VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

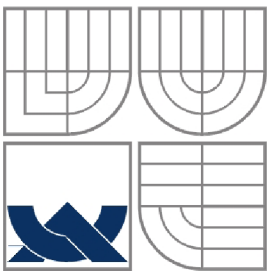
ANALÝZA GENOMICKÝCH DAT S VYUŽITÍM ALGORITMU BLAST

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

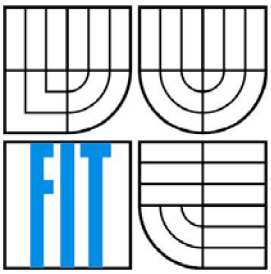
AUTOR PRÁCE
AUTHOR

MARTINA KŘÍŽOVÁ

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ANALÝZA GENOMICKÝCH DAT S VYUŽITÍM ALGORITMU BLAST

Analysis of Genomic data with assistance of algorithm blast

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE
AUTHOR

MARTINA KŘÍŽOVÁ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. PETR JAŠA

BRNO 2008

Zadání bakalářské práce/6644/2007/xkrizo00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2007/2008

Zadání bakalářské práce

Řešitel: **Křížová Martina**
Obor: Informační technologie
Téma: **Analýza genomických dat s využitím algoritmu BLAST**
Kategorie: Databáze

Pokyny:

1. Seznamte se s oblastí biologie pracující s genomickými daty, seznamte se s biologickou strukturou těchto dat a prostudujte formáty v jakých se uchovávají z pohledu informatiky.
2. Seznamte se s algoritmem BLAST implementovaným v prostředí ORACLE 10g.
3. Navrhněte aplikaci využívající tohoto algoritmu a zkuste analyzovat testovací data.
4. Zhodnoťte výsledky.

Literatura:

- Dan K. Krane, Michael L. Raymer: *Fundamental Concepts of Bioinformatics*, ISBN: 0-8053-4633-3, Benjamin Cummings 2003
- Jean-Michel Claverie, Cedric Notredame: *Bioinformatics for Dummies*, ISBN: 0-7645-1696-5, Wiley Publishing, Inc., 2003
- Jiawei Han, Micheline Kamber: *Data mining: concepts and techniques*, ISBN 1-55860-901-6, Morgan Kaufmann Publishers 2006
- Oracle Data Mining Concepts 10g Release 1 (10.1), URL http://download.oracle.com/docs/cd/B14117_01/datamine.101/b10698/toc.htm

Při obhajobě semestrální části projektu je požadováno:

- body 1 a 2

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Jaša Petr, Ing.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2007
Datum odevzdání: 14. května 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2
L.S.

doc. Ing. Jaroslav Zendulka, CSc.
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. Slečna

Jméno a příjmení: **Martina Křížová**
Id studenta: 79324
Bytem: Voříškova 403/13, 623 00 Brno
Narozena: 25. 01. 1986, Brno
(dále jen "autor")

a

2. Vysoké učení technické v Brně

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
   bakalářská práce

Název VŠKP: Analýza genomických dat s využitím algoritmu BLAST
Vedoucí/školitel VŠKP: Jaša Petr, Ing.
Ústav: Ústav informačních systémů
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:

tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☐ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

....

Nabyvatel Autor

Abstrakt

Tato bakalářská práce se zabývá analýzou genomických dat s využitím algoritmu BLAST. V teoretické části popisuje genetické informace, jejich význam a strukturu. Dále zde naleznete popis algoritmu BLAST, Smith-Waterman a Needleman-Wunsch a s algoritmem BLAST spojenou Karlin-Altschul statistiku. Tato práce také pojednává o typech databází jako prostorové, deduktivní a relační, které ukládají genomické informace. Druhá část se zabývá návrhem a implementací aplikace, která využívá zmíněné algoritmy a databázi Oracle 10g R2. Součástí práce je i testování či srovnání výstupů algoritmů.

Klíčová slova

Genetika, DNA, protein, BLAST, Smith-Waterman, Needleman-Wunsch, substituční matice BLOSUM, PAM, teorie Margaret Dayhoff, Karlin-Altschul statistiky, Oracle 10g

Abstract

This bachelor thesis describes analysis of genomic data with assistance of algorithm BLAST. In theoretic part, it describes genomes, their importance and structure. In this thesis, you can find a description of algorithm BLAST, Smith-Waterman and Needleman-Wunsch and with algorithm BLAST connected Karlin-Altschul statistics. This thesis is discussing types of databases such as spatial, deductive and relational which store genomic data. Second part is concerned with design and implementation of application which use mentioned algorithm and database Oracle 10g R2. Testing and confrontation output of algorithm is also a part of this work.

Keywords

Genetics, DNA, protein, BLAST, Smith-Waterman, Needleman-Wunsch, substitution matrix BLOSUM, PAM, theory Margaret Dayhoff, Karlin-ALtschul statistics, Oracle 10g

Citace

Křížová Martina: Analýza genomických dat s využitím algoritmu BLAST. Brno, 2008, bakalářská práce, FIT VUT v Brně.

Analýza genomických dat s využitím algoritmu BLAST

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně pod vedením Ing. Petra Jaše
Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

……
Martina Křížová
12.5.2008

Poděkování

Ráda bych poděkovala panu Ing. Petru Jašovi za vstřícné konzultace, projevenou ochotu a odbornou pomoc, kterou mi poskytl při psaní této bakalářské práce.

Obsah

Úvod

Tato bakalářská práce se zabývá genomickými daty, což jsou genetické informace na molekulární úrovni. Základními molekulami jsou DNA a protein. Analýza těchto dat nám napomáhá ke sledování genetických odchylek a mutací, což může být příčinou lidské predispozice k vadám a nemocem, kterým by se mohlo v budoucnu předejít. Stejně tak i pokrok, který je zaznamenán v klonování orgánů a dalších částí lidského těla, vedoucí ke zlepšení zdravotního stavu populace, využívá genetické informace na molekulární úrovni.

První kapitola popisuje tři základní molekuly DNA, protein a RNA, zabývá se jejich stavbou, strukturou a vznikem. V této části bych chtěla, abyste pochopili, jak spolu tyto tři molekuly souvisí a jak jsou dále DNA a protein vyjádřeny formou sekvencí písmen, které vyjadřují u DNA nukleotidy a u proteinů aminokyseliny.

Genomické informace mohou být uloženy v různých formách. Nejčastěji se k uchování genomických informací používají databáze a soubory, v tomto případě jde o speciální soubory fasta. Jak již bylo zmíněno, lze data uchovávat v databázích. Ve druhé části se můžete dočíst o databázích prostorových, deduktivních a relačních, které se právě používají k ukládaní těchto dat.

Jedním z algoritmů, který nám přispívá k analýze genomických dat, je algoritmus BLAST. Tento algoritmus porovnává vstupní sekvenci se sekvencemi uloženými v databázi. Další algoritmy, které se mohou pro tento účel použít, jsou mimo jiné algoritmy Smith-Waterman a Needleman-Wunsch. Algoritmus Smith-Waterman porovnává sekvence na lokální úrovni jako algoritmus BLAST a algoritmus Needleman-Wunsch porovnává sekvence na globální úrovní. Popis těchto algoritmů naleznete ve třetí části. Tato část vysvětluje princip porovnávání jednotlivých algoritmů. A dále poukazuje na principy a rozdíly substitučních matic BLOSUM a PAM.

Pro určení významnosti nalezené podobné sekvence za použití algoritmu BLAST nám slouží Karlin-Altschul statistika. Popis této statistiky je součástí čtvrté části. Vysvětluje počítání jednotlivých skóre a pravděpodobností výskytu nalezených podsekvencí. Tato kapitola obsahuje nastínění historie porovnávání sekvencí, což popsala Margaret Dayhoff v 60., 70. letech.

Algoritmus BLAST je implementován v databázích Oracle 10g ve formě funkcí. Výstupem této práce je právě aplikace, která implementované funkce v databázích Oracle 10g využívá. V této aplikaci jsou také implementovány algoritmy Smith-Waterman a Needleman-Wunsch. Cílem aplikace je porovnávání vstupní sekvence se sekvencemi umístěnými v databázi v tabulkách, tedy nalezení nejpodobnějších sekvencí umístěných v databázi vůči zadané vstupní sekvenci. Výstupem této aplikace jsou nalezené, podobné sekvence a číselné či procentuální vyjádření jejich podobnosti. Popis analýzy, implementace aplikace a jednotlivých algoritmů naleznete v páté části, kde se dočtete i o testování a srovnání jednotlivých algoritmů.

1 Genetika

Tato kapitola popisuje genetiku, zabývá se především genetikou na molekulární úrovni. Pojednává o jednotlivých molekulách, jejich vzniku, struktuře a významu.

1.1 Úvod do genetiky

Genetika se zabývá dědičností a proměnlivostí organismu, sleduje rozdílnost genetických znaků při přenosu z rodiče na potomka. Genetická informace určuje budoucí fyziologickou stavbu organismu. Dále má velké místo při pohlavním a nepohlavním rozmnožování (klonování).

Genetika nyní pomáhá vědcům při klonování důležitých lidských buněk, jejich zkoumání a výskyt. Toto napomáhá k předurčovaní lidské predispozice k určitým nemocem. Uplatňuje se při léčbě a studiu rakoviny. Dále genetika má obrovský podíl k usvědčování pachatelů dle jejich genetické informace.

Genetika má čtyři úrovně, dle toho co zkoumá[2]:

1. úroveň organismu – přenos genů a vloh z rodičů na potomka
2. úroveň buňky – studuje části buňky, na něž je vázán přenos genetické informace
3. úroveň molekulární – zkoumá organické látky, na které jsou vázány procesy dědičnosti
4. úroveň populací – závislost výskytu některých znaků v populaci

Tato práce se bude zabývat genetikou na úrovni molekulární. Molekulární biologie studuje buněčné a biologické procesy a tudíž se věnuje popisu molekul. Tři nejdůležitější molekuly jsou DNA, RNA, proteiny.

1.2 DNA

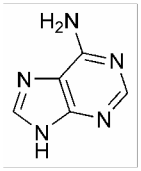
DNA neboli kyselina deoxyribonukleová patří mezi nukleové kyseliny a je nositelkou genů. DNA je dvoušroubovice tvořena ze [2]:

1. zbytku kyseliny fosforečné (H_3PO_4)
2. deoxyribózy - cukr
3. dusíkatých bází[1]: Purinové báze: Adenin A, Guanin G, Pirimidinové báze: Cytozin C, Thymin T

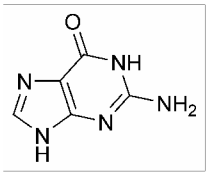
Základem nukleových kyselin, tedy i DNA, jsou nukleotidy, které jsou tvořeny z dusíkatých bází a deoxyribózy. Nukleotidy jsou uspořádány v řadě za sebou a jsou spojeny fosfátovými zbytky, které spojují uhlík 3'[1] jedné deoxyribózy s uhlíkem 5'[2] druhé deoxyribózy.

Purinové báze[1]:

Adenin A

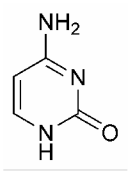


Guanin G

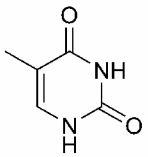


Pyrimidinové báze[1]:

Cytozin C



Thymin T



Princip komplementarity:

Princip komplementarity nám určuje spojení dusíkatých bázích ve dvoušroubovici, to znamená, že např. dusíkatá báze Adenin v prvním vlákně se může spojit pouze s bází Thymin v druhém vlákně dvoušroubovice. *Viz tabulka č.1- Princip komplementarity*

DNA		DNA
A	=	T
G	≡	C
C	≡	G
T	=	A

tabulka č.1: Princip komplementarity DNA – DNA

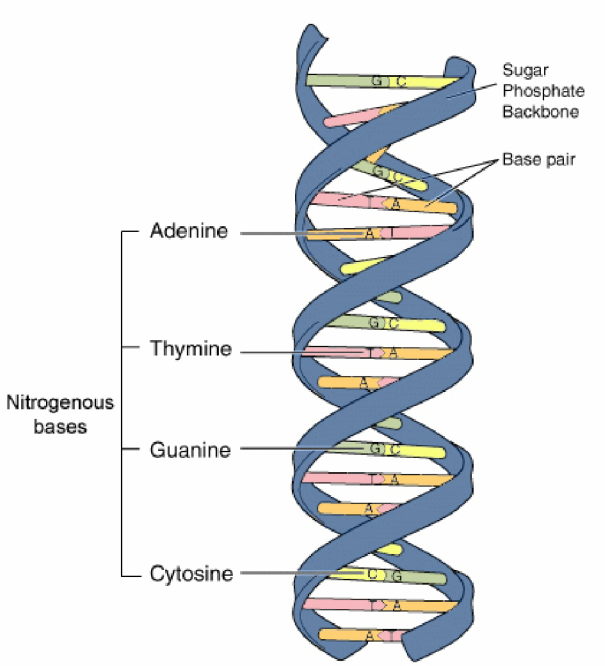
Jednotlivé báze, které se spojují dle principu komplementarity, drží při sobě díky vodíkovým můstkům[3]. Tyto vodíkové můstky také drží dvoušroubovici DNA pohromadě [4]. Princip komplementarity se používá při ***replikaci*** DNA.

