřen přes více krychlí, je nutný přístup také k sousedícím krychlím. Pro urychlení tohoto přístupu jsou sousedící krychle propojeny.

Krok 2

Pouze krychle s vyšší hustotou a krychle s nimi propojené jsou uvažovány pro hledání shluků. Díky struktuře z předchozího kroku je možné efektivně vypočítat lokální funkci hustoty $\hat{f}^D_{Gauss}(x)$. Stejně tak lze vypočítat lokální stoupání $\bigtriangledown \hat{f}^D_{Gauss}(x)$. K určení atraktorů hustoty každého bodu krychlí, založených na lokální funkci hustoty a stoupání, je použita metoda hill-climbing [10].

DENCLUE 2.0

Nevýhodou předchozí verze algoritmu DENCLUE je, že použitá metoda hill-climbing může provádět zbytečně malé kroky na začátku a nikdy nekonverguje k maximu, ale pouze se k němu blíží. Ve vylepšené verzi je velikost kroku automaticky upravována bez ztráty na rychlosti. Provádí se mnohem méně iterací a výpočet lze urychlit metodami založenými na vzorkování pouze za cenu malé nepřesnosti [9].

Kapitola 3

Databáze proteinových rodin

Skoro všechny proteiny mají strukturální podobnosti s ostatními proteiny. V mnoha případech sdílejí společný evoluční původ. Znalost těchto vztahů je významným přínosem pro molekulární biologii a další související oblasti vědy. Jedná se o střed našeho chápání struktury a evoluce proteinů. Hraje důležitou roli v interpretaci sekvencí produkovaných genomovými projekty a pro pochopení evolučního vývoje.

Stále rostoucí počet proteinů, jejichž struktury byly získávány rentgenovou krystalografií a NMR[1] spektroskopií, si vyžádal vznik proteinových databází, které usnadňují chápání a přístup k těmto informacím [20].

3.1 SCOP[2]

Databáze poskytující detailní a komplexní popis strukturálních a evolučních vztahů mezi proteiny, u kterých je známa jejich struktura. Pro vytvoření klasifikace proteinů v této databázi se zásadně používá vizuální kontrola a porovnání struktur. Aby byla tato úloha zvládnutelná, využívají se různé automatické nástroje pomáhající zajistit obecnost. Jednotkou pro klasifikaci je zpravidla proteinová doména. Malé proteiny, a většina střední velikosti, mají jedinou doménu a jsou proto považovány za celek. Domény ve větších proteinech se obvykle klasifikují zvlášť.

Klasifikace je na hierarchické úrovni, která vyjadřuje evoluční a strukturální vztahy:

- **Rodina** – proteiny jsou shlukovány do rodin na základě jednoho ze dvou kritérií: všechny proteiny mají identitu residuí alespoň 30 %, nebo proteiny s nižší identitou mají velmi podobnou funkci a strukturu.
- **Superrodina** – rodiny, jejichž proteiny mají nižší identitu, ale jejich struktury, v mnoha případech i funkční vlastnosti, naznačují, že je pravděpodobný společný evoluční původ, jsou zařazeny do jedné superrodiny.
- **Společný „tvar" (fold)** – superrodiny a rodiny jsou definované jako se společným tvarem, pokud jejich proteiny mají stejnou sekundární strukturu ve stejném uspořádání se stejnými spojeními. Různé proteiny se stejným tvarem mají obvykle obvodové prvky sekundární struktury a regiony otáček, které se liší svou velikostí a prostorovým uspořádáním. V případech větší odlišnosti mohou tyto lišící se regiony tvořit více než

[1] Nuclear magnetic resonance
[2] Structural Classification of Proteins.

polovinu každé struktury. Strukturální podobnosti pro proteiny zařazené do stejného tvaru pravděpodobně vznikají z fyzikálních a chemických vlastností proteinů, které upřednostňují jisté uspořádání sbalení a topologie řetězce.

- **Třída** – různé tvary jsou pro uživatelský komfort seskupeny do tříd:
  1. **Vše alfa** – pro proteiny, jejichž struktura je v zásadě složena z α-spirál.
  2. **Vše beta** – struktura je v zásadě složena z β-listů.
  3. **Alfa a beta** – α-spirály a β-listy jsou z velké části proložené.
  4. **Alfa + beta** – α-spirály a β-listy jsou z velké části oddělené.
  5. **Vícedoménové** – obsahující domény z různých tvarů, pro které nejsou v současné době známé homology.

  Méně obvyklé proteiny, peptidy a PDB záznamy pro navržené proteiny, teoretické modely, nukleové kyseliny a karbohydráty byly zařazeny do dalších tříd.

Každý záznam v databázi má k dispozici odkazy na souřadnice a obrázky struktury, interaktivní molekulární prohlížeče, data sekvence a referenční literaturu. Lze také provádět dva druhy vyhledávání:

- K vyhledání homologií může uživatel zadat sekvenci a získá seznam struktur, které jí jsou významně podobné.
- Hledání shod uživatelem zadaného klíčového slova v textu SCOP databáze a v hlavičkách strukturních souborů.

Databáze je volně přístupná na adrese http://scop.mrc-lmb.cam.ac.uk/scop/ [20], její další vývoj byl ovšem ukončen verzí 1.75. Obsahuje manuálně klasifikované struktury, tzn. že každou strukturu musel posoudit a zařadit zkušený biolog. Statistiky databáze SCOP 1.75 z června 2009, obsahující 110 800 domén, jsou uvedeny v tabulce 3.1.

Třída	**Počet**		
	tvarů	**superrodin**	**rodin**
a: Vše alfa proteiny	284	507	871
b: Vše beta proteiny	174	354	742
c: Alfa a beta proteiny	147	244	803
d: Alfa + beta proteiny	376	552	1 055
e: Vícedoménové proteiny	66	66	89
f: Proteiny a peptidy membrán a buněčného povrchu	58	110	123
g: Malé proteiny	90	129	219
Celkem	**1 195**	**1 962**	**3 902**

Tabulka 3.1: Statistiky databáze SCOP 1.75.

3.2 SCOP2[3]

SCOP2 je nástupcem databáze SCOP a slouží ke stejnému účelu. Byla navržena k poskytování pokročilejšího aplikačního rámce pro anotaci a klasifikaci proteinových struktur.

[3] Structural Classification of Proteins 2.

Definuje nový přístup ke klasifikaci proteinů, který se od původní SCOP zásadně liší, ale zachovává její nejlepší vlastnosti. Klasifikace je popsána orientovaným acyklickým grafem (obrázek 3.1), jehož uzly tvoří komplexní síť vztahů „mnoho k mnoha" a jsou reprezentovány regionem proteinové struktury a sekvence. Důležité je, že tato struktura může obsahovat více než jeden rodičovský uzel pro uzel potomka.

Vztahy ve SCOP2 spadají do čtyř velkých kategorií. Dvě z nich, *Proteinové typy* a *Evoluční události*, nemají protějšky v původní SCOP databázi:

- *Proteinové typy* – proteiny jsou rozděleny do čtyř hlavních typů: rozpustný, membránový, vláknitý a vnitřně nezřízený, z nichž každý do značné míry koreluje s charakteristikou sekvence a strukturními vlastnostmi.
- *Evoluční události* – směřují k usnadnění anotace různých strukturálních uspořádání a zvláštností, které byly pozorovány mezi příbuznými proteiny a daly vzniknout podstatným strukturálním rozdílům.
- *Strukturální třídy* – organizují proteinové tvary přesně podle jejich sekundárního strukturálního obsahu. Staly se atributem proteinového tvaru a proto je nezávislé na proteinovém evolučním vztahu. Podobné tvary s odlišnou sekundární strukturou jsou umístěny do různých tříd.
- *Proteinové vztahy* se skládají ze tří podkategorií:
  - *strukturální*,
  - *evoluční*,
  - *ostatní*.

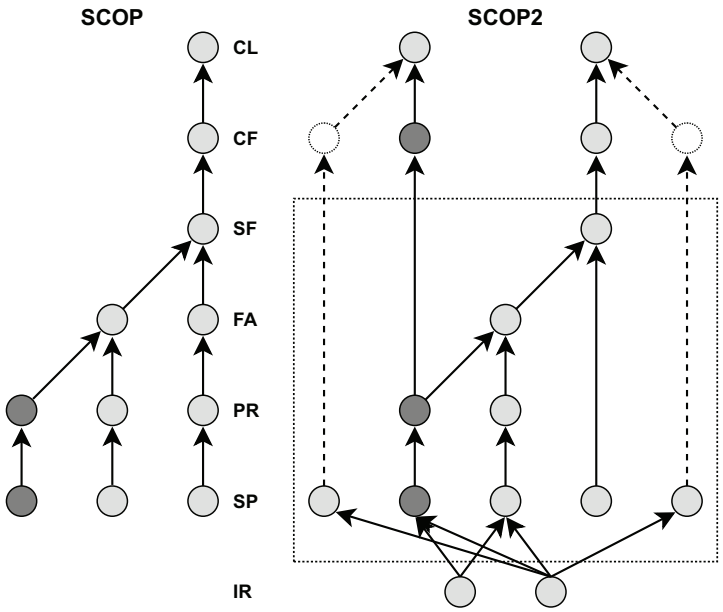
  Vztah *ostatní* má za cíl definovat a anotovat vztahy, jako např. interní strukturální repetice, společné motivy a podtvary, které nebyly předmětem klasifikace ve SCOP.

Evoluční vztahy *druhy*, *protein*, *rodina* a *superrodina* ze SCOP jsou zachovány, ale jejich obsah a definice se liší:

- *Druhy* – korespondují s individuálním genovým produktem a jsou reprezentovány jejich plnou délkou sekvence.
- *Protein* – seskupuje ortologní proteiny a je definován jako podsekvence, která může být nalezena samostatně.
- *Rodina* – koresponduje s konzervovaným regionem sekvence sdíleným úzce souvisejícími proteiny.
- *Superrodina* – reprezentována společným strukturálním regionem sdíleným různými proteinovými rodinami.

Důležité je, že domény reprezentující úroveň rodiny a superrodiny mohou zasahovat do více než jedné strukturální domény. Kromě toho se zavádí nová úroveň *Hyperrodina*, především k pokrytí nejobsáhlejších a strukturálně různorodých SCOP superrodin. Pozoruhodným rozdílem mezi SCOP a SCOP2 je to, že tyto odlišné úrovně nejsou povinné.

Klasifikace SCOP2 je založena na reprezentativních sekvencích a strukturách. Jejich manuální anotace je pak automaticky rozšířena o související záznamy [1]. Databázi lze volně získat z `http://scop2.mrc-lmb.cam.ac.uk/`.



Obrázek 3.1: Porovnání grafů databází SCOP a SCOP2. Vlevo je část hierarchického stromu SCOP, kde je klasifikace do šesti povinných úrovní: *Proteinový druh* (SP), *Protein* (PR), *Rodina* (FA), *Superrodina* (SF), *Tvar* (CF) a *Třída* (CL). Homologní proteiny s odlišnými tvary jsou nuceně zařazeny do stejné rodiny a také postupně do stejné superrodiny, tvaru a třídy. Na každé úrovni je povinný uzel od kořene k listu, i když tento uzel nereprezentuje žádný aktuální vztah. Vpravo ve SCOP2 jsou strukturální a evoluční vztahy odděleny, umožňují tak klasifikaci homologních proteinů do různých tvarů a strukturálních tříd se zachováním jejich stejné evoluční rodiny a superrodiny. Nepovinné jedno-potomkové uzly jsou vynechány. Nová specifická kategorie *ostatní vztahy* (IR) je na grafu také zobrazena. Tyto vztahy zahrnují (ale nejsou omezené na) nehierarchické vztahy mezi homologními a nehomologními proteiny různých tvarů sdílejících velké společné podstruktury nebo motivy [1].