---

[1] Konec 3' - DNA končí skupinou OH

[2] konec 5' - DNA končí skupinou s uhlíkem, tedy skupinou $HOCH_2$, na kterou se navazuje fosfátová skupina

[3] Mezi Adeninem a Thyminem jsou 2 vodíkové můstky a mezi Guaninem a Cytozinem jsou 3 vodíkové můstky



obrázek č.1: Dvoušroubovice DNA se spojenými nukleotidy vodíkovými můstky dle principu komplementarity [8]

1.2.1 Úrovně struktury DNA

Strukturu DNA lze rozdělit dle jejího prostorového uspořádání. Základní strukturou je primární struktura, kterou dále využijeme při vyhledávání za použití algoritmu BLAST, Smith-Waterman a Needleman-Wunch.

1. **Primární struktura [6]**

je dána pořadím nukleotidů, jejich sekvencí. Tato struktura určuje genetickou informaci. Lze ji znázornit jako pořadí písmen, které značí dusíkaté báze.

2. **Sekundární struktura [7]**

je ve formě dvou vláken stočených do dvoušrobovice, nicméně existuje několik forem stočení[4], kde většina DNA je v B formě. Druh formy stočení má vliv na funkci.

[4] Forma A – pravotočivá, 10 párů bází na otáčku, Forma B – pravotočivá, 11 párů bází na otáčku, Forma C – levotočivá, 12 párů na otáčku

3. **Vyšší úroveň struktury [7]**

Jsou to struktury dané dlouhou DNA, kdy se DNA nejenže stáčí např. do dvoušroubovice, ale i dále se skládá a navíjí, čemuž se říká nadšroubovicové vinutí.

1.2.2 Replikace DNA

Replikace DNA znamená, že se DNA podle principu komplementarity sama zdvojí. Toto zdvojení slouží například k předávání informací dceřinným buňkám.

Fáze:

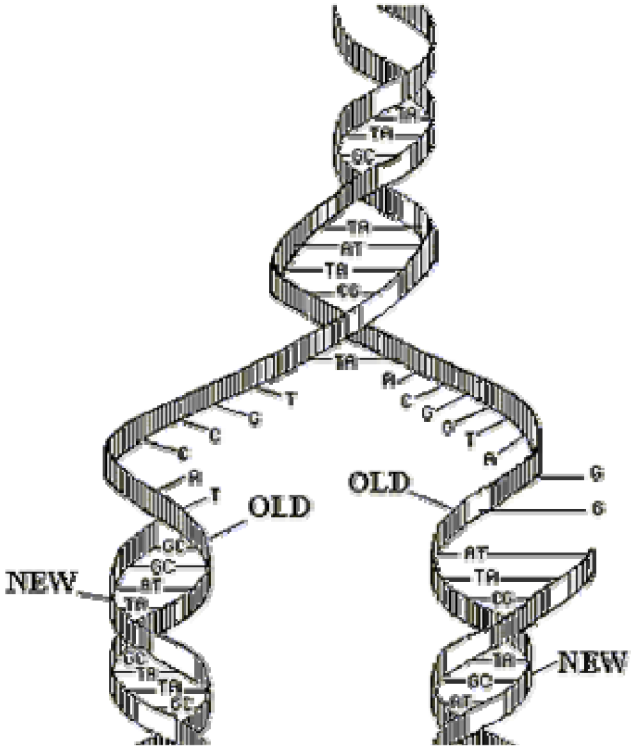
1. **Iniciace**

Dvoušrobovice DNA se rozpojí v iniciačním místě zvaném také lokus OriC[5] [4]. V této fázi se vlákna molekuly začnou rozplétat a vzdalovat vlivem enzymu helikázy.

2. **Elongace – správné navázání nukleotidů**

Dále se vytvoří krátký úsek RNA, který je dle principu komplementarity navázán na vlákno DNA.

Na tento úsek se začnou navazovat nukleotidy dle principu komplementarity *viz tabulka č. 1: Princip komplementarity DNA – DNA*. Při navazování nukleotidů mohou vznikat Okaziho fragmenty[6] [4], které jsou však následně pospojovány enzymem DNA – ligáza. DNA – polymeráza poté opravuje své chyby, kontroluje správné párování napojených nukleotidů [4]. Tímto je replikace kompletní, vznikly 2 nové identické DNA.



obrázek č.2: Replikace DNA – výroba dceřinných DNA [9]

Replikace DNA jakožto každý jiný proces potřebuje energii. Tuto energii jim dodávají trifosfáty ATP, GTP, CTP, TTP [4]

[5] Bakteriální buňka má místo lokus Oric pouze jedno, kdežto eukariotní buňky mají těchto míst mnoho.

[6] Okaziho fragmenty – opožďující se řetězce syntetizovány jako série segmentů, vznikají z důvodů, že DNA - polymeráza může syntetizovat pouze ve směru 3' – 5'

1.3 RNA

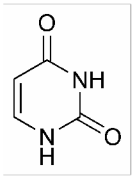
RNA neboli ribonukleová kyselina se podílí na procesech, kterými je genetická informace realizována. RNA je jednošroubovice tvořena ze [3]:

1. zbytku kyseliny fosforečné
2. ribózy - cukr
3. dusíkatých bází: Purinové báze: Adenin A, Guanin G, Pirimidinové báze: Cytozin C, Uracil U

RNA je jako DNA také nukleová kyselina, to znamená, že vytváří nukleotidy z ribózy a dusíkatých bází. Rozdíl je pouze v tom, že RNA nemá dusíkatou bázi Thymin, ale namísto Thyminu obsahuje Uracil, který se váže na Adenin. *Viz tabulka č.2 Princip komplementarity*

Pyrimidinová báze[1]:

Uracil



Struktura ostatních bází: Adenin, Guanin, Cytozin je stejná jako u DNA

Princip komplementarity:

Princip komplementarity nám určuje spojení dusíkatých bázích při vytváření RNA ***transkripcí***.

DNA		RNA
A	=	U
G	≡	C
C	≡	G
T	=	A

tabulka č.2: Princip komplementarity DNA – RNA

U RNA se princip komplementarity používá při ***transkripci.***

1.3.1 Transkripce

Transkripce je proces, který přepíše DNA do mRNA. Nepotřebuje primer ke startu jako tomu bylo u replikace DNA.

Fáze:

1. **Iniciace**

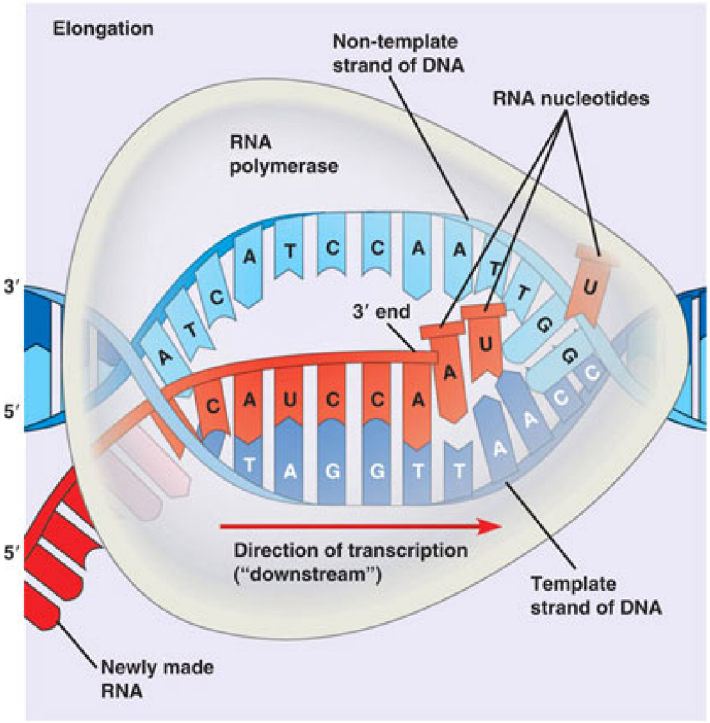
Transkripce začíná v místě, kde RNA – polymeráza nalezne promotor[7].

2. **Elongace**

RNA – polymeráza rozplétá DNA a vytváří nukleotidy[8] a dále pre-mRNA dle principu komplementarity *viz tabulka č.2: Princip komplementarity DNA – RNA* [5].

3. **Terminace**

Elongace pokračuje dokud RNA – polymeráza nenarazí na terminační sekvenci v DNA. V této fázi transkripce končí a je vytvořena pre-mRNA [5].



obrázek č.3: Transkripce [10]

Pre-mRNA obsahuje segmenty zvané introny, které dále přerušují sekvence kódující aminokyseliny – exony. Proto jsou introny v jádře vystřiženy a tím vzniká mRNA [5].

Typy RNA[3]:

1. mRNA – mediátorová – jeden řetězec, ve kterém jsou obsaženy dusíkaté báze, slouží jako matice, podle které se budou spojovat aminokyseliny při tvorbě bílkovin, je vytvořena transkripcí
2. tRNA – transferová – RNA slouží jako dopravce vhodné aminokyseliny při sestavování řetězce
3. rRNA – ribozómová – podílí se na stavbě ribozómů, což je místo pro výrobu bílkovin

[7] promotor - sekvence nukleotidů, která určuje počátek transkripce.

[8] vytváření nukleotidů je ve směru 5' ke konci 3'

1.4 Aminokyseliny

Důležitou součástí organismů jsou bílkoviny, což jsou řetězce aminokyselin - polypeptidy spojené peptidovou vazbou[9]. Bílkoviny se vyrábí translací – přepsáním řetězce nukleotidů mRNA do řetězce aminokyselin bílkovin *viz 1.4.2 Translace.*

Aminokyseliny

Jsou základními prvky proteinů. Sekvence je kódována genetickým kódem tripletů DNA, dělí se na[3]:

hydrofilní	**hydrofobní**	**zásadité**	**kyselé**
Cystein Cys (**C**)	Alanin Ala (**A**)	Histidin His (**H**)	Kyselina asparagová Asp (**D**)
Glycin Gly (**G**)	Fenylalanin Phe (**F**)	Lysin Lys (**K**)	Kyselina glutamová Glu (**E**)
Asparagin Asn (**N**)	Isoleucin Ile (**I**)	Arginin Arg (**R**)	
Glutamin Gln (**Q**)	Leucin Leu (**L**)		
Serin Ser (**S**)	Methionin Met (**M**)		
Threonin Thr (**T**)	Prolin Pro (**P**)		
Tyrozin Tyr (**Y**)	Valin Val (**V**)		
	Tryptofan Trp (**W**)		

Tabulka č.3: Seznam aminokyselin

Jedna aminokyselina může být kódována více triplety, což zajišťuje větší stabilitu vůči mutaci.

Kódování aminokyselin[3]:

	AGA									UUA		
	AGG									UUG		
GCA	CGA						GGA			CUA		
GCC	CGC						GGC		AUA	CUC		
GCG	CGG	GAC	AAC	UGC	GAA	CAA	GGG	CAC	AUC	CUG	AAA	
GCU	CGU	GAU	AAU	UGU	GAG	CAG	GGU	CAU	AUU	CUU	AAG	AUG
Ala	Arg	Asp	Asn	Cys	Glu	Gln	Gly	His	Ile	Leu	Lys	Met
A	R	D	N	C	E	Q	G	H	I	L	K	M

[9] Peptidová vazba je tvořena mezi karboxylovou skupinou (CO) jedné aminokyseliny a amino skupinou (NH) druhé aminokyseliny odštěpením vody.

	AGC	AGU					
	CCA	UCA	ACA			GUA	
	CCC	UCC	ACC			GUC	UAA
UUC	CCG	UCG	ACG		UAC	GUG	UAG
UUA	CCU	UCU	ACU	UGG	UAU	GUU	UGA
Phe	Pro	Ser	Thr	Trp	Tyr	Val	stop
F	P	S	T	W	Y	V	

Kodony UAA, UAG, UGA signalizují konec sekvence a kodon AUG je startovací kodon.

1.4.1 Struktura proteinů

Jakož tomu bylo u DNA i proteiny mohou mít jiné prostorové uspořádání, neboli strukturu. V dalších fázích se bude pracovat s primární strukturou.

1. **Primární struktura [6, 16]**
   peptidový řetězec – aminokyseliny spojeny peptidovou vazbou
2. **Sekundární struktura [6, 16]**
   symetrické prostorové uspořádání polypeptidového řetězce[10]
3. **Terciální struktura [6, 16]**
   asymetrické prostorové uspořádaní polypeptidového řetězce spojení sekundární struktury s nějakými ostatními
4. **Kvartérní struktura [6, 16]**
   uspořádaní jednotlivých polypetidových řetězců v molekule proteinu

1.4.2 Translace

Translace probíhá na ribozómech, které v sobě obsahují několik částí: A – aminoacylové místo tRNA, P – peptidové místo, E – exit. Dále se ribozomy sestávají ze dvou podjednotek, které obsahují rRNA a proteiny [5].

Fáze:

1. **Iniciace**

Iniciační komplex se sestává z obou ribozomálních jednotek, mRNA a tRNA [5]

[10] Vyskytují se jako *Alfa – helix* – pravotočivá šroubovice a *Beta – struktura* – jednotlivé řetězce jsou položeny vedle sebe a spojeny H-můstky

2. **Elongace**

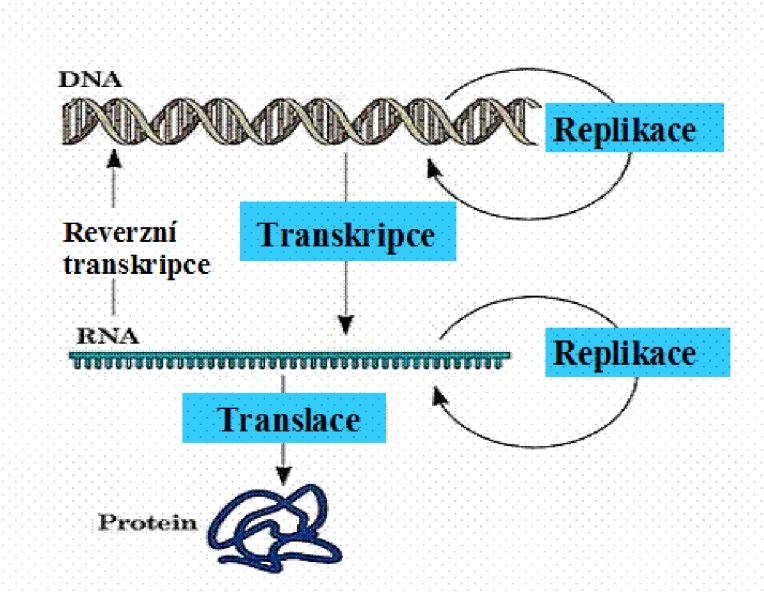
Samotná translace probíhá tak, že ribozomem prochází mRNA, kde jsou rozpoznány jednotlivé kodony mRNA pomocí tRNA. tRNA přinese příslušnou aminokyselinu, která je včleněna do polypeptidového řetězce [5].

3. **Terminace**

Translace končí nalezením stop kodonu u mRNA. Nově vzniklý polypeptid se uvolní od tRNA. tRNA se uvolní od ribozomu a obě ribozomální podjednotky se uvolní od mRNA [5].

1.5 Shrnutí

Jak již bylo v předchozích kapitolách uvedeno, DNA je dvoušroubovice a je nositelkou genů. Kopii tohoto genu, či dceřinnou DNA lze získat replikací – okopírováním DNA. Dále se z DNA může vytvořit mRNA, což je jednošroubovice, transkripcí - vznik mRNA z DNA, která nám slouží především k výrobě proteinů translací[1]. Další složky RNA jako tRNA a rRNA tomuto procesu napomáhají. *Viz obrázek č.4*



Obrázek č.4: Průběh výroby proteinů, dceřiných DNA a RNA [11]

Pro porovnávání sekvencí algoritmem BLAST, ale i jinými algoritmy jako Smith-Waterman a Needleman-Wunsch se používá DNA či protein ve formě sekvencí písmen, které vyjadřují nukleotidy (dusíkaté báze) u DNA a aminokyseliny u proteinů.

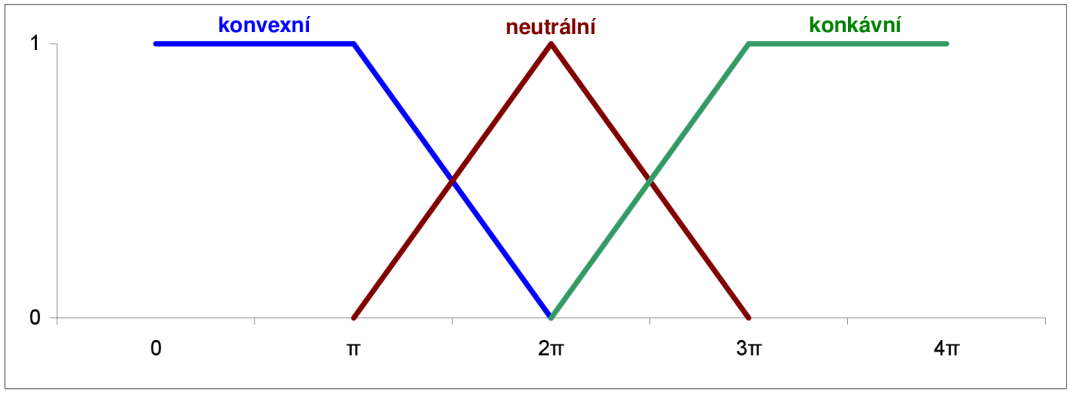
2 Uložení genomických informací v databázi

Genetické informace se nejčastěji ukládají v databázi. Nicméně, v jakých typech databází tyto údaje ukládáme, záleží na dalším využití těchto dat. První typ databází, kde se mohou ukládat genetické informace jako DNA a proteinové sekvence, genová funkční data či dokonce obrazová data, jsou prostorové databáze.

2.1 Prostorové databáze

Prostorové databáze nabízí ukládání genomických dat jako vícedimenzionální obrazce, nejméně však ve 2D. Genomická data se mohou ukládat ve formě 3D, kdy povrch je tvořen trojúhelníky, *viz. bod 1*. Nebo forma uložení informací může být podobná GIS, *viz bod 2*.

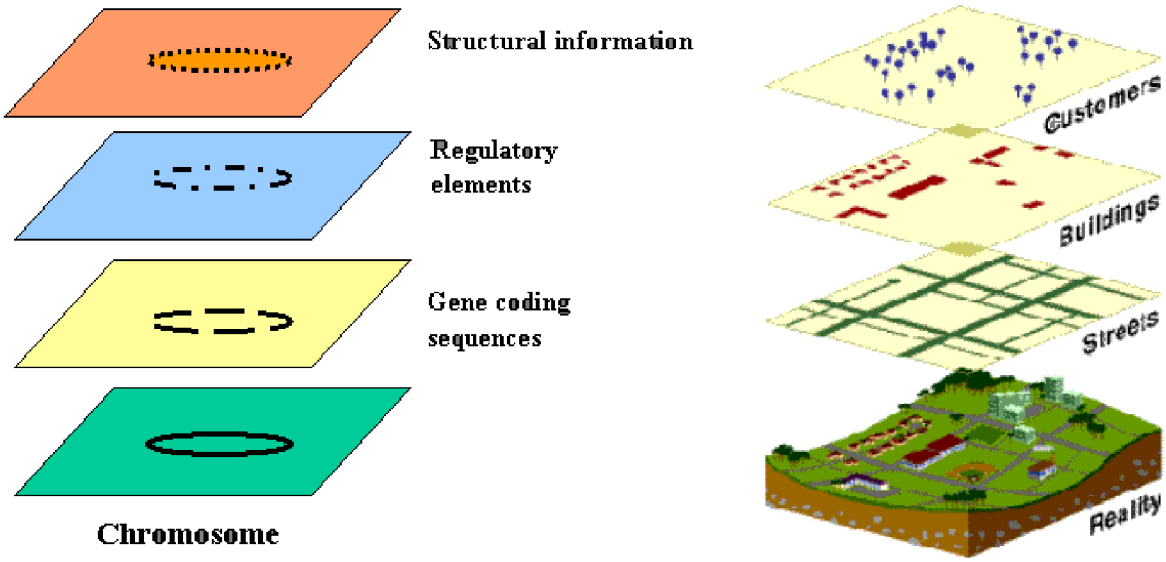
1. Například u proteinové prostorové databáze každý protein má trojúhelníkový povrch s 3D body. Dále se počítá prostorový úhel, který udává konkávnost, konvexnost či rovinu v bodech povrchu daného proteinu. Rozlišení konkávnosti, konvexnosti a neutrálnosti je dle fuzzy systému [13]:



Obrázek č.5: Fuzzy systém pro odvození konvexnosti a konkávnosti vrcholů trojúhelníků tvořící 3D objekt

2. Uložení genomických informací v databázích podobných GIS je ve formě interaktivní genomické mapy. Rysy genetických informací a jejich atributy jsou implementovány jako prostorové objekty a datové vrstvy, které mohou být zobrazeny uživateli. Například vrstvy mohou obsahovat vždy jednu z některých důležitých částí ukládané informace. Například chromozom se skládá z několika důležitých částí jako strukturální informace, řídící prvky a genomická sekvence. To znamená, že potom jedna vrstva bude obsahovat strukturální

informace, druhá vrstva zase řídící prvky a třetí genomickou sekvenci, *viz obrázek č.6*. S pomocí dalších systémů a nástrojů lze data z prostorových databází zobrazit, dotazovat se a vyhledávat části sekvence a následně tyto data analyzovat [14].



Obrázek č.6: Rozložení částí chromozomů do vrstev a jejich návaznost na GIS [14]

2.2 Deduktivní databáze

Deduktivní databáze jsou založeny na technologii, která poskytuje objektově orientovaný konceptuální model, deklarativní pravidla jazyka (například Prolog) a procedurální jazyk. Takováto kombinace nám poskytuje větší verifikaci dat, automatickou detekci a konzistentní údržbu.

Objektově orientovaný konceptuální model nám poskytuje informaci, jak je daná struktura složena. Datové modely se seskupují do tříd a podtříd, které sdílí vlastnosti dědičností.

Deduktivní pravidla poskytují logický důsledek. Toto je právě dobře využitelné v molekulární biologii za použití jazyka Prolog, který je dobře kombinovatelný s objektově orientovanou databází. [12]

2.3 Relační databáze

Relační databáze jsou takové, které vyhovují relačnímu modelu dat a relační algebře. Databázi lze ovládat pomocí jazyka SQL pomocí dotazů nad tabulkami a dalšími příkazy.

Tyto databáze se budou využívat při tvorbě aplikace, kde sekvence proteinů a nukleotidů, budou uloženy v tabulkách. Samotná reprezentace proteinů v databázích je tvořena formou řetězce

písmen, která nám značí, z jakých aminokyselin se daný protein skládá. Popis písmen, která nám označují aminokyseliny, naleznete výše *viz 1.3 Aminokyseliny.*

V některých tabulkách může být i informace o typu proteinu, či jeho založení. Avšak nejčastěji takovéto databáze obsahují identifikátor sekvence a samotný řetězec sekvence.

3 Algoritmus BLAST

Vysvětlení použitých výrazů v následujícím textu:

zarovnání – dvě upravené, podobné sekvence, které jsou výsledkem některého z algoritmu

Smith-Waterman algoritmus – algoritmus, který vyhledává podobné sekvence na úrovni lokální, to znamená, že výsledkem porovnávání jsou podobné části sekvencí

Needleman-Wunsch algoritmus – algoritmus, který vyhledává podobné sekvence na úrovni globální, to znamená, že porovnává celou délku vstupních sekvencí a ve výsledku je zobrazena celá délka sekvencí

podsekvence – část dotazované nebo databázové sekvence

část dotazované sekvence – podsekvence vytvořena v 1. kroku algoritmu BLAST o délce w

list sousedů –seznam, který obsahuje různé variace všech písmen (jde o variaci s opakováním) o délce slova w (parametr algoritmu BLAST) se skórem větší než je práh T (parametr algoritmu BLAST)

soused – prvek v listu sousedů

skóre – bodové ohodnocení (číslo) podobnosti dvou znaků nebo dvou sekvencí, větší skóre – větší podobnost

úspěch (hit)– nalezená podobná podsekvence sekvence umístěné v databázi

HSP – Hight Scoring Segment Pair - HSP je lokální zarovnání, které neobsahuje mezery a zároveň dosahuje největších skór v daném vyhledávání [18]

MSP - Maximal Segment Pair – HSP, které má nejlepší skóre

Parametry algoritmu BLAST[15]:

délka "slova" w – parametr použitý při vytváření listu sousedů, jde o počet písmen v podsekvenci

práh T – např. při zvolené délce "slova" 3 se vytvoří list sousedů, dále se berou v úvahu pouze podsekvence, které mají skóre získané ze substituční matice nebo ze skórového systému větší než je tento práh

X – dropoff – hodnota, která se používá u rozšíření nalezených podsekvencí, kdy výsledné skóre nesmí klesnout o tuto hodnotu

S - skóre – minimální skóre, pod které se algoritmus nesmí dostat při rozšíření nalezené podsekvence

A – distance – vzdálenost 2 podsekvencí, tento parametr se používá při rozšíření nalezených podsekvencí, pokud dva sousedé – prvky z listu sousedů jsou od sebe vzdáleny maximálně o A a nepřekrývají se

3.1 Princip porovnávání

Sekvence nukleotidů a proteinů, které jsou vyjádřeny formou písmen se mohou porovnávat různými bioinformatickými nástroji či algoritmy. V této práci jsou uvedeny a popsány pouze algoritmy BLAST, Smith-Waterman a Needleman-Wunsch.