3.3 SCOPe[4]

Databáze proteinových strukturálních vztahů, která rozšiřuje poslední verzi 1.75 původní databáze SCOP. Pomocí metod automatizované správy klasifikuje mnoho struktur vydaných od této poslední verze. SCOPe je také částečně manuálně spravována pro opravení některých chyb. Snaží se být zpětně kompatibilní a poskytující stejné zpracovatelné soubory a historii změn mezi všemi stabilními vydáními [7].

Zdarma lze databázi získat na `http://scop.berkeley.edu/`. Statistiky poslední verze databáze SCOPe 2.05 z února 2015, obsahující 203 026 domén, jsou uvedeny v tabulce 3.2.

3.4 Pfam[5]

Jednou z hlavních výzev při vytváření proteinových databází je současné splnění protichůdných požadavků úplnosti na jedné straně a kvality zarovnání a definic domén na straně

[4]Structural Classification of Proteins–extended.
[5]Protein families.

Třída	Počet		
	tvarů	superrodin	rodin
a: Vše alfa proteiny	286	509	1037
b: Vše beta proteiny	176	359	931
c: Alfa a beta proteiny	148	245	965
d: Alfa + beta proteiny	381	558	1 301
e: Vícedoménové proteiny	68	68	109
f: Proteiny a peptidy membrán a buněčného povrchu	57	113	153
g: Malé proteiny	92	132	260
Celkem	**1 208**	**1 984**	**4 756**

Tabulka 3.2: Statistiky databáze SCOPe 2.05.

druhé. Poslední jmenované vlastnosti se řeší v nejlepším případě manuálním přístupem, zatímco úplnost je v praxi předmětem pouze automatických metod. Pfam má základ ve skrytých markovských modelech (HMMs[6]), které kombinují vysokou kvalitu a úplnost.

Databáze se skládá z částí A a B. Přesná Pfam-A obsahuje dobře charakterizované rodiny proteinových domén s vysoce kvalitním zarovnáním. Zachovává si použití manuální kontroly zarovnání semínek a využívá HMMs k nalezení a zarovnání všech jejich členů. Pfam-B obsahuje rodiny sekvencí, které byly vygenerovány automaticky aplikací shlukovacího algoritmu, a zarovnání zbývajících proteinových sekvencí po odstranění domén nacházejících se v Pfam-A. Dohromady tedy tvoří kompletní celek shluků proteinových sekvencí. Snahou je postupně převést největší rodiny z Pfam-B do Pfam-A [24].

Databáze je zdarma k dispozici na webu `http://pfam.xfam.org/`. Statistiky Pfam 27.0 z března 2013 jsou uvedeny v tabulce 3.3.

Část	Počet		
	rodin	sekvencí	residuí
Pfam-A	14 831	18 523 877	4 413 005 459
Pfam-B	544 866	3 843 092	427 492 613
Celkem	**559 697**	**22 366 969**	**4 840 498 072**

Tabulka 3.3: Statistiky databáze Pfam 27.0.

3.5 CATH[7]

Databáze CATH používá proteinovou strukturu jako citlivou sondu pro vzdálené evoluční vztahy a také pro poskytnutí další vrstvy náhledu na vztah mezi sekvencí a její funkcí. Je to hierarchická klasifikace proteinových domén, proto se struktury rozdělí na jednotlivé domény. Pokud existuje dostatečný důkaz, že množina domén zřetelně sdílí společného předka, jsou seskupeny do jedné homologní superrodiny. Když homologní superrodiny (úroveň „H") sdílí stejný tvar, ale neexistuje dostatečný důkaz o jednoznačném evolučním vztahu, umístí se do stejné topologie (úroveň „T"). Topologie, které sdílejí přibližně podobné prostorové

[6] Hidden Markov Models.
[7] Class, Architecture, Topology, Homology.

uspořádání sekundární struktury, se seskupí do stejné architektury (úroveň „A”). 40 architektur databáze CATH je uspořádáno do jedné ze 4 hlavních tříd (úroveň „C”) na základě obsahu sekundární struktury:

- převážně alfa,
- převážně beta,
- smíšené alfa a beta,
- málo sekundárních struktur.

CATH lze získat z http://www.cathdb.info/ také zcela zdarma [23]. Statistické údaje o její poslední verzi 4.0, založené na datech z března 2013, jsou shrnuty v tabulce 3.4.

Třída	**Počet**			
	architektur	**topologií**	**superrodin**	**domén**
1	5	397	907	48 121
2	20	241	547	58 944
3	14	626	1 158	125 772
4	1	111	126	3 021
Celkem	**40**	**1 375**	**2 738**	**235 858**

Tabulka 3.4: Statistiky databáze CATH 4.0.

Kapitola 4

Nedostatky současných algoritmů

Kapitola 2 popisuje známější a používanější algoritmy pro shlukování proteinů. Většina se týká zajímavých přístupů a heuristik, které byly vytvořeny s cílem dosáhnout rozumné přesnosti na velké vstupní datové sadě. Obvykle je to vždy kompromis mezi přesností a výpočetní náročností, který může ovlivnit kvalitu výsledků. Za hlavní nedostatky lze považovat především způsob vzájemného porovnávání sekvencí, jejich zařazování do shluků a ukončování samotného procesu shlukování.

Identita sekvencí

Porovnávání sekvencí se v mnoha případech řídí identitou (2.1.1, 2.1.2, 2.2.1) mezi částmi jednotlivých proteinů, která v tomto případě není nejvhodnějším měřítkem. Mnohem lepším způsobem porovnávání je zaměřit se na podobnost těchto částí, aby byly zohledněny případné jednobodové (či vícebodové) záměny jednotlivých aminokyselin. Může nastat situace, že zaměněné aminokyseliny plní v daném proteinu velice podobnou, nebo dokonce identickou funkci.

Složitost algoritmu

Náročnost výpočtu samotného porovnávání je také problémem. Často dochází k porovnávání každé vstupní sekvence se všemi ostatními vstupními sekvencemi (2.2.1, 2.2.2). Takový proces má složitost $O(N^2)$ a trvá velmi dlouho. Ve většině případů je i zbytečný, protože některé sekvence mají s dalšími společného velmi málo. Pokud algoritmus využívá na svém počátku výběr reprezentantů, tzv. centroidů (2.1.1, 2.1.2, 2.1.3) nebo semínek (2.2.3), nedochází k tolika porovnáním, ale stále lze tento počet redukovat. Odpovědí na rychlejší porovnávání může být algoritmus třídící vstupní sekvence určitým filtrem (2.1.1, 2.1.2, 2.1.3), aby rozeznal, které bude nebo nebude mezi sebou porovnávat. Ovšem pro získání kvalitnějšího výsledku se lze smířit s větší časovou náročností.

Princip shlukování

Pomocí identity je prováděno i samotné shlukování (2.1.1, 2.1.2, 2.2.1), které může být méně přesné. Předpokládá se, že všechny shluky budou mít uvnitř stejnou míru identity, takže jsou všechny sekvence porovnávány a seskupovány při překročení nějaké, pro všechny shluky stejné, prahové hodnoty. Tento přístup neřeší situaci, kdy jsou různé shluky vnitřně identické s různým prahem, popř. zmíněnou podobností. Navíc mnoho algoritmů neumí pracovat na identitách pod 30–40 % (2.1.1, 2.1.2). Bylo by vhodné využít takový způsob

shlukování, který není závislý na uživatelem zadané vstupní hodnotě identity, popř. ani podobnosti, a zároveň by dokázal pracovat se sekvencemi s původním významem identity pod 30–40 % (2.1.3).

Ukončující podmínka algoritmu

Za jeden z posledních problémů můžeme považovat definovanou podmínku ukončení algoritmu. Některé vyžadují uživatelem zadaný počet výsledných shluků (k-means), jiné končí při dosažení již zmíněného prahu identity ve všech shlucích (2.1.1, 2.1.2) apod. Algoritmus musí být ukončen, pokud jsou všechny sekvence přiřazeny svým adekvátním shlukům, ale případné odlehlé hodnoty zůstávají osamoceny, tj. každá odlehlá hodnota patří do svého shluku (2.3.1, 2.3.2).

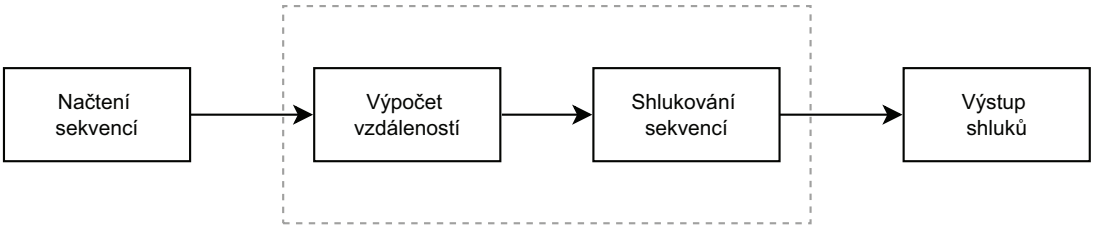
Pravděpodobně nelze nalézt metodu, která by absolutně řešila všechny zde zmíněné nedostatky současně, ale je třeba co nejvíce snížit negativní dopady na výsledek shlukování většiny z nich.

Kapitola 5

Návrh vlastního nástroje

Snahou je odstranit co nejvíce a co nejlépe současné nedostatky uvedené v předchozí kapitole 4. Při shlukování chceme získat především přesné shluky, a to i za cenu větší časové náročnosti. V této kapitole je uveden návrh vlastního řešení shlukování, které se toho pokusí po implementaci dosáhnout.

Shlukovací nástroj lze rozdělit na dvě hlavní části: výpočet vzdáleností mezi sekvencemi a následné shlukování na základě těchto vzdáleností. Schéma základních kroků nástroje zobrazuje schéma 6.2.



Obrázek 5.1: Schéma základních kroků shlukovacího nástroje.

5.1 Podobnost sekvencí

Porovnávání sekvencí na základě identity bude nahrazeno porovnáváním na podobnost, které by mělo být ve vztahu příbuznosti proteinů přesnější, protože můžou obsahovat chybu v podobě záměny, vložení nebo odstranění aminokyseliny ze sekvence. Optimální algoritmy zarovnávání mají ovšem kvadratickou složitost a jsou nevhodné zvláště pro rozsáhlé vstupní sady, bude tedy nutné použít některou rychlejší heuristiku. Dvě takové heuristiky jsou FASTA[1] a BLAST[2]. Obě provádí lokální zarovnání. FASTA na svém začátku vyhledává pouze identické segmenty v sekvencích a následně je propojuje do větších oblastí. Prohledávání má citlivější a výsledné zarovnání přesnější. Oproti BLASTu vyžaduje ale více výpočetního času. BLAST naopak pracuje s chybami aminokyselin hned a o něco rychleji. Jeho hlavním principem je rozšiřování nalezených výskytů. Při experimentování budou vyzkoušeny oba přístupy.

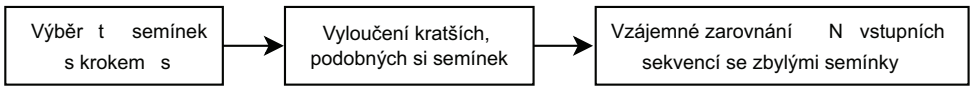
Jádrem pro výpočet podobností mezi všemi sekvencemi bude algoritmus mBed popsaný v podkapitole 2.2.3. Jednou z jeho hlavních výhod je rychlý výpočet podobnosti pro všechny

[1] FAST-All.
[2] Basic Local Alignment Search Tool

sekvence vstupní sady, kdy se neprovádí vzájemné porovnávání každé s každou. Algoritmus má podmínku, že vstupní sekvence musí být seřazeny sestupně podle jejich délky v aminokyselinách. Pokud nebudou seřazeny od uživatele, nástroj tento stav detekuje a seřazení provede automaticky.

Na začátku algoritmus vybere vzorek $t = (log_2 N)^2$ sekvencí. Jednotlivé sekvence do vzorku jsou vybírány z již seřazených, a to s konstantním krokem. Krok ovšem nelze nastavit pevně, protože by se pořadí vybírané sekvence mohlo dostat mimo počet, který je na vstupu, nebo by byl vybrán pouze ze začátku velmi dlouhé vstupní sady. Velikost kroku bude tedy stanovena z počtu sekvencí na vstupu jako $s = log_{10} N$. Vzorky budou ještě vzájemně porovnány, zda si nejsou s určitou (implicitně nulovou) prahovou hodnotou podobné. Z podobných vzorků se vyloučí ten kratší.

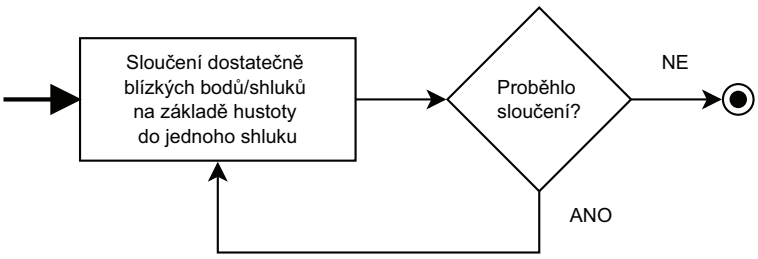
V dalším kroku algoritmu je každé sekvenci přiřazen t-rozměrný vektor. Jednotlivé souřadnice vektoru reprezentují podobnost dané sekvence s t vzorky. Vzdálenost takových vektorů sekvencí by měla být velmi blízká jejich podobnosti. Schéma algoritmu vektorizace znázorňuje obrázek 5.2.



Obrázek 5.2: Schéma algoritmu vektorizace sekvencí.

5.2 Algoritmus shlukování

Shlukování je druhou důležitou částí a hlavním zaměřením práce na nástroji. Protože se bude snažit produkovat shluky s různou vnitřní podobností, byl vybrán princip shlukování na základě hustoty, které nabízí algoritmy DBSCAN (2.3.1) a DENCLUE (2.3.2). Tyto algoritmy končí, pokud nelze sloučit žádné dva dosud objevené shluky. Můžou tak vzniknout shluky různých tvarů a různých vnitřních podobností. DBSCAN bude používat základní obdélníkovou funkci pro určení sousedů v okolí bodu, tzn. vzdálenost dvou bodů se porovná pouze s hodnotou ε. DENCLUE pracuje s funkcí hustoty a vlivu bodu na jeho okolí. Obě funkce budou implementovány jako Gaussovy s parametrem σ. DENCLUE navíc používá algoritmus hill-climbing, který bude v jeho základní verzi, tj. bez heuristiky pro výpočet velikosti následujícího kroku. Diagram průběhu a ukončení algoritmu shlukování na základě hustoty je na obrázku 5.3.



Obrázek 5.3: Diagram algoritmu shlukování na základě hustoty.

5.3 Vstup a výstup

Vstupem běžně používaných programů pro shlukování bývá FASTA soubor se sekvencemi. Nový nástroj bude tedy načítat sekvence ve stejném formátu. Navíc bude mít možnost načítat již vypočítané vektory s podobnostmi, aby je nebylo nutné pro stejnou datovou sadu počítat stále znovu. Výsledek výpočtu vektorů se při použití stejného principu zarovnání na stejnou datovou sadu neliší, tzn. že je deterministický. Příklad záznamů tří sekvencí z databáze SCOP 1.75 (3.1) ve FASTA souboru:

```
>d2hmva1 c.2.1.9 (A:7-140) Ktn bsu222 {Bacillus subtilis [TaxId: 1423]}
kqfaviglgrfggsivkelhrmghevlavdineekvnayasyathavianateenellsl
girnfeyvivaiganiqastlttlllkeldipniwvkaqnyyhhkvlekigadriihpek
dmgvkiaqslsden
>d1j75a_ a.4.5.19 (A:) Dlm-1 {Mouse (Mus musculus) [TaxId: 10090]}
nleqkilqvlsddggpvkigqlvkkcqvpkktlnqvlyrlkkedrvsspepatwsig
>d2fg5a1 c.37.1.8 (A:3-167) Rab31 {Human (Homo sapiens) [TaxId: 9606]}
irelkvcllgdtgvgkssivcrfvqdhfdhnisptigasfmtktvpcgnelhkfliwdta
gqerfhslapmyyrgsaaavivyditkqdsfytlkkwvkelkehgpenivmaiagnkcdl
sdirevplkdakeyaesigaivvetsaknainieelfqgisrqip
```

Hlavním výstupem musí být shluky určené algoritmem. Formát těchto souborů by měl být co nejjednodušší, ale zároveň dobře interpretovatelný. Dalším výstupním formátem budou vypočítané vektory pro jednotlivé sekvence, aby je bylo možné znovu použít při dalších bězích algoritmu.

Kapitola 6

Implementace

Program s řešením vlastního nástroje je naimplementován v programovacím jazyce Java. K diplomové práci bylo potřeba použít i několik pomocných skriptů (např. pro převod souborů se shluky do jednotného formátu, spouštění externích aplikací), které jsou napsány ve skriptovacím jazyce Python, popř. v Bashi (B.1). Tato kapitola rozebírá způsob řešení jednotlivých programových součástí praktické části práce. Informace o použitých třídách Javy jsou čerpány z její dokumentace[1].

6.1 Formáty datových souborů

Nástroj pracuje se dvěma vlastními formáty datových souborů. Jedním pro výsledné shluky a druhým pro zápis vytvořených vektorů.

Jednotný formát souborů se shluky

Výstupy implementovaného nástroje budou porovnávány s výstupy algoritmů UCLUST a CD-HIT. Aby bylo vzájemné porovnávání výsledných sad shluků jednodušší, byly implementovány skripty `uc2cls` a `clstr2cls`. Ty převádějí jejich výstupy na stejný formát, jako má navržený nástroj. Porovnání dvou sad shluků na podobnost je možné skriptem `2ClsSimilarity`.

Pro jednotný formát CLS byl zvolen jednoduchý tabulkový zápis v podobě CSV[2], kdy každý řádek souboru představuje jeden shluk. Názvy sekvencí zařazených do shluku (ležícího na jednom řádku) nejsou ovšem vzájemně odděleny čárkou, ale středníkem, protože se čárka používá pro zápis desetinného čísla. Formát díky své jednoduchosti nezvětšuje velikost souborů dalšími znaky.

Příklad obsahu souboru se třemi shluky, které zahrnují postupně 1, 2 a 3 sekvence:

```
d1eyxb_
d2aa1b1;d1h97a_
d1mbaa_;d1q1fa_;d1wdka2
```

[1] https://docs.oracle.com/javase/

[2] Comma-Separated Values, čárkami oddělené hodnoty.

Vypočítané vektory

Po vektorizaci vstupních sekvencí je možné jejich vektory zapsat do výstupního souboru, případně vytvořit vlastní vektory a nástrojem načíst pro následné shlukování. Formát vektorů opět používá zápis CSV, kde v prvním sloupci očekává název sekvence a poté seznam jeho souřadnic oddělených také středníkem.

Příklad souboru se čtyřmi vektory:

```
d2mysa2;0;32.9;32.9;18.2;31.2;50;20
d1kk8a2;21.9;31.4;31.8;14.3;18.2;56.4;40
d1w7ja2;42;37.6;37.9;38.9;100;35.7;60
d1d0xa2;32.9;0;0.5;12.5;14.3;28.6;15
```

6.2 Shlukovací nástroj FitClust

Vytvořený nástroj, pojmenovaný FitClust, nemá grafické uživatelské rozhraní, ale spouští se pouze z příkazového řádku. Třída `Main` pro jeho spouštění se nachází v Java balíčku `cz.vutbr.fit.xkubis03.dp`. Tento balíček obsahuje několik podbalíčku se součástmi nástroje:

- `dbscan` – algoritmus DBSCAN,
- `denclue` – algoritmus DENCLUE,
- `fitclust` – hlavní balíček s nástrojem FitClust,
- `readers` – třídy pro načítání datových souborů,
- `structures` – struktury používané nástrojem a jeho součástmi.

6.2.1 Načítání vstupních souborů

Most mezi bajtovou reprezentací souboru a jednotlivými znaky je zajištěn pomocí objektu třídy `InputStreamReader`. Ten se předává objektu třídy `BufferedReader`, aby bylo možné načítat po jednotlivých řádcích. Obě třídy jsou importovány z balíčku `java.io`. Pokud se načítá soubor ve formátu CSV, je využit `StringTokenizer` z balíčku `java.util`, který umí rozdělit text dle zadaného oddělovače. V případě implementovaného nástroje se jedná vždy o oddělující středník. Při načítání FASTA souborů nelze tento „rozdělovač" využít a jejich zpracovávání je řešeno skrze podmínky. Sekvence se ihned při načítání porovnávají na délku a pokud nejsou seřazeny od nejdelší po nejkratší, provede se jejich seřazení automaticky.

Práci s FASTA soubory obstarává třída `FASTAReader`. Pro načítání vektorů slouží třída `VectorsReader`. Obě jsou umístěny v podbalíčku `readers`.

6.2.2 Datové struktury

Nástroj pracuje s mnoha vlastními datovými strukturami z podbalíčku `structures`. Ty uchovávají informace pro různé kroky algoritmů. Jedná se zejména o:

- `Clusters` – množina výsledných shluků,
- `Cube` – krychle pro algoritmus DENCLUE,

- `FASTAItem` – jeden FASTA záznam sekvence,
- `FitClustItem` – spojení FASTA záznamu a jeho vektoru,
- `Point` – bod ve shlukovaném prostoru algoritmy DENCLUE a DBSCAN,
- `PointSet` – množina/shluk bodů,
- `Space` – prostor algoritmu DENCLUE,
- `Vector` – shlukovaný vektor sekvence.

Kromě základních datových typů `int`, `double` a `String` jsou struktury složeny z kolekcí `ArrayList`, `HashMap`, `HashSet`, `Map` a `Set`. Všechny importované z balíčku `java.util`. Funkce `toString` reprezentující objekty textovým řetězcem používají třídu `DecimalFormat` balíčku `java.text`, která zajišťuje formátování desetinných čísel.

Třída `FASTAItem` implementuje rozhraní `Comparable` a z něj funkci `compareTo` za účelem řazení vstupních sekvencí dle délky. Řazení sekvencí je pak možné díky objektu komparátoru `FASTAItemComparator`.

6.2.3 Algoritmus DBSCAN

Podbalíček `dbscan` obsahuje jednu třídu `DBScan` pro tento algoritmus. Veřejné jsou pouze konstruktory nastavující parametry (ε, *MinPts*) s množinou shlukovaných bodů a funkce `getClusters`, která provádí algoritmus a vrací objekt třídy `Clusters` s výslednými shluky. Další funkce rozšiřující shluk (`enlargeCluster`) a funkce vracející dostupné sousedy bodu (`getNeighbors`) jsou privátní a tedy zvenku nedostupné.