Porovnávání sekvencí (řetězců písmen) si můžeme představit jako porovnávání jednotlivých znaků. To nám slouží k určení podobnosti dotazované sekvence vůči cílovým sekvencím (těchto sekvencí může být více – např. mohou být uloženy v databázi).

Porovnání se provádí na základě daného systému, který je závislý na zvoleném algoritmu či jeho implementaci. Tento systém nám vrací jednotlivá ohodnocení (čísla) dle podobnosti znaků. Ohodnocení se mohou lišit v závislosti na použitém systému, kdy například ohodnocení počítané ze substituční matice, *viz. 3.3.2.1 Substituční matice,* zahrnuje možnost mutace proteinu na jiný. Tedy některé páry písmen jsou ohodnoceny kladně (jsou si podobné), ale přitom nejsou stejné.

Jedním z výsledků porovnání je skóre (číslo), které nám určuje podobnost nalezených sekvencí. V tomto skóre jsou započítána ohodnocení z výše uvedeného systému. Čím je toto číslo větší, tím si jsou nalezené sekvence podobnější. Dalším statistickým údajem, který nám již vrací funkce představující implementaci výše uvedených algoritmů, je procentuální identita, tedy vyjádření poměru mezi kladně ohodnocenými páry a celkovou délkou sekvence v procentech.

3.2 Úvodem k algoritmu BLAST

BLAST, **B**asic **L**ocal **A**lignment **S**earch **T**ool je algoritmus pro porovnávání dotazované sekvence aminokyselin nebo nukleotidů vůči sekvenci umístěné v databázi. Tento algoritmus byl uveden v roce 1990 panem S. Altschul. Vzniklo mnoho variací tohoto algoritmu pro porovnání nukleotidů a proteinů. Algoritmus využívá při porovnání sekvencí matici PAM nebo BLOSUM.

Algoritmus BLAST vychází ze Smith – Waterman algoritmu, který se využívá pro lokální porovnávání, jehož výsledkem je zarovnání mezi podsekvencí A a podsekvencí B. Dále ještě existuje globální porovnávání – výsledné zarovnání obsahuje celou délku sekvencí A a B. Globálním porovnáváním se zabývá algoritmus Needleman-Wunsch. [17]

BLAST je rychlejší, ale méně citlivý, používá dynamický přístup. Pokouší se omezit použití na sekvence, které se zdají být zajímavé. Heuristická část algoritmu se snaží vytvořit odhad, která sekvence může produkovat zajímavé zarovnání

Algoritmus BLAST je implementován do databáze Oracle ve formě funkcí:

- BLASTN – porovnává nukleotidovou sekvenci s nukleotidovou databází (tabulkou)
- BLASTP – porovnává proteinovou sekvenci s proteinovou databází (tabulkou)

- BLASTX – překládá nukleotidovou sekvenci na proteinovou a následně ji porovnává oproti proteinové databázi (tabulce)
- TBLASTN – porovnává proteinovou sekvenci oproti nukletidové databázi (tabulce) přeložené na proteinovou
- TBLASTX – převádí nukleotidovou sekvenci na proteinovou a porovnává oproti nukleotidové přeložené databázi (tabulce)

Pozn. č.1: Algoritmus implementovaný do databází Oracle je velice blízký NCBI – BLAST 2.0, což je verze algoritmu BLAST.

3.3 Algoritmus vyhledání

Samotné porovnávání sekvencí lze rozdělit na několik kroků:

V prvním kroku dochází k předzpracování dotazů – rozdělení dotazované sekvence dle délky, kterou jsme zvolili pro délku slova, a vytvoření listu sousedů.

Ve druhém kroku se počítá výsledné skóre na základě zvolené substituční matice u proteinů nebo dle skórového systému u porovnávání nukleotidů a generují se úspěchy (hits).

Ve třetím kroku dochází k rozšíření nalezeného úspěchu (hit) a následné počítání skóre.

3.3.1 Předzpracování dotazu

První část algoritmu BLAST (předzpracování dotazu) vytváří list sousedů ke každé části dotazované sekvence. Postup je následující:

Nejprve si zvolíme ***délku "slova" w***. Délka slova w znamená, že dotazovaná sekvence (sekvence, kterou chceme nalézt) se rozdělí na části o délce w. Například ze sekvence o délce 10 vznikne 8 částí s délkou 3. Každou takovou část nazvěme ***část dotazované sekvence***. Pro každou část dotazované sekvence je vytvořen ***list sousedů*** [11] [15].

Příklad č.1:

Zadáme délku slova 3, tedy w = 3, dotazujeme se na sekvenci ATCGACT, vytvořte všechna části dotazované sekvence.

[11] list sousedů – veškerá možná slova – všechny možné variace odpovídajících písmen dle toho, zda jde o nukleotidy či aminokyseliny

Řešení příkladu č.1:

Ze sekvence ATCGACT vzniknou tyto části dotazované sekvence:

A	***T***	***C***	***G***	***A***	***C***	***T***
A	*T*	*C*				
	T	*C*	*G*			
		C	*G*	*A*		
			G	*A*	*C*	
				A	*C*	*T*

Jak již bylo naznačeno výše, každá část dotazované sekvence je spojena s listem sousedů, ze kterých se nakonec vybírají ti sousedi (prvky v listu sousedů), kteří mají skóre z porovnání s části dotazované sekvence větší než zadaný práh *viz 3.3.2 List sousedů a generování úspěchů (hits).*

Příklad č.2:

*Pokud vyhledávám aminokyseliny (počet aminokyselin je 20) s délkou w = 2, bude generováno 20*20, tedy 400 zarovnání*

3.3.2 List sousedů a generování úspěchů (hits)

Druhá část algoritmu nazvaná List sousedů a generování úspěchů (hits) navazuje na první část, kde se vytvořil list sousedů. V něm dále tato druhá část ponechává pouze ty sousedy, jejichž skóre splňují podmínku prahu T, a generuje úspěchy (hits) nalezených podobných slov v databázi a spolu s jejich pozicí je zaznamenává. Postup:

Části dotazované sekvence se porovnávají s příslušným listem sousedů a generuje se skóre[12]. Pokud je nastaven ***práh T,*** v listu sousedů jsou obsaženi pouze ti sousedi, jejichž skóre z porovnání s části dotazované sekvence je větší nebo rovno zadanému prahu T.

V této fázi je každá část dotazované sekvence reprezentována listem sousedů. Samotné porovnání mezi dotazovanou sekvencí a sekvencí umístěnou v databázi je reprezentováno porovnáváním jednotlivých sousedů z listu sousedů s každým slovem (podsekvence o délce w) sekvence umístěné v databázi. V každém porovnávání je počítáno skóre. Skóre se získávají ze substituční matice (*viz. 3.3.2.1 Substituční matice*) u proteinů nebo formou skórového systému (*viz. 3.3.2.2 Skórový systém*) u nukleotidů.

Za úspěch (hit) je považováno takové porovnání mezi sousedy a slovy v cílové sekvenci (sekvenci umístěné v databázi), které si odpovídá – soused je stejný se slovem v cílové sekvenci. Tento úspěch (hit) spolu s jeho pozicí v obou sekvencích je zaznamenám. [15]

---

[12] skóre se počítá dle substituční matice – viz 3.3.2.1 Substituční matice nebo podle skórového systému – viz 3.3.2.2. Skórový systém

3.3.2.1 Substituční matice

Substituční matice obsahuje ohodnocení jednotlivých podobností proteinů. Ohodnocení bere v úvahu možnost změny některé aminokyseliny na jinou. Je to z důvodu, že během evoluce se z jedné generace na druhou mohou sekvence pozměnit nebo zmutovat.

Příklad č.4:

ALEIRYLRD

může zmutovat na:

ALEINYLRD

Substituční matice vyjadřuje pravděpodobnost transformace se skórem:

$$S_{i,j} = \log \frac{p_i * M_{i,j}}{p_i * p_j} = \log \frac{M_{i,j}}{p_j} = \frac{PozorovanaFrekvence}{OcekavanaFrekvence}$$

kde $M_{i,j}$ je pravděpodobnost amino-kyseliny *i* se změnit na amino-kyselinu *j* a p_i je frekvence amino-kyseliny i.

Substituční matice[18]:

1. **PAM**

   Tato matice byla vytvořena jako silně teoretická a je v ní obsaženo velice málo evolučních předpokladů, byla počítána sledováním odlišností u blízce souvisejících proteinů. Matice PAM1 byla vytvořena se sadou proteinů, které si byly identické z 85% a více. Matice PAM1 je základem pro počítání ostatních matic spolu s ošetřením opakovaných mutací. Nyní se nejvíce používá matice PAM30 a PAM70.

2. **BLOSUM**

   Metoda, která se využívá u matice PAM, nepracuje příliš dobře u odchylujících se sekvencí během evoluce. Změny v sekvenci během delší evoluční doby se příliš nepřibližují tomu, pokud bychom skládali jednotlivé změny za kratší časové úseky. Matice BLOSUM řeší tento problém. Tato matice je více empirická a vytvořena po zkoumání velkého počtu dat. K vytvoření této matice bylo využito rozmanitých uspořádání proteinů, které se během evoluce odchylovaly. Pravděpodobnost použita v matici je počítána s ohledem na bloky zachovaných sekvencí nalezených v uspořádání. Pro algoritmus BLAST je standardně využívána matice BLOSUM62

Substituční matice BLOSUM 62 [19]:

	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W
C	**9**	-1	-1	-3	0	-3	-3	-3	-4	-3	-3	-3	-3	-1	-1	-1	-1	-2	-2	-2
S	-1	**4**	1	-1	1	0	1	0	0	0	-1	-1	0	-1	-2	-2	-2	-2	-2	-3
T	-1	1	**4**	1	-1	1	0	1	0	0	0	-1	0	-1	-2	-2	-2	-2	-2	-3
P	-3	-1	1	**7**	-1	-2	-1	-1	-1	-1	-2	-2	-1	-2	-3	-3	-2	-4	-3	-4
A	0	0	-1	-1	**4**	0	-1	-2	-1	-1	-2	-1	-1	-1	-1	-1	-2	-2	-2	-3
G	-3	1	1	-2	0	**6**	-2	-1	-2	-2	-2	-2	-2	-3	-4	-4	0	-3	-3	-2
N	-3	0	0	-1	-1	-2	**6**	1	0	0	-1	0	0	-2	-3	-3	-3	-3	-2	-4
D	-3	0	1	-1	-2	-1	1	**6**	2	0	-1	-2	-1	-3	-3	-4	-3	-3	-3	-4
E	-4	0	0	-1	-1	-2	0	2	**5**	2	0	0	1	-2	-3	-3	-3	-3	-2	-3
Q	-3	-1	0	-1	-1	-2	0	0	2	**5**	0	1	1	0	-3	-2	-2	-3	-1	-2
H	-3	-1	0	-2	-2	-2	-1	-1	0	0	**8**	0	-1	-2	-3	-3	-2	-1	2	-2
R	-3	0	-1	-2	-1	-2	0	-2	0	1	0	**5**	2	-1	-3	-2	-3	-3	-2	-3
K	-3	-1	0	-1	-1	-2	0	-1	1	1	-1	2	**5**	-1	-3	-2	-3	-3	-2	-3
M	-1	-2	-1	-2	-1	-3	-2	-3	-2	0	-2	-1	-1	**5**	1	2	-2	0	-1	-1
I	-1	-2	-2	-3	-1	-4	-3	-3	-3	-3	-3	-3	-3	1	**4**	2	1	0	-1	-3
L	-1	-2	-2	-3	-1	-4	-3	-4	-3	-2	-3	-2	-2	2	2	**4**	3	0	-1	-2
V	-1	-2	-2	-2	-2	0	-3	-3	-3	-2	-2	-3	-3	-2	1	3	**4**	-1	-1	-3
F	-2	-2	-2	-4	-2	-3	-3	-3	-3	-3	-1	-3	-3	0	0	0	-1	**6**	3	1
Y	-2	-2	-2	-3	-2	-3	-2	-3	-2	-1	2	-2	-2	-1	-1	-1	-1	3	**7**	2
W	-2	-3	-3	-4	-3	-2	-4	-4	-3	-2	-2	-3	-3	-1	-3	-2	-3	1	2	**11**

Jak si u této matice můžete všimnout, skóre u některých párů písmen je větší než u jiných, je to dáno především tím, že substituční matice je vytvořena tak, že bere v úvahu chemickou strukturu daných aminokyselin a tudíž i možnost mutace, která by neohrozila samotnou strukturu proteinu.

Příklad č.5:

máme dvě sekvence AGC a CGC, které porovnáváme, výsledné body spočítáme s BLOSUM 62 matice a následně je možno jej porovnávat s prahem T

E G C

-4 +6 +9 = 11

C G C

Výsledné skóre je 11.

Vztah mezi maticemi BLOSUM a PAM [19]:

BLOSUM 80	BLOSUM 62	BLOSUM 45
PAM 1	PAM 120	PAM 250

Méně odchylující se ⟷ Více odchylující se

3.3.2.2 Skórový systém

Skórový systém se používá při počítání skóre u porovnávání nukleotidů. Lze si zvolit počet bodů, které se budou do konečného skóre přičítat při shodných písmenech, počet bodů, které se budou od konečného skóre odečítat při neshodných písmenech a počet bodů, které se budou odečítat při začínající a rozšiřující se mezeře [17]. U programu blastn je standardně nastaveno -3 při odlišných písmenech a +1 při stejných písmenech.