6.2.4 Algoritmus DENCLUE

Implementace algoritmu DENCLUE je v podbalíčku `denclue`. Stejně jako DBSCAN obsahuje pouze jednu třídu `DenClue`. Konstruktorů má hned několik, záleží na tom, které všechny parametry mají být nastaveny. Lze postupně (shora-dolů) definovat tyto:

- `sigma` – parametr σ pro vliv bodu,
- `xi` – parametr významnosti atraktoru ξ (implicitně 4),
- `maxIterations` – maximální počet iterací metody hill-climbing (implicitně 5),
- `delta` – parametr pro stoupání metody hill-climbing (implicitně 1).

Jedinou veřejnou funkcí je opět `getClusters` vracející objekt `Clusters` se shluky. Privátní funkce třídy, které algoritmus využívá, ale nejsou dostupné zvenku:

- `mergeClusters` – slučuje shluky, pokud mezi nimi existuje cesta,
- `determineAttractors` – určení atraktorů,
- `calcInfluence` – výpočet vlivu mezi dvěma body,
- `calcDensity` – výpočet hustoty v bodu,
- `calcGradient` – výpočet bodu stoupání,

- `getDensAttractor` – určení nového atraktoru,
- `pathExist` – vrací logickou hodnotu, zda existuje cesta mezi body a jejich shluky.

6.2.5 FitClust

Nástroj FitClust je umístěn v podbalíčku `fitclust` a skládá se ze dvou tříd.

Třída `Alignment` zajišťuje spouštění externích aplikací pro výpočet zárovnání dvou sekvencí. K dispozici jsou veřejné funkce `alignFASTA` a `alignBLAST`. Privátní funkce `align` obaluje společnou (obecnou) část kódu zarovnávání. Do funkcí vstupují dva objekty třídy `FASTAItem` obsahující zarovnávané sekvence. Před samotným zarovnáváním se porovnají délky sekvencí, aby byla delší vždy na prvním místě. Následně jsou obě sekvence zapsány pomocí `java.io.PrintWriter` do dvou souborů *f1.fa* a *f2.fa*. Po zapsání souborů se statickou funkcí `getRuntime` třídy `Runtime` získá objekt asociovaný s prostředím běžící Java aplikace. Funkcí `exec` takového objektu je vytvořen a spuštěn samostatný proces třídy `Process` s externí aplikací. Na jeho ukončení se čeká metodou `waitFor` a potom se pomocí metody `exitValue` získá návratová hodnota. Pokud proces skončil s chybou, vyhodí funkce výjimku, jinak ze standardního výstupu procesu přečtě hodnotu zarovnání sekvencí. Tímto způsobem se dostane výstupní hodnota z externí aplikace do FitClustu.

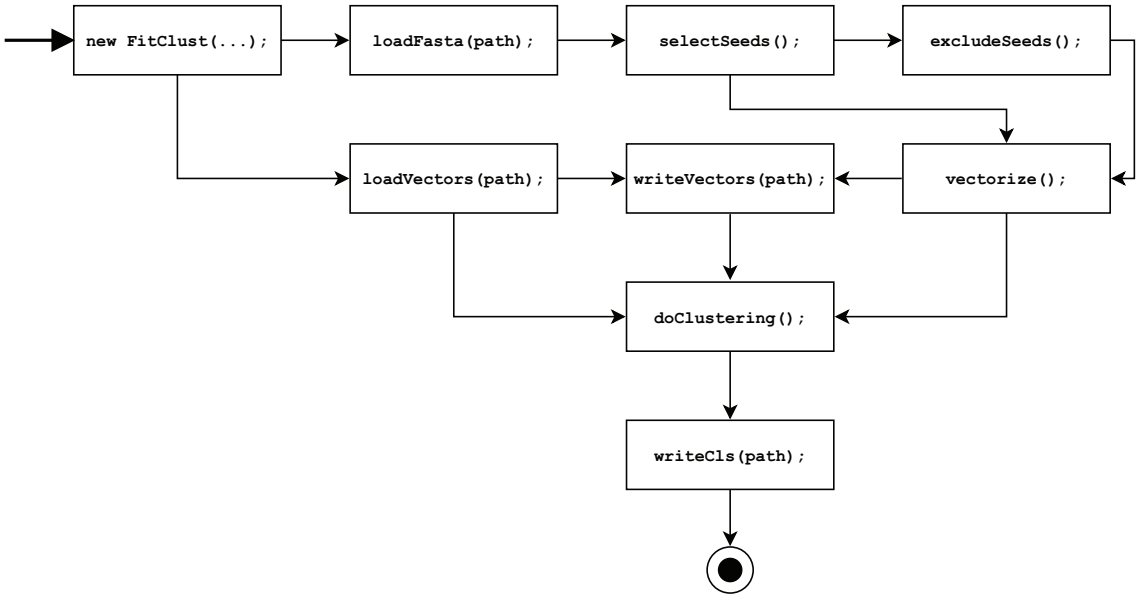
`FitClust` obsahuje hlavní jádro nástroje, které spojuje všechny výše uvedené implementované části do jednoho funkčního celku. Vytvořením instance této třídy lze přistupovat k dostupným funkcím nástroje. Konstruktor přebírá 7 parametrů používaných dále v jednolivých funkcích:

- `threshold` – určuje práh podobnosti semínek pro vektorizaci sekvencí. Na základě tohoto prahu se vyloučí podobná si semínka, vždy to kratší z nich.
- `alignment` – řetězec udávající použitou metodu zarovnání sekvencí. Volit lze z FASTA a BLAST.
- `clustering` – řetězec s názvem shlukovacího algoritmu. Na výběr jsou možnosti DBSCAN a DENCLUE.
- `param` – parametr σ nebo ε, záleží na zadaném shlukovacím algoritmu.
- `number` – parametr ξ nebo $minPts$, opět záleží na zvoleném algoritmu.
- `maxIterations` – maximální počet iterací metody hill-climbing algoritmu DENCLUE.
- `delta` – parametr stoupání metody hill-climbing.

Vytvoření objektu třídy `FitClust` a volání jeho jednotlivých veřejných funkcí se zapisuje do spouštějící hlavní třídy `Main`. Možnosti pořadí volání funkcí z důvodu návaznosti přitomnosti dat jsou zobrazeny na diagramu 6.1. Které funkce budou použity záleží na požadovaných operacích:

- `loadFasta` – načtení sekvencí z FASTA souboru,
- `writeFasta` – zápis seřazených sekvencí do souboru,
- `selectSeeds` – výběr semínek pro vektorizaci,
- `excludeSeeds` – vyloučení podobných si semínek,

- `vectorize` – vektorizace sekvencí,
- `loadVectors` – načtení vektorů pro shlukování,
- `writeVectors` – zápis načtených/vypočítaných vektorů do souboru,
- `doClustering` – provedení shlukování nad vektory,
- `writeCls` – zápis výsledných shluků do souboru.



Obrázek 6.1: Diagram možností pořadí volání funkcí objektu FitClust. Funkci `writeFasta` je možné volat kdekoliv po funkci `loadFasta`.

Každý shlukovací algoritmus má v této třídě vlastní privátní metodu. Pro implementované algoritmy DBSCAN a DENCLUE tedy existují metody `doDBSCAN` a `doDENCLUE`. Další privátní funkcí je `align` pro zarovnání sekvencí, kterou volají metody `excludeSeeds` a `vectorize`.

Kapitola 7

Experimentální ohodnocení

Kapitola o testování je věnována popisu vybraných testovacích dat, způsobu testování existujících algoritmů a způsobu porovnání získaných výsledků s jejich vyhodnocením.

7.1 Výběr testovací sady sekvencí

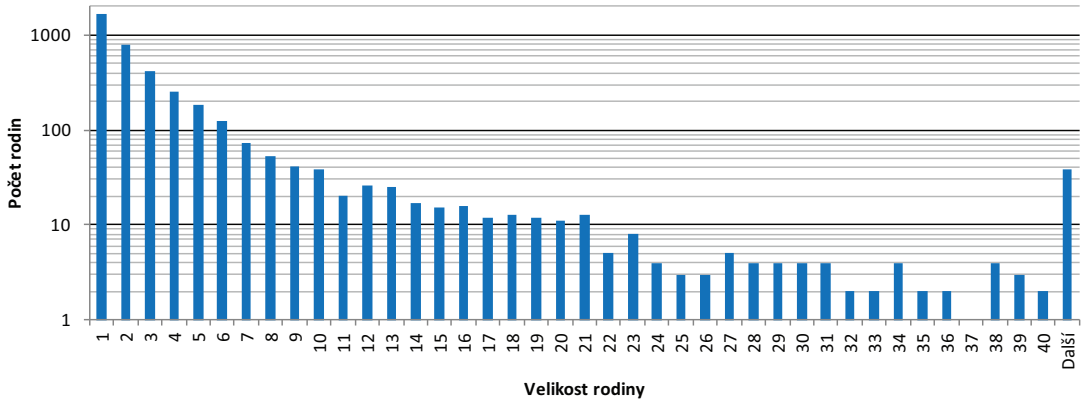
Proteinové databáze, popsané v kapitole 3, obsahují velké množství sekvencí, které jsou rozděleny do různých hierarchických struktur. Pro účel testování nástrojů je nejvhodnější vybrat databázi s hierarchií, která nejvíce odpovídá shlukům na různých úrovních podobnosti. Takové rozdělení se nachází v databázích SCOP (3.1), SCOPe (3.3) a CATH (3.5).

Další požadavek na testovací sadu je, aby obsahovala na jednotlivých úrovních proteiny, které spolu opravdu souvisejí a mají dostatečně podobné vlastnosti. Do jisté míry tento požadavek splňují všechny čtyři výše zmíněné databáze. Nejlepší volbou z nich bude SCOP, protože proteiny zařazené do jednotlivých skupin podle vlastností jsou ověřeny manuálně, a lze ji tedy považovat za referenční. Ostatní databáze byly vytvořeny (alespoň částečně) pomocí automatických algoritmů a nemusí být natolik přesné.

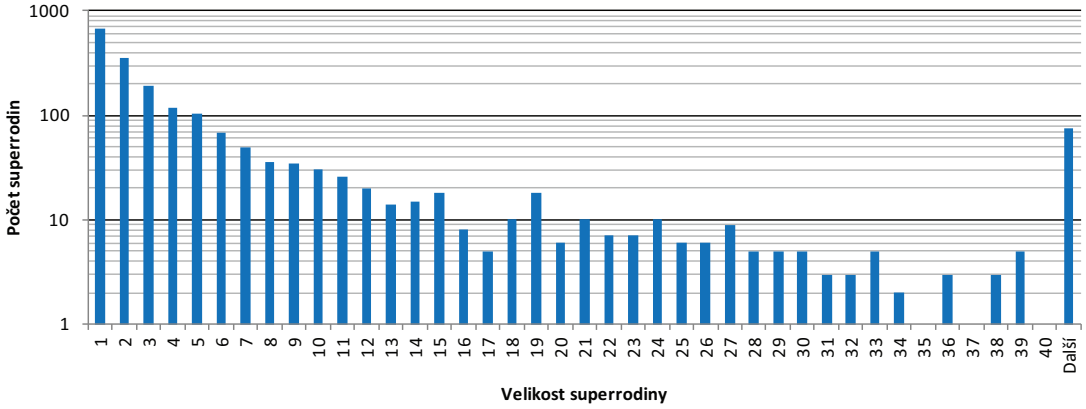
Kompletní SCOP 1.75 zahrnuje 33 352 sekvencí, ale pro testování bude dostatečný i menší počet. Na webových stránkách databáze SCOP lze získat její podmnožiny obsahující sekvence s identitou pod určitý práh. Jako vhodná byla vybrána podmnožina s identitou pod 95 %, která má 16 712 sekvencí. Nejkratší (tři) sekvence mají délku 21 aminokyselin, (jedna) nejdelší je dlouhá 1 504 aminokyselin. Nejčastější délku 107 aminokyselin má 173 sekvencí. Oproti kompletní databázi v ní chybí pouze jedna superrodina s jednou její (pod)rodinou, což by nemělo mít na výsledky testů žádný zásadní vliv.

Počty sekvencí v jednotlivých rodinách a superrodinách se dosti liší. Největší (jedna) rodina obsahuje celkem 667 sekvencí, nejmenší jsou rodiny s jednou sekvencí, kterých je 1 667. Četnosti velikostí rodin ukazuje histogram 7.1. Největší (jedna) superrodina má 920 sekvencí, nejmenší superrodiny zahrnují také pouze jednu sekvenci. Takových superrodin je 672. Histogram 7.2 zobrazuje četnosti velikostí superrodin. Oba grafy mají pro lepší názornost logaritmické měřítko o základu 10.

Testování bude probíhat především na úrovni proteinových superrodin a jejich rodin (představující shluky), protože třídy a tvary seskupují dohromady sekvence s podobností, která je založena hlavně na sekundární struktuře proteinů.



Obrázek 7.1: Histogram velikostí rodin databáze SCOP 1.75.



Obrázek 7.2: Histogram velikostí superrodin databáze SCOP 1.75.

7.2 Technika porovnání sad shluků

Porovnání výsledků shlukování bude stěžejní částí testování. Aby bylo možné porovnat dva výsledky shlukování stejné datové sady různými algoritmy, popř. stejným algoritmem (např. s jiným pořadím vstupních sekvencí) mezi sebou, je nutné zavést nějakou míru jejich podobnosti.

Shluk je množina objektů, které si jsou podobné, ale zároveň se liší od objektů v ostatních shlucích. Pro účel této práce se budou porovnávat algoritmy shlukování, které produkují exkluzivní shluky. Tedy takové, které se nemohou překrývat.

Pro výpočet podobnosti dvou sad shluků definujme [25]: Nechť $C = \{C_1, C_2, \ldots, C_m\}$ a $D = \{D_1, D_2, \ldots, D_n\}$ jsou výsledky dvou shlukování na stejné datové sadě. Předpokládáme, že C i D jsou exkluzivní shlukování, kdy jsou shluky vzájemně disjunktní, tj. každý vzorek z datové sady patří právě do jednoho shluku. Pak podobnostní matice pro C a D je

matice $S_{C,D}$ s rozměry $m \times n$ (7.1),

$$S_{C,D} = \begin{bmatrix} S_{11} & S_{12} & \cdots & S_{1j} & \cdots & S_{1n} \\ \vdots & \vdots & & \vdots & & \vdots \\ S_{i1} & S_{i2} & \cdots & S_{ij} & \cdots & S_{in} \\ \vdots & \vdots & & \vdots & & \vdots \\ S_{m1} & S_{m2} & \cdots & S_{mj} & \cdots & S_{mn} \end{bmatrix} \tag{7.1}$$

kde $S_{ij} = p/q$ (Jaccardův koeficient podobnosti[1]), p je velikost průniku a q je velikost sjednocení dvou shluků C_i a D_j. Podobnost dvou výsledků shlukování C a D lze pak definovat vztahem (7.2) [25].

$$Podobnost(C, D) = \sum_{i \leq m, j \leq n} S_{ij}/max(m, n). \tag{7.2}$$

Například: Nechť $C_1 = \{1, 2, 3, 4\}$, $C_2 = \{5, 6, 7, 8\}$ a $D_1 = \{1, 2\}$, $D_2 = \{3, 4\}$, $D_3 = \{5, 6\}$, $D_4 = \{7, 8\}$, tedy $m = 2$ a $n = 4$. Potom podobnost mezi shlukováními C a D je dána maticí $S_{C,D}$ v tabulce 7.1.

Shluk	D_1	D_2	D_3	D_4
C_1	2/4	2/4	0/6	0/6
C_2	0/6	0/6	2/4	2/4

Tabulka 7.1: Matice $S_{C,D}$ pro příklad výpočtu podobnosti dvou shlukování [25].

Pro buňku C_1D_1 platí, že $p = |C_1 \cap D_1| = |\{1, 2\}| = 2$ a $q = |C_1 \cup D_1| = |\{1, 2, 3, 4\}| = 4$. Potom $C_1D_1 = p/q = 2/4$. Stejně jsou vypočítány i ostatní buňky matice. Takže podobnost mezi sadami shluků C a D bude v tomto případě $Podobnost(C, D) = (2/4 + 2/4 + 0/6 + 0/6 + 0/6 + 0/6 + 2/4 + 2/4)/4 = 0,5$.

Jednoduše lze ukázat, že $0 < Podobnost(C, D) \leq 1$ a $Podobnost(C, D) = 1$ pro dvě identická shlukování, kdy je podobnostní matice čtvercová. Tato míra podobnosti se dá aplikovat pouze na shlukování konečné sady vzorků do konečného počtu disjunktních (nebo nepřekrývajících se) shluků [25].

Výstupní sady shluků sekvencí z testovaných algoritmů budou porovnávány pomocí této techniky na míru vzájemné podobnosti oproti referenční sadě shluků popsané v předchozí podkapitole 7.1.

7.3 Výsledky algoritmů

Vstupem všech testovaných algoritmů byla testovací sada sekvencí (podkapitola 7.1) seřazená sestupně podle jejich délky (od nejdelší po nejkratší), na základě doporučení u standardních algoritmů (podkapitola 2.1).

7.3.1 UCLUST

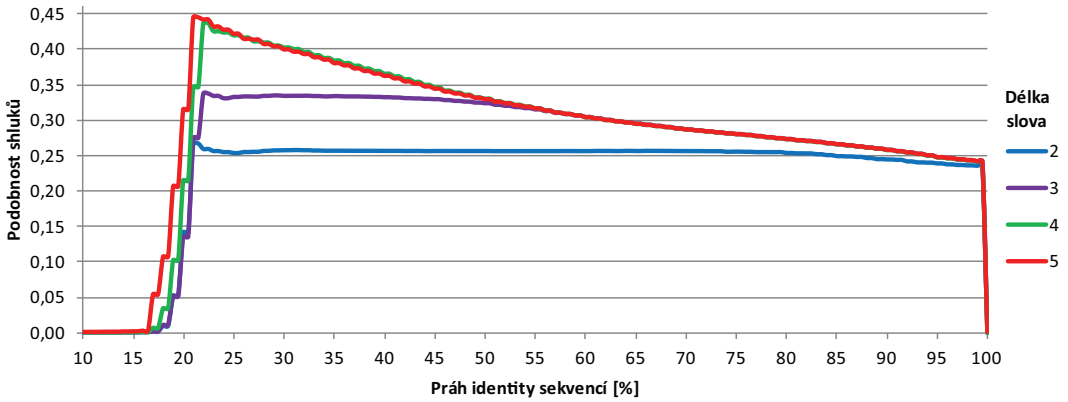
Testy algoritmu UCLUST proběhly pomocí stejnojmenného programu ve verzi 1.2.22q[2]. Pro shlukování testovací sady byl spouštěn s kombinacemi parametrů, které jsou uvedeny v tabulce 7.2. Celkově se jednalo o 724 různých kombinací vstupních parametrů.

[1] Více informací: http://people.revoledu.com/kardi/tutorial/Similarity/Jaccard.html.
[2] K dispozici na: http://www.drive5.com/uclust/downloads1_2_22q.html.

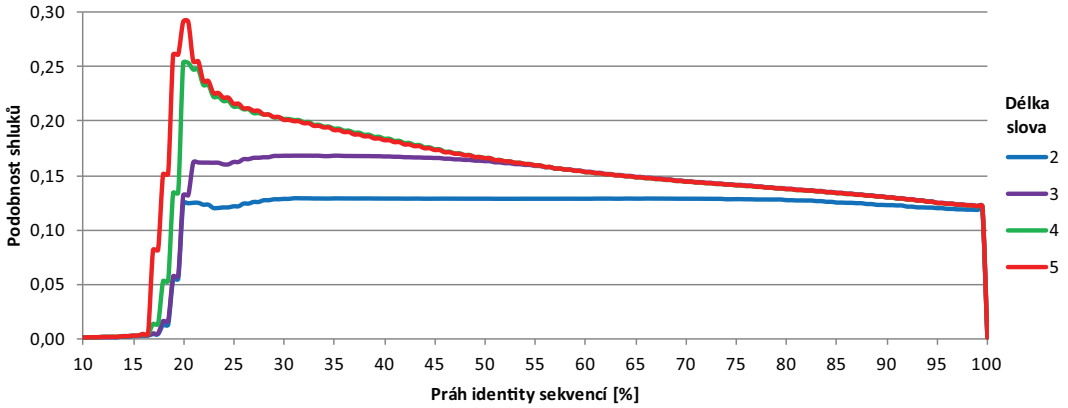
Identita [%]		Délka slova	Počet testů
Rozsah	**Krok**		
10,0 – 100,0	0,5	2–5	724
Celkem			**724**

Tabulka 7.2: Kombinace vstupních parametrů programu UCLUST.

Soubory s výslednými shluky ze všech běhů programu byly následně převedeny na jednotný formát (6.1), aby je bylo možné vzájemně porovnat na podobnost technikou uvedenou v podkapitole 7.2. Porovnávání vůči referenčním shlukům bylo provedeno zvlášť na úrovni rodin a superrodin. Získané hodnoty podobností pro rodiny jsou vyneseny v grafu 7.3. Podobnosti se superrodinami lze vidět v grafu 7.4.



Obrázek 7.3: UCLUST: Hodnoty podobností shluků na úrovni rodin.



Obrázek 7.4: UCLUST: Hodnoty podobností shluků na úrovni superrodin.

Výsledky pro různá nastavení délky slova se mezi sebou dosti liší na itervalu 10,0–50,0 % prahové identity. Od prahu 50,0 % je rozdíl pouze při délce slova 2. Shluky získáné algoritmem UCLUST se těm referenčním nejvíce podobaly s nastavením parametrů, které jsou uvedeny v tabulce 7.3.

Úroveň porovnání	Práh identity sekvencí [%]	Délka slova	Podobnost s referencí
Rodina	21,0	5	0,445529
Superrodina	20,0	5	0,291428

Tabulka 7.3: Nejlepší dosažené výsledky algoritmem UCLUST.

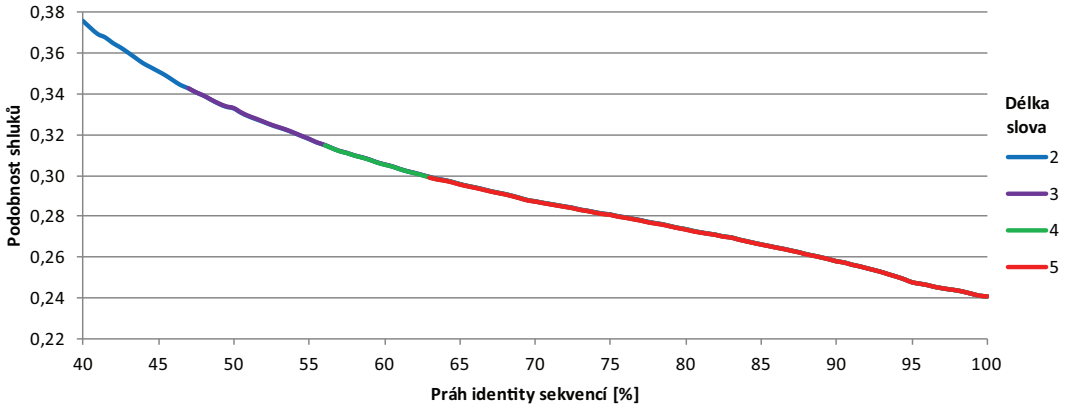
7.3.2 CD-HIT

Experimentování s algoritmem CD-HIT probíhalo programem stejného jména verze 4.6.1[3]. Kombinace vstupních parametrů, se kterými byl spouštěn, jsou shrnuty tabulkou 7.4. Bylo provedeno celkem 784 běhů s různými parametry. Oproti předchozímu UCLUST algoritmu má CD-HIT možnost volby přiřazení aktuálně zpracovávané sekvence k nejbližšímu z více vyhovujících centroidů.