Příklad č.6:

Mějme protein tvořen aminokyselinami: AGFRCD, vytvořte listy sousedů, které budou mít práh T = 13 s délkou slova w = 3 za použití substituční matice BLOSUM62.

Řešení příkladu č.6:

Sekvenci AGFRCD na veškeré možné části, které budou mít délku slova 3

A G F R C D
A G F
G F R
F R C
R C D

Dále pro všechny tyto části se vytvoří list sousedů (veškerá možná spojení písmen aminokyselin)

AAA
AAC
AAD
...
WWT
WWY
WWW

Z tohoto listu se vyberou pouze ty sekvence, které mají skóre z porovnání s danými částmi dotazované sekvence větší než zadaný práh T.

AGF	*Skóre*	*GFR*	*Skóre*	*FRC*	*Skóre*	*RCD*	*Skóre*
AGF	*16*	*GFR*	*17*	*FRC*	*20*	*RCD*	*20*
SGF	*13*	*GFK*	*14*	*FKC*	*17*	*KCD*	*18*
AGY	*13*	*GYR*	*14*	*YRC*	*17*	*RCE*	*17*
CGF	*12*	*GFQ*	*13*	*FQC*	*16*	*QCD*	*16*
GGF	*12*	*GFN*	*12*	*FNC*	*15*	*RCE*	*15*
SGY	*10*	*GFE*	*12*	*FEC*	*15*	*NCD*	*15*
SGW	*8*	*GFH*	*12*	..	..	...	..
STF	*8*	*GMR*	*11*	*YQC*	*13*	*KCE*	*13*
...	...	...	..	*WKC*	*12*	*KCT*	*12*

Dále se používají pouze ty sekvence, které mají skóre větší než náš zadaný práh T = 13

3.3.3 Rozšíření generovaných úspěchů (hits)

Třetí část rozšiřuje nalezené úspěchy (hits), které byly spolu s jejich pozicí zaznamenány ve druhé části algoritmu. Tento krok zabírá asi 90% celkového času. Postup:

Každý generovaný úspěch (hit) je rozšířen[13] v obou směrech, rozšiřování je zastaveno, jakmile skóre rozšířeného úspěchu (hit) poklesne o ***X***, kde ***X*** je parametr algoritmu. Každý takový rozšířený segment, jehož skóre je rovno či větší než ***S***, kde ***S*** je také parametr algoritmu, je nazván ***HSP – Hight Scoring Segment Pair***. HSP je podsekvence, která neobsahuje mezery a zároveň dosahuje největšího skóre v daném porovnávání. Nejlepší skóre HSP se nazývá ***MSP - Maximal Segment Pair***[18].

Tato metoda je základní. Verze NCBI-BLAST2, kterou využívá databáze Oracle 10g je založena na tom, že při 1. části, kde se vytváří list sousedů, se zvolí nižší práh T. To zapříčiní, že bude více sousedů v listu sousedů.

Nižší práh se volí z důvodů používání dalšího ***parametru A***, což je maximální vzdálenost 2 nalezených úspěchů (hits). Parametr A se používá při posledním kroku – rozšíření generovaných úspěchů (hits). V případě že dva nalezené úspěchy (hits) mají vzdálenost menší než ***parametr A,*** úspěch (hit) je rozšířen. Vzdáleností se myslí vzdálenost, kdy daný úspěch leží na stejné diagonále jako ten druhý a má vzdálenost A. Nicméně tato metoda vede k menší citlivosti algoritmu.[18]

Pozn. č.2: Při rozšiřování nalezeného segmentu může být do skóre započítána penalta, což jsou záporné body, za „otevření“ nebo „rozšíření“ mezery, takové skóre se nazývá řádkové.

[13] nalezený úspěch (hit) v sekvenci umístěné v databázi je rozšířen – např. k původnímu úspěchu o délce slova w =3 se přidají dva okrajové znaky, takže rozšířený úspěch (hit) bude mít délku slova w = 5, tento rozšířený úspěch (hit) se dále porovnává se vstupní sekvencí

3.4 Srovnání s ostatními algoritmy

Jak již bylo uvedeno v úvodu, algoritmus BLAST byl odvozen od již existujících algoritmů jako Needleman-Wunsch a Smith-Waterman. Tato kapitola by měla popsat algoritmy Needleman-Wunsch a Smith-Waterman a nastínit rozdíly mezi těmito algoritmy a algoritmem BLAST.

3.4.1 Needleman-Wunsch algoritmus

Needleman-Wunsch algoritmus je algoritmus vyhledávající a porovnávající sekvence na globální úrovni. Princip tohoto algoritmu spočívá ve vytvoření matice, kde první horní řádek obsahuje písmena první sekvence a první levý sloupec obsahuje písmena druhé sekvence. Matice se vyplňuje tím způsobem, že se počítá skóre podobnosti jednotlivých párů písmen. [20]

Pro zvolení nejlepšího - nejpodobnějšího zarovnání je zvolen systém S(x, y), jimž může být i substituční matice. Pokud nejde o substituční matice, může se jednat o skórový systém, kde se volí ohodnocení pro shodná písmena(kladné ohodnocení), neshodná(záporné ohodnocení) na indexu x první sekvence a indexu y druhé sekvence.

Skóre v jednotlivých buňkách je počítáno následovně:

$$M[x, y] = \max \begin{cases} M[x-1, y-1] + S(x, y) \\ M[x-1, y] + gap \\ M[x, y-1] + gap \end{cases} \quad (1)$$

kde S(x, y) je systém pro výpočet jednotlivých skóre a gap je penalta (záporné číslo) za otevření či rozšíření mezery.

Při počítání skóre se zaznamenává, ze kterého směru bylo skóre vypočítáno. Toto nám pomůže při hledání a zpětném procházení matice pro určení nejlepších zarovnání. Procházení matice začíná od buňky v pravém dolním rohu matice a dále pokračuje dle šipek, dokud se neskončí v levém horním rohu matice.[20]

Příklad č.7:

máme porovnat 2 sekvence GA-TTA a GAATTC se skórovým systémem uvedeným výše, to znamená:

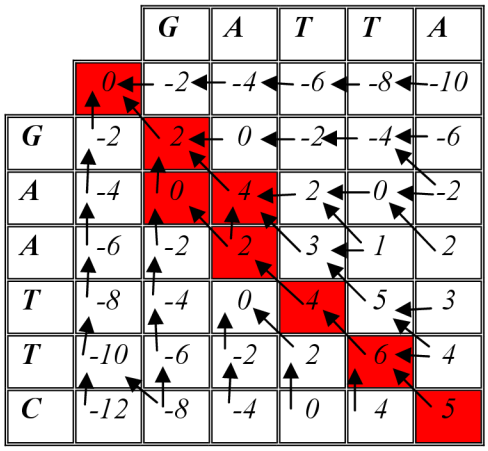
shodná písmena => +2

mezera => -2

špatná písmena => -1

Řešení příkladu č.7:

výsledná matice a průchod vypadá následovně:



Porovnání vypadá následovně:

```
G A - T T A
| |   | |
G A A T T C
```

Výsledné zarovnání obsahuje veškerá vstupní písmena obou sekvencí. Je zde vložena jedna mezera, čtyři písmena se shodují a jedno se neshoduje, výsledná délka zarovnání je 6 a skóre je 5.

```
G - A T T A
|   | | |
G A A T T C
```

Výsledné zarovnání obsahuje veškerá vstupní písmena obou sekvencí. Je zde vložena jedna mezera, čtyři písmena se shodují a jedno se neshoduje, výsledná délka zarovnání je 6 a skóre je 5. Rozdíl od předchozího zarovnání je v pozici mezery.

3.4.2 Smith-Waterman algoritmus

Smith-Waterman algoritmus vznikl modifikací Needleman-Wunsch algoritmu. Slouží pro vyhledávání a porovnávání sekvencí na úrovni lokální. To znamená, že výsledkem porovnávání jsou podsekvence sekvencí. Odlišnosti od Needleman-Wunsch algoritmu jsou [20]:

1. vrchní a levé buňky skórové matice obsahují 0
2. algoritmus nikdy neklesne pod hodnotu 0
3. nejpodobnější zarovnání je nalezeno průchodem vzniklé matice od buňky, která obsahuje největší skóre k buňce obsahující 0

Jednotlivá skóre v buňkách matice jsou počítány následovně:

$$M[x, y] = \max \begin{cases} 0 \\ M[x-1, y-1] + S(x, y) \\ M[x-1, y] + gap \\ M[x, y-1] + gap \end{cases} \quad (2)$$

Systém S(x, y) je obdobný systému, který používá algoritmus Needleman-Wunsch, opět je zde možnost použít skórový systému nebo substituční matici, gap je penalta za mezeru.

Jak již bylo napsáno výše, je princip tohoto algoritmu podobný algoritmu Needleman-Wunsch. Také se do matice zaznamenává skóre a směr odkud bylo vypočítáno *viz Příklad č.8.*

Příklad č.8:

mějme porovnat dvě sekvence GATTA a GAATTC (jako jsme měli u algoritmu Needleman-Wunsch) pomocí algoritmu Smith-Waterman se skórovým systémem S(x, y):

shodná písmena => +2

rozdílná písmena => -1

mezera => -2

minimálním skórem => 5

Řešení příkladu č.8:

výsledná matice je následující

		G	A	T	T	A
	0	0	0	0	0	0
G	0	2	0	0	0	0
A	0	0	4	2	0	0
A	0	0	2	3	1	2
T	0	0	0	4	5	3
T	0	0	0	2	6	4
C	0	0	0	0	4	5

a zarovnané sekvence jsou tři, první dvě představují v matici červená políčka (rozdíl je pouze v pozici mezery) a třetí představuje modrá políčka:

```
G - A T T
|   |   |
G A A T T
```

První sekvence začíná na prvním písmenu a končí na čtvrtém písmenu. Druhá sekvence začíná také na prvním písmenu a končí na pátém písmenu. Délka nalezeného zarovnání je pět a je v ní obsažena jedna mezera, čtyři písmena se shodují. Skóre je 6.

```
G A - T T
| |   | |
G A A T T
```

První sekvence začíná na prvním písmenu a končí na čtvrtém písmenu. Druhá sekvence začíná také na prvním písmenu a končí na pátém písmenu. Délka nalezeného zarovnání je pět a je v ní obsažena jedna mezera, čtyři písmena se shodují. Skóre je 6.

```
G A T T
| |   |
G A A T
```

První sekvence začíná na prvním písmenu a končí na čtvrtém písmenu. Druhá sekvence začíná také na prvním písmenu a končí na čtvrtém písmenu. Délka nalezeného zarovnání je čtyři, tři písmena se shodují a jedno se neshoduje. Skóre je 5.

3.4.3 Závěr

Algoritmus Smith-Waternan porovnává sekvence na úrovni lokální. To znamená, že porovnává celou délku sekvence, ale výsledkem může být zarovnání obsahující pouze podsekvence sekvencí. Algoritmus Needleman-Wunsch porovnává sekvence na úrovni globální, to znamená, že výsledná délka nalezených podobných sekvencí bude minimálně maximum z délky obou porovnávaných sekvencí. Algoritmus BLAST porovnává sekvence opět na úrovni lokální, jak tomu bylo u algoritmu Smith-Waterman.

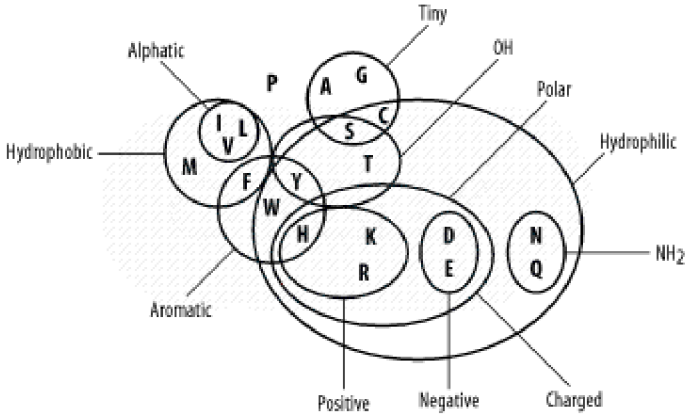
Všechny algoritmy jsou přesné, co se týče genetická mutace, selekce a odchylky, jelikož u algoritmu Smith-Waterman a Needleman-Wunsch lze zvolit substituční matici pro počítání skóre. To znamená, že v případě této volby se nezanedbává zmíněná mutace, selekce a odchylka, jak by tomu mohlo být při volbě skórového systému. Nicméně je zde podstatný rozdíl v rychlosti algoritmu, kde algoritmus BLAST je heuristický na rozdíl od algoritmu Smith-Waterman a Needleman-Wunsch, kde se musí porovnávat každé písmeno. Algoritmus BLAST je mnohem rychlejší nicméně je také v tomto směru méně přesný, to znamená, že nemusí najít veškeré podobné podsekvence.