Identita [%]		Délka slova	Centroid sekvence	Počet testů
Rozsah	Krok			
40,0 – 46,5	0,5	2	první/nejbližší	28
47,0 – 55,5		2–3		72
56,0 – 62,5		2–4		84
63,0 – 100,0		2–5		600
Celkem				**784**

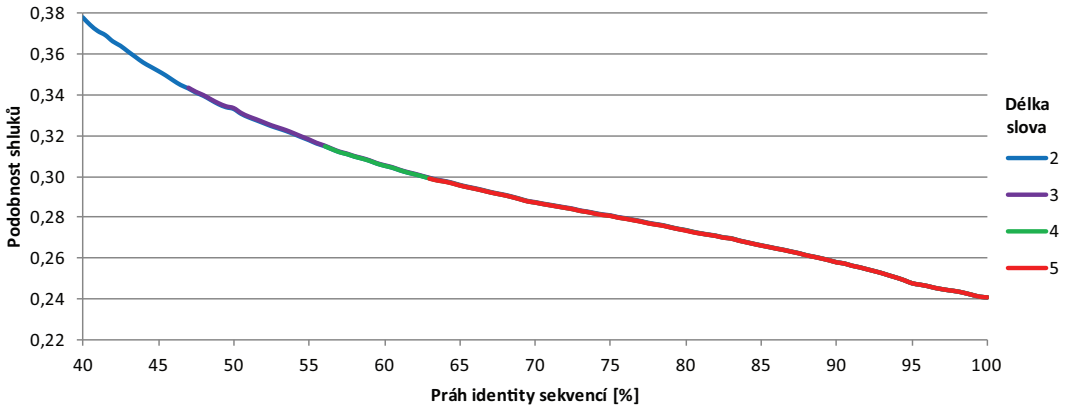
Tabulka 7.4: Kombinace vstupních parametrů programu CD-HIT.

Pro porovnání výsledků shlukování z algoritmu CD-HIT pomocí techniky z podkapitoly 7.2 bylo také nutné výstupní soubory převést na jednotný formát (6.1). Porovnávání s referenčními shluky proběhlo opět na úrovni rodin a superrodin zvlášť. Vypočítané hodnoty podobností rodin jsou zobrazeny v grafech 7.5 (první vyhovující centroid) a 7.6 (nejbližší vyhovující centroid). Podobnosti superrodin zobrazují grafy 7.7 (první vyhovující centroid) a 7.8 (nejbližší vyhovující centroid).

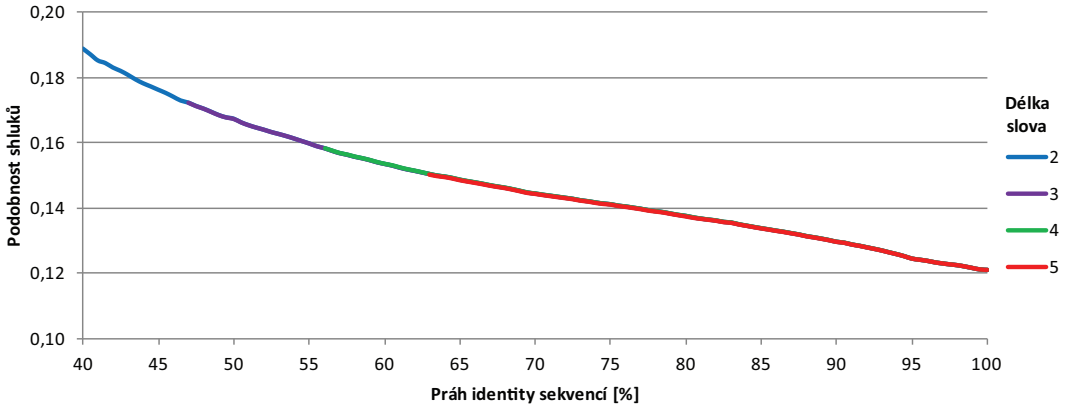


Obrázek 7.5: CD-HIT: Hodnoty podobností shluků na úrovni rodin při zařazení k prvnímu vyhovujícímu centroidu.

[3]Lze stáhnout z: http://weizhongli-lab.org/cd-hit/download.php.



Obrázek 7.6: CD-HIT: Hodnoty podobností shluků na úrovni rodin při zařazení k nejbližšímu z více vyhovujících centroidů.

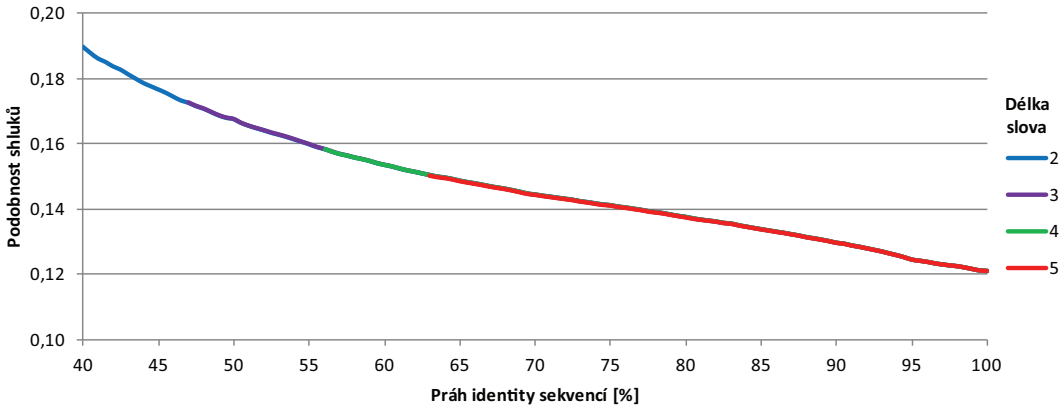


Obrázek 7.7: CD-HIT: Hodnoty podobností shluků na úrovni superrodin při zařazení k prvnímu vyhovujícímu centroidu.

I když algoritmus pracoval s různými délkami slova, jejich výsledné hodnoty se na celém testovaném intervalu indentity vzájemně téměř neliší a jednotlivé křivky pro délky slov se na všech grafech víceméně překrývají. Mírně lepší výsledky dává algoritmus při zařazování sekvencí k nejbližšímu vyhovujícímu centroidu. Největší podobnosti mezi referencemi a výstupy algoritmu CD-HIT byly dosaženy s nastavením hodnot parametrů uvedených v tabulce 7.5.

Úroveň porovnání	**Práh identity sekvencí [%]**	**Délka slova**	**Centroid sekvencí**	**Podobnost s referencí**
Rodina	40,0	2	nejbližší	0,378325
Superrodina	40,0	2	nejbližší	0,189955

Tabulka 7.5: Nejlepší dosažené výsledky algoritmem CD-HIT.



Obrázek 7.8: CD-HIT: Hodnoty podobností shluků na úrovni superrodin při zařazení k nejbližšímu z více vyhovujících centroidů.

7.3.3 Implementovaný nástroj

Při testování implementovaného nástroje nastal problém s nedostatkem operační paměti pro algoritmus DBSCAN a bylo nutné změnit vstupní testovací sadu. Z původní kompletní databáze SCOP 1.75 (7.1) s 16 712 sekvencemi bylo vybráno pět nějvětších superrodin. Testy probíhaly na každé z nich zvlášť a také na všech pěti rodinách jako celku. Jejich statistiky jsou uvedeny v tabulce 7.6. Histogram velikostí rodin tohoto podvýběru je na obrázku 7.9.

Superrodina	Počet rodin	Počet sekvencí
a.4.5	84	220
b.1.1	4	920
c.2.1	12	306
c.37.1	24	421
c.47.1	23	178
Celek	147	2045

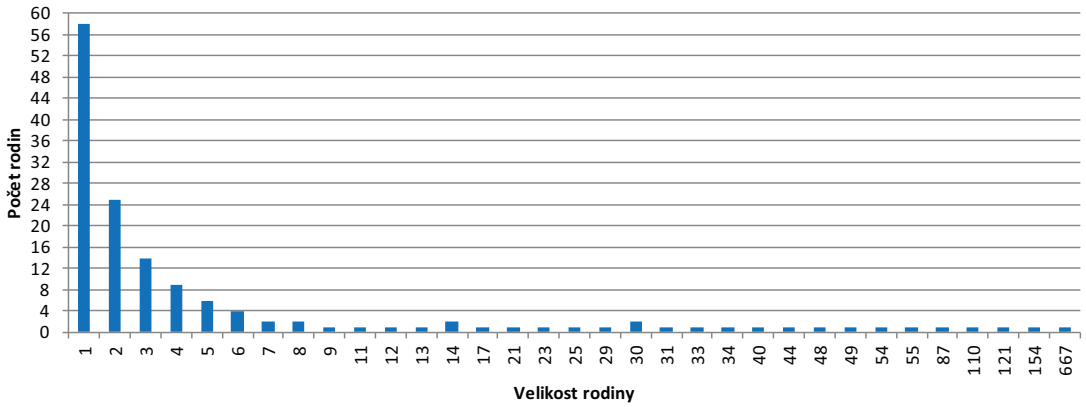
Tabulka 7.6: Statistiky pěti největších superrodin databáze SCOP 1.75.

Testované kombinace algoritmů

FitClust umožňuje zarovnání sekvencí algoritmy BLAST a FASTA. Dále umí shlukování DBSCAN a DENCLUE. K testování bylo vybráno pět největších superrodin a jejich celek, tedy celkem šest vstupních datových sad. Dohromady tyto možnosti tvoří 24 různých kombinací:

$$\{BLAST, FASTA\} \times \{DBSCAN, DENCLUE\} \times \{a.4.5, b.1.1, c.2.1, c.37.1, c.47.1, celek\}.$$

Pro každou kombinaci byla experimentálně určena dolní a horní hranice parametru ε (DBSCAN), resp. σ (DENCLUE). Dolní hranice určuje nejnižší hodnotu daného parametru, kdy je každá sekvence ve svém vlastním shluku. Horní hranice naopak určuje hodnotu parametru, kdy jsou všechny sekvence zařazeny do shluku jednoho. Rozsahy prezentuje tabulka 7.7. Mezi těmito hranicemi potom byly kombinace spuštěny s krokem 1.



Obrázek 7.9: Histogram velikostí rodin pěti největších superrodin databáze SCOP 1.75.

Superrodina	**DBSCAN (ε)**		**DENCLUE (σ)**	
	BLAST	**FASTA**	**BLAST**	**FASTA**
a.4.5	160–255	35–90	50–250	30–90
b.1.1	20–325	15–170	10–320	5–165
c.2.1	105–215	40–160	70–210	4–160
c.37.1	80–210	65–100	30–205	20–95
c.47.1	170–235	50–85	35–235	15–80
Celek	80–225	65–145	8–220	8–140

Tabulka 7.7: Rozsahy parametrů ε a σ pro kombinace algoritmů.

7.4 Porovnání algoritmů

Na nových testovacích datech byly opět spuštěny UCLUST a CD-HIT, včetně stejných rozsahů jejich parametrů, které jsou uvedeny v tabulkách 7.2 a 7.4. Nejlepší dosažené výsledky každého algoritmu na nových datový sadách obsahuje tabulka 7.8. Z výsledků je patrné, že implementovaný nástroj v mnoha případech dosahuje lepšího shlukování než hojně používaný CD-HIT. Naopak stále výrazně zaostává za UCLUSTem, který dokáže pracovat na nízkých identitách.

Superrodina	**CD-HIT**	**UCLUST**	**FitClust**			
			DBSCAN		**DENCLUE**	
			BLAST	**FASTA**	**BLAST**	**FASTA**
a.4.5	0,480	0,570	0,425	0,448	0,431	0,449
b.1.1	0,029	0,492	0,277	0,253	0,277	0,256
c.2.1	0,062	0,379	0,149	0,138	0,149	0,138
c.37.1	0,104	0,286	0,129	0,124	0,124	0,111
c.47.1	0,223	0,368	0,317	0,290	0,273	0,283
Celek	0,175	0,307	0,089	0,082	0,174	0,072

Tabulka 7.8: Porovnání nejlepších dosažených výsledků jednotlivými algoritmy.

Kapitola 8

Závěr

Cílem této diplomové práce bylo navrhnout a implementovat shlukovací nástroj, který by dosahoval lepších výsledků, než ty existující. Navíc využívá pro tento obor netradiční přístupy. Dosažené výsledky na reálných a opravdu referenčních datech bych zhodnotil jako uspokojivé, protože se podařilo ukázat, že shlukování proteinových sekvencí, které by bylo založené na hustotě shluků, je jistou cestou k lepším výsledkům oproti tradičním shlukovacím algoritmům.

Současná implementace používá základní verze vybraných algoritmů. Bylo by například vhodné využít rozšířené analyzování potenciálních semínek u algoritmu mBed, tedy heuristiku používající středové objekty nebo středové skupiny. Dále by se dal rozvinout algoritmus hill-climbing heuristikou zajišťující rychlejší dosažení maxima s menším počtem iterací. Aktuálně implementované přístupy rozhodně nabízejí velký poteciál k dalšímu rozvoji projektu, aby dosahoval ještě mnohem lepších výsledků, než dosud.

Literatura

[1] Andreeva, A.; Howorth, D.; Chothia, C.; aj.: SCOP2 prototype: a new approach to protein structure mining. *Nucleic Acids Research*, ročník 42, č. D1, leden 2014: s. D310–314, iSSN: 1362-4962, [online]. DOI: 10.1093/nar/gkt1242.

[2] Blackshields, G.; aj.: Sequence embedding for fast construction of guide trees for multiple sequence alignment. *Algorithms for Molecular Biology*, ročník 5, č. 1, květen 2010: str. 21, [online]. DOI: 10.1186/1748-7188-5-21.

[3] Edgar, R. C.: UCLUST algorithm. *Drive5: Bioinformatics software and services*, [online]. Dostupné z: http://drive5.com/usearch/manual/uclust_algo.html.

[4] Edgar, R. C.: Search and clustering orders of magnitude faster than BLAST. *Bioinformatics*, ročník 26, č. 19, 2010: s. 2460–2461, [online]. DOI: 10.1093/bioinformatics/btq461.

[5] Enright, A. J.; Van Dongen, S.; Ouzounis, C. A.: An efficient algorithm for large-scale detection of protein families. *Nucleic Acids Research*, ročník 30, č. 7, duben 2002: s. 1575–1584, [online]. DOI: 10.1093/nar/30.7.1575.

[6] Ester, M.; aj.: A density-based algorithm for discovering clusters in large spatial databases with noise. *Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining*, 1996: s. 226–231, [online]. DOI: 10.1.1.121.9220.

[7] Fox, N. K.; Brenner, S. E.; Chandonia, J.-M.: SCOPe: Structural Classification of Proteins–extended, integrating SCOP and ASTRAL data and classification of new structures. *Nucleic Acids Research*, ročník 42, č. D1, leden 2014: s. D304–D309, iSSN: 1362-4962, [online]. DOI: 10.1093/nar/gkt1240.

[8] Hauser, M.; Mayer, C. E.; Söding, J.: kClust: fast and sensitive clustering of large protein sequence databases. *BMC Bioinformatics*, ročník 14, č. 1, srpen 2013: str. 248, [online]. DOI: 10.1186/1471-2105-14-248.

[9] Hinneburg, A.; Gabriel, H.-H.: DENCLUE 2.0: Fast Clustering based on Kernel Density Estimation. *Advances in Intelligent Data Analysis VII*, 2007: s. 70–80, [online]. DOI: 10.1007/978-3-540-74825-0_7.

[10] Hinneburg, A.; Keim, D. A.: An Efficient Approach to Clustering in Large Multimedia Databases with Noise. 1998: s. 58–65, [online]. DOI: 10.1.1.44.3961.

[11] Hobohm, U.; Sander, C.: Enlarged representative set of protein structures. *Protein Science*, ročník 3, č. 3, březen 1994: s. 522–524, [online]. DOI: 10.1002/pro.5560030317.

[12] Hobohm, U.; aj.: Selection of representative protein data sets. *Protein Science*, ročník 1, č. 3, březen 1992: s. 409–417, [online]. DOI: 10.1002/pro.5560010313.

[13] Huang, Y.; aj.: CD-HIT Suite: a web server for clustering and comparing biological sequences. *Bioinformatics*, ročník 26, č. 5, leden 2010: s. 680–682, [online]. DOI: 10.1093/bioinformatics/btq003.

[14] Li, W.; Godzik, A.: Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences. *Bioinformatics*, ročník 22, č. 13, leden 2006: s. 1658–1659, [online]. DOI: 10.1093/bioinformatics/btl158.

[15] Li, W.; Jaroszewski, L.; Godzik, A.: Clustering of highly homologous sequences to reduce the size of large protein databases. *Bioinformatics*, ročník 17, č. 3, leden 2001: s. 282–283, [online]. DOI: 10.1093/bioinformatics/17.3.282.

[16] Li, W.; Jaroszewski, L.; Godzik, A.: Tolerating some redundancy significantly speeds up clustering of large protein databases. *Bioinformatics*, ročník 18, č. 1, leden 2002: s. 77–82, [online]. DOI: 10.1093/bioinformatics/18.1.77.

[17] Li, W.; aj.: CD-HIT User's Guide. *Research Group of Weizhong Li*, [online]. Dostupné z: http://weizhong-lab.ucsd.edu/cd-hit/wiki.

[18] Loewenstein, Y.; aj.: Efficient algorithms for accurate hierarchical clustering of huge datasets: tackling the entire protein space. *Bioinformatics*, ročník 24, č. 13, červenec 2008: s. i41–i49, [online]. DOI: 10.1093/bioinformatics/btn174.

[19] Mayer, C. E.: *Fast Method for Sequence Comparison and Application to Database Clustering*. diplomová práce, Eberhard Karls Universität Tübingen, 2007.

[20] Murzin, A. G.; Brenner, S. E.; Hubbard, T.; aj.: SCOP: A structural classification of proteins database for the investigation of sequences and structures. *Journal of Molecular Biology*, ročník 247, č. 4, duben 1995: s. 536–540, iSSN: 0022-2836, [online]. DOI: 10.1016/S0022-2836(05)80134-2.

[21] Pevsner, J.: *Bioinformatics and Functional Genomics*. Hoboken, New Jersey: John Wiley & Sons, druhé vydání, 2009, iSBN: 978-0-470-08585-1.

[22] Remmert, M.: *Fast, sensitive protein sequence searches using iterative pairwise comparison of hidden Markov models*. disertační práce, Ludwig-Maxmilians-Universität München, 2011, [online]. Dostupné z: http://nbn-resolving.de/urn:nbn:de:bvb:19-138555.

[23] Sillitoe, I.; Lewis, T. E.; Cuff, A.; aj.: CATH: comprehensive structural and functional annotations for genome sequences. *Nucleic Acids Research*, ročník 43, č. D1, únor 2015: s. D376–D381, iSSN: 1362-4962, [online]. DOI: 10.1093/nar/gku947.

[24] Sonnhammer, E. L.; Eddy, S. R.; Durbin, R.: Pfam: A comprehensive database of protein domain families based on seed alignments. *Proteins: Structure, Function, and Bioinformatics*, ročník 28, č. 3, červenec 1997: s. 405–420, iSSN: 1097-0134, [online]. DOI: 10.1002/(SICI)1097-0134(199707)28:3<405::AID-PROT10>3.0.CO;2-L.

[25] Torres, G. J.; Basnet, R. B.; Sung, A. H.; aj.: A Similarity Measure for Clustering and its Applications. *World Academy of Science, Engineering and Technology*, ročník 2, č. 5, 2008: s. 1097–1103, iSSN: 1307-6892, [online]. Dostupné z: http://waset.org/publications/9316.

Dodatek A

Formální definice a lemmata

Zde jsou uvedeny doplňující detaily k podkapitole 2.3 o shlukování na základě hustoty. Především formální definice a lemmata, která jsou odkazována z textu diplomové práce.

A.1 DBSCAN

Definice A.1.1 (ε-okolí bodu)**.** *ε-okolí bodu P, označované $N_\varepsilon(P)$, je definováno $N_\varepsilon(P) = \{Q \in D \mid dist(P,Q) \leq \varepsilon\}$.*

Definice A.1.2 (Přímá dosažitelnost na základě hustoty)**.** *Bod P je přímo dosažitelný na základě hustoty z bodu Q vzhledem k ε a* MinPts *pokud:*

1. *$P \in N_\varepsilon(Q)$ a zároveň*
2. *$|N_\varepsilon(Q)| \geq MinPts$ (podmínka jádra).*

Zřejmě je přímá dosažitelnost na základě hustoty symetrická pro dvě jádra. Obecně není symetrická pro jedno jádro a jeden hraniční bod. Obrázek 2.15 zobrazuje případ asymetrie.

Definice A.1.3 (Dosažitelnost na základě hustoty)**.** *Bod P je dosažitelný na základě hustoty z bodu Q vzhledem k ε a* MinPts*, pokud existuje řetěz bodů $P_1, P_2, \ldots, P_n$, $P_1 = Q$, $P_n = P$, takový, že P_{i+1} je přímo dosažitelný na základě hustoty z P_i.*

Dosažitelnost na základě hustoty je kanonické rozšíření přímé dosažitelnosti na základě hustoty. Tento vztah je tranzitivní, ale není symetrický. Obrázek 2.16 zobrazuje vztahy několika vzorových bodů, konkrétně případ asymetrie. I když není obecně symetrický, je zřejmé, že dosažitelnost na základě hustoty je symetrická pro jádra.

Dva hraniční body stejného shluku C nejsou pravděpodobně navzájem dosažitelné na základě hustoty, protože podmínka jádra nemusí platit pro oba. Nicméně musí existovat jádro v C, pro které jsou oba hraniční body shluku C dosažitelné na základě hustoty. Proto je zaveden pojem spojení na základě hustoty, který pokrývá tento vztah hraničních bodů.

Definice A.1.4 (Spojení na základě hustoty)**.** *Bod P je spojený na základě hustoty s bodem Q vzhledem k ε a* MinPts*, pokud existuje bod O takový, že oba body P a Q jsou dosažitelné na základě hustoty z bodu O vzhledem k ε a* MinPts*.*

Spojení na základě hustoty je symetrický vztah. Pro body dosažitelné na základě hustoty je vztah spojení na základě hustoty také reflexivní (obrázek 2.16).