4 Podobnost sekvencí

Abychom mohli říct, jak si jsou nalezené sekvence podobné, tak k tomu potřebujeme teorii, kterou podložíme správnost vypočítaného skóre. Skóre nám říká, jak moc si jsou sekvence podobné. U algoritmu BLAST se používá statistika Karlin-Altschul. Nicméně si ukážeme i starší teorie jakou měla např. Margaret Dayhoff z toho důvodu, že částečně se využívá ve statistikách Karlin-Altschul.

4.1 Teorie Margaret Dayhoff

V podobnosti aminokyselin nejdříve Margaret Dayhoff propagovala techniku pro měření podobnosti aminokyselin. Používala sekvence, které byly v ten čas dostupné(60., 70. léta), a konstruovala zarovnání ze souvisejících si proteinů. Jak očekávala, byly tam jisté odlišnosti, které se odvíjely od chemické struktury proteinů [21] *viz obrázek č.7.*



obrázek č.7: Chemický vztah aminokyselin [21]

Dále prezentovala podobnost mezi aminokyselinami jako $\log_2$ „pravděpodobnostní“ (odds) koeficient znám také jako „lod“ skóre.

$$S_{ij} = \log_2 (q_{ij} / p_i p_j) \tag{3}$$

kde q_{ij} je frekvence párovaní aminokyselin i a j a p_i a p_j je pravděpodobnost jejich frekvence, pozitivní skóre znamená, že pár písmen je běžný, kdežto negativní skóre znamená, že párování není v pořádku [21].

Příklad č.9 [21]:

Mějme porovnat aminokyseliny M a L. Aminokyselina M se vyskytuje s pravděpodobností frekvencí 0,01 a L s pravděpodobností 0,1. Jejich frekvence párování je 1/500.

Výsledné skóre je poté

$$S = \log_2(\frac{\frac{1}{500}}{0{,}01 * 0{,}1}) = \log_2 2 = 1$$

Tento výsledek znamená, že lod skóre je 1 a jelikož to je kladné skóre, znamená to, že párování je možné.

4.2 Karlin-Altschul statistika

Algoritmus BLAST je založen na Karlin-Altschul statistice. Nicméně pro jeho pochopení je nutno porozumět λ, což je parametr Karlin-Altschul statistiky.

Hlavní proměnné, které nám vyjadřují významnost skóre, nalezeného zarovnání je E (E-hodnota) a P-hodnota, kde E–hodnota nám udává očekávanou hodnotu výskytu zarovnání v databázi a P-hodnota pravděpodobnost výskytu zarovnání.

4.2.1 Lambda

Řádkové skóre může být nesprávné kvantity. Kdežto normalizované skóre koresponduje k originálnímu lod skóre, nicméně požaduje konstantu λ, která je však odhadována či odvozována [21].

Odvození

Platí

$$\sum_{i=1}^{n}\sum_{j=1}^{i} q_{i,j} = 1 \tag{4}$$

kde q_{ij} je frekvence párování aminokyselin i a j

dále platí

$$\lambda S_{i,j} = \log_e(q_{ij} / p_i p_j) \tag{5}$$

což je odvozeno od takzvaného lod skóre, kde platí rovnice $S_{ij} = \log(q_{ij} / p_i p_j)$ *viz 4.1 Teorie Margaret Dayhoff, vzorec (3)*

Z výše uvedené rovnice vyplývá:

$$q_{ij} = p_i p_j e^{\lambda S_{ij}} \tag{6}$$

λ je odhadována či odvozována a je používána pro počítání očekávaného skóre každého HSP v algoritmu BLAST.[21]

4.2.2 Statistika

Jak již bylo psáno výše Karlin-Altschul statistika je základem pro vyjádření podobností nalezených sekvencí algoritmem BLAST. Karlin-Altschul statistika je statistika lokálního zarovnání. Je založena na pěti centrálních předpokladech [21]:

- pozitivní skóre musí být možné
- očekávané skóre musí být záporné
- písmena sekvence jsou nezávislá a shodně rozmístěna
- sekvence jsou nekonečně dlouhé
- výsledné zarovnání neobsahuje mezery

První dva body jsou pravdivé pro každou skórovou matici založenou na reálných datech. Poslední tři body jsou problematické, protože biologická sekvence má závislosti, není nekonečně dlouhá a obsahuje mezery [21].

Základními hodnotami jsou E-hodnota, která je vyjádřena jako vstupní parametr funkcí, které používají algoritmus BLAST a P-hodnota, kterou lze z E-hodnoty vyjádřit [21].

4.2.2.1 E-hodnota

je očekávaná hodnota výskytu podobné sekvence v databázi mající skóre S nebo lepší. E – hodnota exponenciálně klesá, jak S roste. E – hodnota udává statistické skóre, odráží velikost databáze a použitý skórový systém. Standardní hodnota pro E – hodnotu u blastn, blastp, blastx a tblast je 10. Pokud tuto hodnotu navýšíme, můžeme očekávat nalezení většího počtu podobných sekvencí. Stejně tak i naopak platí: nižší E – hodnota znamená více významné zarovnání [18].

E – hodnota se vypočítá následně:

$$E = Kmne^{-\lambda S} \tag{7}$$

E – je očekávaná hodnota frekvence výskytu HSP mající skóre S nebo větší

K – proměnná závislá na použité matici, často má hodnotu kolem 0,1

λ – parametr, ve kterém je vyjádřena váha pro skórový systém

m – délka dotazované sekvence

n – délka sekvence v databázi

S – řádkové(raw) skóre – skóre ze substitučních matic + skóre za mezerné penalty

Jak lze vidět, vztah mezi skórem S a E-hodnotou je exponenciální, to znamená, že i malé změny ve skóre S mohou vyústit ve velké změny v E-hodnotě.

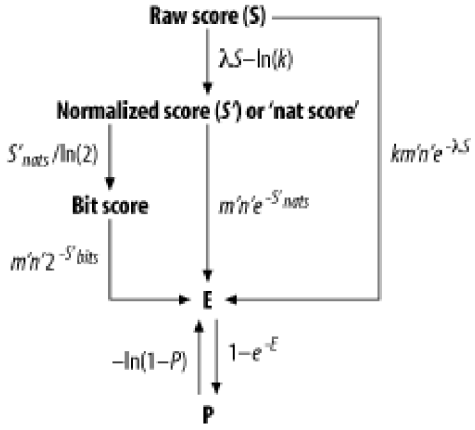
Příklad č.10:

E – hodnota = 0,5 znamená, že podobnost má 5 z 100 možností

4.2.2.2 P – hodnota

P – hodnota udává pravděpodobnost výskytu jednoho nebo více spojení se skórem větším než S. Pro hodnoty E menší než e^{-5} je často akceptována aproximace k P – hodnotě [18]. Jinak pro převod mezi P – hodnotou a E – hodnotou platí následující vzorec:

Výpočet E-hodnoty a P-hodnoty



obrázek č.8: Výpočet E-hodnoty, P-hodnoty

4.2.2.3 Skóre

Raw score (řádkové skóre) S – je skóre zarovnání, vypočítané jako suma skóre ze substitučního skóre a ze skóre mezer [15].

Normalizované skóre S'_{nats} - skóre, ve kterém je zahrnut skórový systém, to nám umožňuje porovnávat výsledky, které byly vypočítány s různým skórovým systémem [21]

$$S'_{nats} = \lambda S - \ln K \tag{8}$$

E-hodnota se z normalizovaného skóre vypočítá:

$$E = mne^{-S'_{nats}} \tag{9}$$

Bit skóre S'_{bits} - je odvozeno od řádkového skóre S, kde je započítáno skóre ze substitučního skóre a ze skóre penalt. Jelikož bit skóre bylo normalizováno a je v něm zahrnut používaný skórový systém, toto skóre se může používat i pro porovnání uspořádaní z různých vyhledávání[18]

$$S'_{bits} = \frac{S'_{nats}}{\ln 2} = \frac{\lambda S - \ln K}{\ln 2} \tag{10}$$

E-hodnota se z bit skóre vypočítá následovně:

$$E = mn2^{-S'_{bits}} \tag{11}$$

Převod mezi E-hodnotou a P-hodnotou:

Jelikož funkce, které implementují algoritmus Blast vrací E-hodnotu, pro větší představu, kdy P-hodnota nám udává pravděpodobnost výskytu nějaké podsekvence se skórem větším než S, můžeme si E – hodnotu převést na P-hodnotu a obráceně dle těchto vzorců [18]:

$$P = 1 - e^{-E} \tag{12}$$

$$E = -\ln(1 - P) \tag{13}$$

5 Aplikace

Hlavním úkolem bylo vytvořit aplikaci, která bude srovnávat zadanou sekvenci nukleotidů nebo proteinů s nukleotidy či proteiny uloženými v tabulkách v databázi. Takové porovnání nám dává informace o podobnosti porovnávaných sekvencí, délce vyhledaných porovnaných sekvencí a další informace popsané v následujících kapitolách.

Aplikace bude využívat již zmíněné algoritmy BLAST, Smith-Waterman a Needleman-Wunsch, které nám vrátí výsledky o procentuální podobnosti, délce nalezených podobných podsekvencí, začátek a konec podsekvencí, skóre, informace o mezerách (jejich umístění v podsekvencích)

Aplikace pracuje s databází Oracle 10g R2, protože je v této databázi algoritmus BLAST již implementován. Nicméně další dva uvedené algoritmy v databázi Oracle již implementovány nejsou, proto databáze je v tomto případě používána pouze pro výběr sekvencí, oproti kterým je zadaná sekvence porovnávána.

5.1 Analýza

Úkolem je vytvořit aplikaci, která bude porovnávat zadanou sekvenci se sekvencemi v databázi. Bude se jednat o WinForms aplikaci. V této aplikaci můžeme zadávat vstupní sekvenci a porovnávat ji se sekvencemi umístěnými v tabulkách v databázi. Porovnávání je na základě vybraného algoritmu:

BLAST – slouží pro lokální vyhledávání

Smith-Waterman – slouží pro lokální vyhledávání

Needleman-Wunsch – slouží pro globální vyhledávání

se zadanými vstupními parametry. Výstupem je graficky znázorněno zarovnání sekvence a dále se zvolené podobné sekvence mohou uložit do formátu XML.

5.1.1 Technologie

Implementace samotné aplikace je v jazyce C# na platformě .NET Framework 2.0 pod Visual Studiem 2005. Za výhody platformy .NET 2.0 a jazyka C# lze považovat:

- Platforma .NET Framework i jazyk C# je založen na objektově orientovaných principech
- V technologii .NET se všechny jazyky (Visual Basic, C#, J#, C++) kompilují do společného jazyka – Intermediate Language. Umožňuje to jazykovou nezávislost(jednotlivé jazyky mohou lépe spolupracovat) a nezávislost na platformě
- Dobrý přístup k datům uložených v databázi pomocí komponent ADO.NET

- podpora XML
- sdílení kódu mezi aplikacemi pomocí assembly – nahrazují DLL knihovny

Tato aplikace také využívá databáze Oracle 10g Release 2, kde jsou v tabulkách uloženy genomické informace, se kterými se bude porovnávat dotazovaná sekvence. Oracle 10g obsahuje funkce, které za použití algoritmu BLAST toto porovnávání umožňují.

V případě, že chcete využít databáze – tabulky s genomickými informace např. databáze swissprot, můžete si z internetu tyto data stáhnout ve formátu *.dat a pomocí Perl skriptu a SQL * Loaderu nahrát do vámi vytvořené tabulky. Postup naleznete v *Příloze č.7.*

5.1.2 Vstupní parametry

Vstupními parametry aplikace je vstupní sekvence a vstupní parametry samotných algoritmů (skóre za shodná písmena, za různá písmena, mezeru ,...)

5.1.2.1 Vstupní sekvence

Vstupní sekvence se zadává ze souboru *.fasta, tedy tato sekvence je ve formátu fasta, nebo samotnou sekvenci lze vepsat do textového pole. Sekvence ve fasta formátu začíná řádkem s popisem a následně samotnou sekvencí, která je již na dalším řádku. Řádek, kde je sekvence popsána, začíná znakem ”>”. Každý soubor fasta obsahuje právě jednu sekvenci.

Aminokyseliny, které se mohou použít ve vstupní sekvenci, jsou popsány v *tabulce č.3*. Soubor ve fasta formátu s aminokyselinami může vypadat následovně:

```
>SEQUENCE_1
MTEITAAMVKELRESTGAGMMDCKNALSETNGDFDKAVQLLREKGLGKAAKKADRLAAEG
LVSVKVSDDFTIAAMRPSYLSYEDLDMTFVENEYKALVAELEKENEERRRLKDPNKPEHK
IPQFASRKQLSDAILKEAEEKIKEELKAQGKPEKIWDNIIPGKMNSFIADNSQLDSKLTL
```

Nukleotidy, které se mohou použít ve vstupní sekvenci, jsou vyjádřeny:

A	Adenin
U	Uracil
T	Thymin
G	Guanin
C	Cytozin

Soubor ve fasta formátu může vypadat následovně:

```
>SEQUENCE_2
ADGCDAAGDCGDACGCDGCACGACDGCDGCACGACGDACGCDGCACGACGACDGCDG
```

5.1.2.2 Parametry algoritmů

Parametry algoritmů se liší od toho, který algoritmus použijeme. Je to z toho důvodu, že při porovnávání proteinů a nukleotidů algoritmem BLAST se liší systém, který se použije pro vypočítávání skóre, tedy i nejpodobnějšího zarovnání.