Definice A.1.5 (Shluk)**.** *Nechť* D *je databáze bodů. Shluk C vzhledem k* ε *a* MinPts *je neprázdná podmnožina* D *splňující tyto podmínky:*

1. $\forall$ *P, Q: Pokud* $P \in C$ *a Q je dosažitelné na základě hustoty z P vzhledem k* ε *a* MinPts*, pak* $Q \in C$ *(Maximalita).*
2. $\forall$ *P,* $Q \in C$*: P je spojené na základě hustoty s Q vzhledem k* ε *a* MinPts *(Konektivita).*

Definice A.1.6 (Šum)**.** *Nechť* $C_1, \ldots, C_k$ *jsou shluky databáze* D *vzhledem k parametrům* ε_i *a* $MinPts_i$*,* $i = 1, \ldots, k$*. Pak definujeme šum jako množinu bodů v databázi* D *nepatřících do žádného shluku* C_i*, tj. šum* $= \{P \in D \,|\, \forall i : P \notin C_i\}$*.*

Shluk C vzhledem k ε a *MinPts* obsahuje alespoň *MinPts* bodů protože:

1. C obsahuje jeden bod P, P musí být spojené na základě hustoty samo k sobě přes nějaký bod O (který může být totožný s P).
2. Tedy, alespoň O musí splňovat podmínku jádra, a proto ε-okolí bodu O obsahuje alespoň *MinPts* bodů.

Následující lemmata jsou důležitá pro ověření správnosti algoritmu DBSCAN a vyjadřují následující:

Vzhledem k parametrům ε a *MinPts* lze nalézt shluk během dvou kroků. Nejprve se vybere z databáze bod splňující podmínku jádra jako semínko. Poté se získají všechny body, které jsou dosažitelné na základě hustoty z tohoto semínka, jako shluk semínka.

Lemma A.1.1. *Nechť P je bod v D a* $|N_\varepsilon(P)| \geq MinPts$*. Pak množina* $O = \{o \,|\, o \in D \wedge o$ je dosažitelné na základě hustoty z P vzhledem k ε a MinPts$\}$ *je shluk vzhledem k* ε *a* MinPts*.*

Není zcela zřejmé, že shluk C vzhledem k ε a *MinPts* je jednoznačně určen nějakým jeho jádrem. Nicméně každý bod v C je dosažitelný na základě hustoty z jakéhokoliv jádra v C a tedy shluk C obsahuje právě body, které jsou dosažitelné na základě hustoty z jakéhokoliv jádra v C.

Lemma A.1.2. *Nechť C je shluk vzhledem k* ε *a* MinPts *a nechť P je nějaký bod v C s* $|N_\varepsilon(P)| \geq MinPts$*. Pak C je shodné s množinou* $O = \{o \,|\, o$ je dosažitelné na základě hustoty z P vzhledem k ε a MinPts$\}$*.*

A.2 DENCLUE

Definice A.2.1 (Funkce vlivu)**.** Funkce vlivu *datového objektu* $y \in F^d$ *je funkce* $f_B^y : F^d \to R_0^+$*, která je definována z hlediska základní funkce hustoty* f_B*:*

$$f_B^y(x) = f_B(x, y).$$

Definice A.2.2 (Funkce hustoty)**.** Funkce hustoty *je definována jako součet funkcí vlivu všech datových bodů. Pro N datových objektů popsaných množinou vektorů rysů* $D = \{x_1, \ldots, x_N\} \subset F^d$ *je funkce hustoty definována jako:*

$$f_B^D(x) = \sum_{i=1}^{N} f_B^{x_i}(x).$$

Funkce hustoty může být v podstatě libovolná funkce. Pro definici specifických funkcí vlivu potřebujeme vzdálenostní funkci $d : F^d \times F^d \to R_0^+$, která určuje vzdálenost dvou d-dimenzionálních vektorů rysů. Vzdálenostní funkce musí být reflexivní a symetrická. Kvůli jednoduchosti budeme předpokládat Euklidovskou vzdálenostní funkci. Definice jsou ovšem na volbě vzdálenostní funkce nezávislé. Příklady základních funkcí vlivu:

1. Obdélníková funkce vlivu:

$$f_{Square}(x, y) = \begin{cases} 0 & \text{pokud } d(x, y) > \sigma \\ 1 & \text{jinak} \end{cases}$$

2. Gaussova funkce vlivu:

$$f_{Gauss}(x, y) = e^{-\frac{d(x,y)^2}{2\sigma^2}}$$

Funkce hustoty, která vznikne z Gaussovy funkce vlivu je:

$$f^D_{Gauss}(x) = \sum_{i=1}^{N} e^{-\frac{d(x,x_i)^2}{2\sigma^2}}.$$

Definice A.2.3 (Stoupání (gradient))**.** *Stoupání funkce $f^D_B(x)$ je definováno jako:*

$$\bigtriangledown f^D_B(x) = \sum_{i=1}^{N} (x_i - x) \cdot f^{x_i}_B(x).$$

V případě Gaussovy funkce vlivu je stoupání definováno jako:

$$\bigtriangledown f^D_{Gauss}(x) = \sum_{i=1}^{N} (x_i - x) \cdot e^{-\frac{d(x,x_i)^2}{2\sigma^2}}$$

Obecně je vhodné, aby funkce vlivu byla symetrická, spojitá a derivovatelná. Definice stoupání je na těchto vlastnostech nezávislá.

Definice A.2.4 (Atraktor hustoty)**.** *Bod $x^* \in F^d$ se nazývá* atraktor hustoty *pro danou funkci vlivu tehdy a jen tehdy, pokud x^* je lokální maximum funkce hustoty f^D_B.*

Bod $x \in F^d$ je atrakován na základě hustoty *k atraktoru x^* tehdy a jen tehdy, pokud $\exists k \in N : d(x^k, x^*) \leq \epsilon$ kde:*

$$x^0 = x, x^i = x^{i-1} + \delta \cdot \frac{\bigtriangledown f^D_B(x^{i-1})}{\| \bigtriangledown f^D_B(x^{i-1}) \|}.$$

Nyní lze definovat shluky a odlehlé hodnoty. Odlehlé hodnoty jsou body, které nejsou ovlivněny „mnoha" dalšími datovými body. Je třeba mez ξ k formalizaci „mnoho".

Definice A.2.5 (Shluk definovaný středem)**.** Shluk definovaný středem *(vzhledem k σ, ξ) pro atraktor hustoty x^* je podmnožina $C \subseteq D$, kde $x \in C$ je atrakováno na základě hustoty x^* a $f^D_B(x^*) \geq \xi$. Body $x \in D$ nazýváme* odlehlé hodnoty*, pokud jsou atrakovány na základě hustoty lokálním maximem x^*_0, kde $f^D_B(x^*_0) < \xi$.*

Definice A.2.6 (Shluk libovolného tvaru)**.** Shluk libovolného tvaru *(vzhledem k σ, ξ) pro množinu atraktorů hustoty X je podmnožina $C \subseteq D$, kde:*

1. $\forall x \in C \, \exists x^* \in X : f_B^D(x^*) \geq \xi$, *x je atrakováno na základě hustoty k* x^* *a*

2. $\forall x_1^*, x_2^* \in X : \exists$ *cesta* $P \subset F^d$ *z* x_1^* *do* x_2^*, *kde* $\forall p \in P : f_B^D(p) \geq \xi$.

Definice A.2.7 (Lokální funkce hustoty)**.** *Lokální funkce hustoty* $\hat{f}^D(x)$ *je*

$$\hat{f}^D(x) = \sum_{x_1 \in near(x)} f_B^{x_1}(x).$$

Dodatek B

Manuál

Tato příloha popisuje jednotlivé funkční části diplomové práce, především způsob jejich spouštění a použití.

B.1 Pomocné skripty

Zde jsou popsány pomocné skripty práce nacházejí se v adresářích `app/java` a `app/tools` přiloženého DVD.

uc2cls

Skript v Pythonu pro převod souboru se shluky ve formátu UC, produkovaného algoritmem UCLUST, na jednotný formát CLS. Použití:

```
./uc2cls cluster.uc cluster.cls
```

clstr2cls

Algoritmus CD-HIT má výstupní shluky v souborech formátu CLSTR. Tento skript v Pythonu jej převádí na jednotný formát CLS. Použití:

```
./clstr2cls cluster.clstr cluster.cls
```

2ClsSimilarity

Porovnání dvou sad shluků ve formátu CLS na základě techniky 7.2 je možné provést tímto skriptem v Pythonu. Výstupem je desetinné číslo, které udává míru podobnosti sad, ale **NEJEDNÁ SE O PROCENTA**. Použití:

```
./2ClsSimilarity cluster1.cls cluster2.cls
```

blast

Jednořádkový skript v Bashi je „zástupcem" programu `blastp` pro potřebu BLAST zarovnání dvou sekvencí. Spouští jej třída `Alignment`. Aby mohl být spuštěň, musí být ve spouštěcím adresáři přítomny dva FASTA soubory *f1.fa* a *f2.fa*. **SKRIPT OBSAHUJE PŘÍKAZ PRO JEJICH SMAZÁNÍ.** Použití:

```
./blast
```

fasta

Stejnou funkci jako předchozí `blast` má i tento skript. Provádí FASTA zarovnání dvou FASTA sekvencí *f1.fa* a *f2.fa*. **SKRIPT OBSAHUJE PŘÍKAZ PRO JEJICH SMAZÁNÍ.** Použití:

```
./fasta
```

build

Překlad části diplomové práce napsané v Javě provádí tento skript. Vytvoří adresář `bin`, do kterého jsou umístěny přeložené soubory. Použití:

```
./build
```

run

Spouští hlavní třídu `Main` diplomové práce. Lze mu na příkazové řádce zadat až 9 parametrů `p1-p9`, které se dají použít ve funkci `main`. Použití:

```
./run [p1 p2 p3 p4 p5 p6 p7 p8 p9]
```

makedoc

Pro vytvoření dokumentace Java části diplomové práce. Spouští `javadoc` s potřebnými parametry. Dokumentace se po spuštění nachazí v adresáři `doc`. Použití:

```
./makedoc
```

B.2 FitClust

Pro spuštění shlukování nástrojem FitClust je třeba nastavit jeho parametry a do souboru *Main.java* zapsat požadované operace podle podkapitoly 6.2.5. Po tomto nastavení stačí nástroj přeložit skriptem `build` a poté spustit skriptem `run`. Příklad:

```
FitClust fc = new FitClust(
  0.0, /* Práh podobnosti semínek */
  "FASTA", /* Metoda zarovnání sekvencí */
  "DENCLUE", /* Metoda shlukování sekvencí */
  0.65, /* Parametr sigma nebo eps */
  4, /* Parametr xí nebo minPts */
  5, /* Parametr maxIterations */
  1.0 /* Parametr delta */
);

/**
 * Z následujících řádků je třeba odkomentovat ty,
 * které mají být použity nástrojem.
 */
/* Načítá FASTA záznamy ze souboru */
fc.loadFasta("scop.fa");
/* Zapisuje seřazené záznamy do souboru */
fc.writeFasta("scop_sorted.fa");
/* Vybírá semínka ze vstupních FASTA záznamů */
fc.selectSeeds();
/* Vylučuje podobná si semínka */
fc.excludeSeeds();
/* Provádí vektorizaci se semínky */
fc.vectorize();
/* Zapisuje vektory do souboru */
fc.writeVectors("vektory.csv");
/* Provádí shlukování */
fc.doClustering();
/* Zapisuje shluky do souboru */
fc.writeCls("shluky.cls");
```

Dodatek C

Obsah DVD

V diplomové práci je vloženo DVD s následující strukturou:

- `app` – složka vytvořeného nástroje
  - `data` – testovací data
  - `java` – programová část práce a některé pomocné skripty
    - `src` – zdrojové kódy nástroje
  - `programs` – externí a testované aplikace
  - `results` – výsledky testování
  - `tools` – ostatní pomocné skripty
- `thesis`
  - `pdf` - tato zpráva ve formátu PDF
  - `src` - zdrojové soubory pro vygenerování této zprávy

Dodatek D

Poznámky