Vstupní parametry pro porovnávání protein – protein (včetně přeložených nukleotidů na protein) algoritmem BLAST

`matrix` – použitá substituční matice (BLOSUM62, BLOSUM80, PAM70)

`expect` – práh pro hlášení podobnosti

`open cost gap` – penalta za otevření mezery

`extend cost gap` – penalta za rozšíření mezery

`word` – délka slova

`final x drop off` – konečný pokles o final x drop off – parametr X

Vstupní parametry pro porovnávání nukleotid – nukleotid algoritmem BLAST:

`expect` – práh pro hlášení podobnosti

`open cost gap` – penalta za otevření mezery

`extend cost gap` – penalta za rozšíření mezery

`mismatch` – skóre za neshodná písmena

`match` – skóre za shodná písmena

`word` – délka slova

`final x drop off` – konečný pokles o final x drop off – parametr X

Vstupní parametry pro porovnávání sekvencí algoritmy Needleman-Wunsch a Smith-Waterman:

`match` – skóre za shodná písmena

`mismatch` – skóre za neshodná písmena (záporné)

`gap` – skóre za mezeru (záporné)

`skóre` – minimální hodnota pro skóre nalezené podobné sekvence

5.1.3 BLAST

Algoritmus BLAST je implementován v Oracle 10g ve formě funkcí, kterých lze využít při vyhledávání podobných sekvencí.

Algoritmy které využívá databáze Oracle 10g:

- BLASTN – porovnává nukleotidovou sekvenci s nukleotidovou databází (tabulkou)
- BLASTP – porovnává proteinovou sekvenci s proteinovou databází (tabulkou)

- BLASTX – překládá nukleotidovou sekvenci (RNA) na proteinovou ve všech 6 možných variantách a následně ji porovnává oproti proteinové databázi (tabulce)
- TBLASTN – porovnává proteinovou sekvenci oproti nukletidové (RNA) databázi (tabulce) přeložené na proteinovou
- TBLASTX – převádí nukleotidovou sekvenci (RNA) na proteinovou a porovnává oproti nukleotidové (RNA) přeložené databázi (tabulce)

Algoritmus BLAST je v Oraclu 10g implementován ve formě tabulkových funkcích [23]:

- BLASTN_MATCH – využívá algoritmus BLASTN pro porovnávání nukleotidů, používá skórový systém[14]. Funkce vrací tabulku – *5.1.3.1 Návratová tabulka funkcí xxx_match()*, podrobný popis této funkce naleznete v *Příloze č.1*
- BLASTP_MATCH – využívá algoritmus BLASTP pro porovnávání proteinů, pro počítání skóre používá substituční matice[15]. Funkce vrací tabulku – *5.1.3.1 Návratová tabulka funkcí xxx_match()*, podrobný popis funkce naleznete v *Příloze č.2*
- TBLAST_MATCH – využívá algoritmy BLASTX, TBLASTN, TBLASTX, kde dotazovaná sekvence nebo sekvence obsažena v databázi je překládána z RNA na proteiny. Algoritmus, který bude použit, se zadává jako vstupní parametr této funkce. Ostatní vstupní parametry jsou stejné jako u funkce BLASTP_MATCH – porovnávání proteinů. Podrobné informace o této funkci naleznete v *Příloze č.3*
- BLASTN_ALIGN – využívá algoritmus BLASTN, tato funkce je velice podobná funkci BLASTN_MATCH, veškeré vstupní parametry jsou stejné jako u této funkce. Rozdíl je pouze v návratové tabulce - *viz. 5.1.3.2 Návratová tabulka funkcí xxx_aling()*, podrobné informace o této funkci naleznete v *Příloze č.4*
- BLASTP_ALIGN – využívá algoritmus BLASTP, podobná funkci BLASTP_MATCH především ve vstupních parametrů. Funkce vrací tabulku - *5.1.3.2 Návratová tabulka funkcí xxx_aling()*, podrobné informace o této funkci naleznete v *Příloze č.5*
- TBLAST_ALIGN – využívá algoritmus TBLAST, je velice podobná funkci TBLAST_MATCH v podobě vstupních parametrů, tedy se pro počítání skóre používá substituční matice a zadává se zde typ algoritmu, který se použije, jde o algoritmy, kde se dotazovaná sekvence nebo sekvence v databázi překládá na proteiny, tedy typy algoritmů jsou blastx, tblastn, tblastx. Funkce vrací tabulku - *5.1.3.2 Návratová tabulka funkcí xxx_aling()*. Vrací pouze informace o původních sekvencí, tedy počátek a konec podsekvencí značí

[14] skórový systém – jako vstupní parametry se zadává ohodnocení při stejných písmenech a ohodnocení při různých písmenech.

[15] v Oracle jsou implementovány pouze substituční matice BLOSUM62, BLOSUM80 a PAM70

v případě nukleotidů start a konec podsekvence v sekvenci nukleotidů, podrobnější informace o této funkci naleznete v *Příloze č. 6*

5.1.3.1 Návratová tabulka funkcí xxx_match() algoritmu BLAST

Funkce blastp_match(), blastn_match() a tblast_match() nám vrací tabulku, která obsahuje skóre, očekávanou hodnotu a identifikátor sekvence.

5.1.3.2 Návratová tabulka funkcí xxx_aling() algortimu BLAST

Funkce blastp_align(), blastn_align() a tblast_align() vrací tabulku, která obsahuje informace o počátku a konci nalezených podsekvencí, jejich délce, skóre, expect hodnotě, procentuální identitě, identifikátoru sekvence uložené v databázi, počtu písmen s kladným a záporným ohodnocením a o tabulce gap_list, která nám určuje pozici a délka mezer. Díky těmto informacím lze vytvořit zarovnání sekvencí.

5.1.4 Smith-Waterman

Algoritmus Smith-Waterman porovnává sekvence na lokální úrovni *viz 3.3.2 Smith-Waterman algoritmus.* Tento algoritmus již v databázi Oracle 10g implementován není, proto implementace tohoto algoritmu bude součástí vytvářené aplikace. Tento algoritmus je implementován dle popisu v části *3.3.2 Smith-Waterman algoritmus*.

Pro samotnou implementaci budou potřeba dvě pole, jedno pro uchovávaní vypočítaného skóre a druhé pro zaznamenání ukazatelů na buňky, ze kterých skóre bylo vypočítáno a zvoleno jako maximální skóre. Více informací o samotné implementaci tohoto algoritmu se dozvíte v části *5.2 Implementace.*

Vstupními parametry jsou dotazována sekvence, ohodnocení za stejné písmeno, ohodnocení za různé písmeno, ohodnocení za mezeru a nakonec skóre, které nám určuje, že budou vybrány pouze ty podsekvence, které mají maximální skóre větší než zadané.

Výstup tohoto algoritmu je formou tabulky, která obsahuje dotazovanou sekvenci, podobnou sekvenci nalezenou v databázi, podsekvenci z první sekvence, která dle tohoto algoritmu odpovídá podobné podsekvenci z druhé sekvence z databáze. Obě tyto sekvence, pokud je to nutné, již obsahují mezery a jsou stejně dlouhé. Dále tato tabulka bude obsahovat skóre nalezených zarovnání a procentuální identitu.

5.1.5 Needleman-Wunsch

Algoritmus Needleman-Wunsch je algoritmus, který porovnává sekvence na globální úrovni *viz 3.3.1 Needleman-Wunsch algoritmus.* Tento algoritmus také není v databázi Oracle 10g implementován.

Proto implementace je obsahem vytvářené aplikace. Implementovaný algoritmus je dle popsaného algoritmu Needleman-Wunsch v části *3.3.1 Needleman-Wunsch algoritmus*. Podobá se algoritmu Smith-Waterman. Pro implementaci budou použity dvě pole pro uložení skóre a pro uložení ukazatelů na buňky, odkud bylo maximální skóre vypočítáno.

Vstupními parametry jsou dotazovaná sekvence, ohodnocení při shodných písmenech, ohodnocení při různých písmenech, ohodnocení za mezeru a minimální skóre. Rozdíl mezi Needleman-Wunsch algoritmem a Smith-Waterman algoritmem je ten, že u Needleman-Wunsch algoritmu se bere maximální skóre z pozice buňky, která je v pravém, dolním rohu, kdežto u Smith-Waterman algoritmu, může být několik stejných maximálních skór na různých pozicích.

Výstupní parametry jsou stejné jako u Smith-Waterman algoritmu, tedy je vrácena tabulka s dotazovanou sekvencí, nalezenou podobnou sekvencí, upravenými sekvencemi s případnými mezerami, dále skóre a index počátku a konce nalezených sekvencí v sekvencích.

5.1.6 Výstup aplikace

Výstupem aplikace jsou nalezené sekvence, které se vykreslují do okna formuláře. Jsou to především obě původní sekvence (vstupní a cílová) celé délky, zarovnané sekvence, které mohou obsahovat mezery, a vypočítané skóre. V okně bude vykreslen i čas, který byl potřeba pro prohledávání tabulek zvolenými algoritmy.

Vybrané nalezené nebo veškeré podobné sekvence se mohou uložit do formátu XML. Do tohoto souboru se ukládají obě původní sekvence a obě sekvence, které jsou již upraveny dle výstupních parametrů algoritmu BLAST nebo přímo vráceny algoritmem Smith-Waterman či Needleman-Wunsch. Do souboru XML se ukládá i vypočítané skóre.

5.2 Implementace

Aplikace je implementována v jazyce C#, což je objektově orientovaný jazyk. Jedná se o MDI (Multiple Document Interface) aplikaci, to znamená, že aplikace může zobrazovat několik instancí stejného typu formuláře. Ty nám dále slouží pro samotné zadávání vstupní sekvence, parametrů a vyhledávání dle zvoleného algoritmu.

5.2.1 Návrh tříd

Jazyk C# je objektově orientovaný. To znamená, že hlavními stavebními bloky jsou objekty a třídy. Třídy implementovány v aplikaci

- `Program` – obsahuje metodu main, spouští a zobrazuje hlavní okno - `MainWindow`
- `MainWindow : Form` – hlavní okno aplikace, dědí od třídy `Form`, *viz. 5.2.2.1 Třída MainWindows – hlavní okno aplikace*
- `BLASTNP : Form` – dceřinné okno hlavního okna, dědí od `Form`, určeno pro porovnávání protein – protein, nukleotid – nukleotid, *viz. 5.2.2.2 Třída BLASTNP*
- `TBLAST : Form` – dceřinné okno hlavního okna, dědí od `Form`, určeno pro porovnávání: přeložený nukleotid – protein, protein – přeložený nukleotid, přeložený nukleotid – přeložený nukleotid, *viz. 5.2.2.3 Třída TBLAST*
- `Vysledky : Form` – dceřinné okno hlavního okna, zobrazuje výsledky vyhledávání, dědí od `Form`, *viz. 5.2.2.7 Třída Vysledky - Zobrazování výsledků*
- `Algoritmy` – třída obsahující funkce, které jsou dále zděděny třídou `Needleman_Wunsch` a `Smith_Waterman`, tyto funkce které algoritmy používají jsou pro oba algoritmy shodné
- `Needleman_Wunsch : Algoritmy` – třída pro porovnávání dvou sekvencí pomocí algoritmu Needleman-Wunsch, dědí některé funkce od třídy `Algoritmy`, *viz. 5.2.2.5 Třída Needleman_Wunsch*
- `Smith_Waterman : Algortimy` – třída pro porovnávání dvou sekvencí pomocí algoritmu Smith-Waterman, dědí některé funkce od třídy `Algoritmy`, *viz. 5.2.2.6 Třída Smith_Waterman*
- `Data` – třída využívající technologie ADO.NET, slouží pro přístup do databáze, *viz.5.2.2.4 Třída Data - Přístup k databázi*
- `Sekvence` – pomocná třída pro třídy `Smith_Waterman`, `Needleman_Wunsch` a `Data`, obsahuje statistiky a informace o porovnávaných sekvencí zmíněnými třídami
- `Vyjimky` – třída pro výjimky, dle kódu výjimky určuje jaká zpráva bude uživateli vypsána, obsahuje přetěžovanou statickou metodu pro rozpoznání výjimky s parametrem OracleException a Exception
- `XMLAmino` – třída, kterou využívá pouze třída `Vysledky`, používá se při překladu nukleotidové sekvence na protein, kodony jsou vyjádřeny v souboru Amino.xml, a pro uložení určených sekvencí do souboru *.xml
- `IO` – třída zajišťující načítání vstupní sekvence

5.2.2 Třída MainWindows – hlavní okno aplikace

Třída `MainWindow` dědí od třídy `Form`, která obsahuje ovládací prvky(button, textBox, ..) a jejich metody a vlastnosti. Tato třída je velice rozsáhlá, vesměs slouží k vytvoření uživatelského rozhranní aplikace.

Tedy třída `MainWindow` slouží pro zobrazení hlavního okna aplikace (okna MDI Parent). Obsahuje nabídky, dle kterých se vytváří a zobrazují další okna (dceřinná), ty jsou vyjádřeny ve třídě `BLASTNP` a `TBLAST`.

5.2.3 Třída BLASTNP

Třída `BLASTNP` opět dědí od třídy `Form`, to znamená, že slouží pro zobrazení uživatelského rozhranní. Tato třída se vytváří, pokud uživatel chce porovnávat sekvence typu protein – protein nebo nukleotid – nukleotid. Obsahuje ovládací prvky pro zadávání vstupní sekvence, která se může zadávat i ze souboru – využitím třídy `IO`. Dále obsahuje prvky pro zadávání vstupních parametrů, výběru tabulky, ve kterých jsou umístěny cílové sekvence, a možnost přepnutí algoritmu, který chceme pro porovnání použít (Blast, Smith-Waterman, Needleman-Wunsch). Při zmáčknutí tlačítka `Vyhledat` se dále volají metody ze třídy `Data`.

5.2.4 Třída TBLAST

Třída `TBLAST` slouží pro zobrazení uživatelského rozhranní. Dědí od třídy `Form`. Třída `TBLAST` se vytváří, pokud uživatel chce porovnávat sekvence typu přeložený nukleotid – protein, protein-přeložený nukleotid nebo přeložený nukleotid – přeložený nukleotid. Obsahuje ovládací prvky pro zadávání vstupní sekvence, která se může zadávat i ze souboru – využitím třídy `IO`. Dále obsahuje prvky pro zadávání vstupních parametrů, výběru tabulky, kde jsou umístěny cílové sekvence, a možnost přepnutí algoritmu, dle toho co chceme porovnávat (BLASTX, TBLASTN, TBLASTX). Při zmáčknutí tlačítka `Vyhledat` se dále volají metody ze třídy `Data`.

5.2.5 Třída Data - Přístup k databázi

Pro přístup do databáze se používá technologie ADO.NET. Její výhodou je použitelnost v odpojených aplikacích a dále přístup k datům v různých datových zdrojích. ADO.NET je vlastně soubor typů(tříd, výčtů, ...), které můžeme použít pro přístup k datům.

Tyto typy se nacházejí ve jmenných prostorech `System.Data`, který nám poskytuje přístup k datům ve formě `DataSet` a `DataTable`. V našem případě ještě potřebujeme jmenný prostor `Oracle.DataAccess.Client`, který nám poskytuje třídy jako `OracleConnection`, `OracleDataAdapter`, `OracleException` a další.

Přístup k databázi je možný dvěma způsoby, každý má své výhody a nevýhody, nicméně v našem případě budeme používat přístup, kdy se pracuje v offline režimu.

Přístupy:

- **připojená aplikace k databázi**

  aplikace má s danou databází vždy aktivní připojení, využívá se to především v případech, kdy se data v databázi rychle mění a my potřebujeme v aplikaci aktuální verzi dat

- **odpojená aplikace od databáze**

  V tomto případě aplikace získá data a uloží si je do lokální cache klienta, to nám umožní rozhranní `OracleDataAdapter` a `Dataset`, dále se od databáze můžeme odpojit. Samotný `Dataset` představuje třída, kam se data z databáze ukládají. Do jednoho `DataSetu` můžeme uložit i několik tabulek (`Tables`). Toto má velkou výhodu v tom, že síť je minimálně zatížena, nicméně tento přístup má také velkou nevýhodu a tím jsou konflikty mezi uživateli, kdy jeden uživatel chce smazat řádek, který již byl smazán jiným uživatelem

Naše aplikace přistupuje k databázi pouze ve formě výběru řádků pomocí příkazu select, není zde možné data do databáze vkládat či je upravovat. Proto z tohoto důvodu a dle popisu režimů uvedených výše jasně vyplývá, že v našem případě je lepší použít režim, kdy se bude pracovat s daty v offline režimu, tedy kdy aplikace nebude připojena k databázi.

Pro získávání dat z databáze se používají tyto třídy:

`OracleConnection` – zajišťuje spojení s databází, v této třídě se uvádí `ConnectionString`, což je jméno, heslo a data source, vlastní spojení se otevře metodou `Open` a zavře metodou `Close`

`OracleCommand` – třída pro zápis SQL dotazu, lze využít parametrizované dotazy

`OracleDataAdapter` – třída představující spojení do databáze formou čtyř příkazů[16], ze kterých aplikace používá pouze `SelectCommand`, dále tato třída má metodu `Fill()`, která naplní `DataSet` řádky z databáze pomocí `OracleCommand`, kde je uložen příslušný SQL dotaz

`DataSet` – třída, která poskytuje lokální zpracování dat

5.2.5.1 SQL dotazy

Při použití vytvořených algoritmů Smith-Waterman a Needleman-Wunsch se pouze vybírají všechny záznamy ze zvolené tabulky. Pro každý záznam – sekvenci se následně volá třída `Smith_Waterman` či `Needleman_Wunsch`, která nám reprezentuje porovnávání sekvencí a poskytuje nejlepší zarovnání dle zvoleného skóre a další statistické údaje.

Nicméně něco jiného je to u algoritmu Blast, který je již v databázi Oracle 10g implementován. Protože funkce, které reprezentují algoritmus Blast, nám vrací tabulku, která v sobě obsahuje další

[16] `SelectCommand` – vybrání řádků z databáze, `UpdateCommand` – uložení změn, `DeleteCommand` – smazání řádků, `InsertCommand` – vložení nových řádků

vnořenou tabulku, je nutné veškeré údaje získat pomocí dvou příkazů. Jeden nám slouží pro vybrání řádků, které obsahují normální datové typy a druhý nám slouží pro vybrání veškerých řádků, které jsou ve všech vnořených tabulkách.

Příklad SQL dotazu:

1. SQL dotaz, který nám vybere všechny řádky vracející funkce BLASTP_ALIGN, které obsahují sloupce s obvyklými datovými typy

```
select t_seq_id, query_string, seq_data,
pct_identity,alignment_length,mismatches, positives, gap_openings,
q_seq_start, q_frame, q_seq_end, t_seq_start, t_seq_end, t_frame, score
from
TABLE(BLASTP_ALIGN (
(select sequence from query_db where sequence_id = 1),
CURSOR(SELECT seq_id, seq_data FROM zkusebniP ))t1 ,
(select sequence query_string from query_db where sequence_id = 1),
zkusebniP t2
where t2.seq_id=t1.t_seq_id AND t1.score > 25;
```

2. SQL dotaz, který nám vybere veškeré řádky všech vnořených tabulek gap_list, které nám vrací funkce BLASTP_ALIGN

```
select t2.* from TABLE
(BLASTP_ALIGN (
(select sequence from query_db where sequence_id = 1),
 CURSOR(SELECT seq_id, seq_data FROM zkusebniP)))t1,
TABLE(t1.gap_list)t2  where t1.score > 25;
```

Jelikož tyto tabulky nejsou provázány, jak by tomu bylo při použití tabulek obsahující primární a cizí klíč, je nutné toto propojení dodělat. 1. SQL dotaz nám vrací počet otevřených mezer (gap_openings) z toho lze odvodit, kolik řádků z tabulky, kterou nám vrací 2. SQL dotaz, patří k řádku 1. tabulky, kde je informace o daném počtu mezer. Tedy 1. tabulce se přidají indexy, které budou unikátní, tudíž je zde jistá podobnost s primárním klíčem. A 2. tabulce se přidá sloupec s čísly, které odkazují na indexy 1. tabulky, opět si pod tímto můžeme představit cizí klíč. Veškerá práce s databází je obsažena ve třídě `Data`. Veškeré metody obsaženy v této třídě jsou statické, to znamená, že je lze volat, i když neexistuje žádný objekt třídy `Data` – nepracujeme s konkrétní instancí třídy.

5.2.6 Třída Needleman_Wunsch

Algoritmus Needleman-Wunsch je algortimus, který prohledává sekvence a vytváří nejpodobnější zarovnání na úrovni globální. To znamená, že když si zvolíme nějaké minimální skóre, tak výsledné podobné sekvence budou mít skóre všechny stejné a to takové, které je větší nebo rovné zvolenému minimálnímu skóre.

Tento algoritmus je vyjádřen ve třídě `Needleman_Wunsch`. Základem tohoto algoritmu jsou dvě matice. V první je vyjádřeno skóre a ve druhé je vyjádřeno, ze kterých sousedních buňek(levá, horní nebo levo-horní buňka) se skóre vypočítalo. Počítání skóre je uvedeno ve vzorci *(1) (Výpočet skóre pro algoritmus Needleman-Wunsch)*

Implementaci algoritmu můžeme rozdělit do několika částí, které současně tvoří nejdůležitější metody třídy `Needleman_Wunsch`(pouze metoda `Inicializace()` je obsažena ve třídě `Algoritmy`) a to na:

5.2.6.1 Metoda Inicializace()

V této metodě se v první matici, kam se zadává skóre, inicializuje první řádek a první sloupec, kde každé další skóre v řádku se počítá takto: M[x-1, 0] + gap a ve sloupci takto: M[0, y-1] + gap, hodnota M[0,0] je 0.

Druhá matice N, kde se ukládají ukazatele na buňky, odkud bylo skóre vypočítáno, se inicializuje následovně: první řádek - do buněk se uloží řetězec "--l", první sloupec – do buněk se uloží "-u-", buňka N[0,0] = "---" .

Pokud je skóre v buňce vypočítáno z některého směru, je písmeno, které toto značí, obsaženo v řetězci, jinak je zde "-". Pro buňku, která se nachází šikmo – vlevo nahoře, je označení písmenem "d", pro buňku nalevo je to "l" a pro buňku nahoře je to "u"

5.2.6.2 Metoda Naplneni()

V této metodě dochází k počítání skóre dle vzorečku, který je uveden ve vzorci *(1) (Výpočet skóre pro algoritmus Needleman-Wunsch)* a zároveň je naplněna druhá matice pointerů. Dále se zde zaznamenává skóre, což je skóre levé, dolní buňky. Pokud je toto skóre menší než uživatelem zvolené, tak funkce vrátí hodnotu `false`, jinak vrátí hodnotu `true`. Navrácená hodnota `false` znamená, že metoda, která by následovala – `Prochazeni()` se neprovede a aplikace vypíše upozornění, že nebyly vybrány žádné řádky.

5.2.6.3 Metoda Prochazeni()

Tato metoda je volána po naplnění obou matic a ověření, zda skóre vyhovuje zadanému skóre uživatelem. Jelikož předem nevíme, kolik podobností sekvencí nám vyjde z obou matic, řeší se to tím způsobem, že po každém posunu se obě zpracované sekvence uloží do `ArrayListu` (pole, do kterého se mohou ukládat objekty) i s hodnotami souřadnic, kde se skončilo.

Kam se při zpátečním průchodu posuneme, řeší druhá matice s řetězci reprezentující ukazatele. To znamená, že pokud jedna buňka obsahuje hodnotu "d-u", původní rozpracované sekvence se zdvojí, kdy jedna se posune ve směru nahoru a druhá ve směru nahoru doleva. Pro ukládání je vytvořena nová třída `Sekvence`, která obsahuje rozpracované sekvence a souřadnice x a y.

Tato třída je dále ukládána do `ArrayListu`. Odkud je neustále načítána, dokud hodnoty x a y nebudou rovny 0. To znamená, že konec průchodu je v levém horním rohu. A začátek průchodu je v pravém dolním rohu. Hodnota v této buňce je brána i jako skóre, které se porovnává se zadaným skórem.

Dále se ještě používá metoda, která nám výsledné podobné sekvence zpracuje tak, že uloží informace např. o délce sekvencí, podobnosti (v procentech), kolik je otevřených mezer, stejných písmen, různých písmen a další a ty dále předá třídě `Data`.

5.2.7 Třída Smith_Waterman

Algoritmus Smith-Waterman je algoritmus, který porovnává sekvence na úrovni lokální, to znamená, že výsledkem je zarovnání obsahující pouze části sekvencí. Pokud si zvolíme minimální skóre, budou vyhledány všechny podsekvence, které mají maximální skóre větší nebo rovno než námi zadané.

Tento algoritmus je vyjádřen ve třídě `Smith_Waterman`. Využívají se zde dvě matice. Jedna je pro uchování skóre a druhá pro vyjádření, ze kterého směru se skóre vypočítalo. Tento algoritmus je velice podobný algoritmu `Needleman-Wunsch`, tudíž stejně jako u tohoto algoritmu se používá pro vyjádření, ze kterého směru se dané skóre vypočítalo, řetězec tří znaků: "d" (levá horní buňka), "l" (levá buňka), "u" (horní buňka), pokud z některého z uvedených směrů skóre nebylo vypočítáno, je toto písmeno nahrazeno znakem "-". Skóre se počítá dle vzorce *(2): Výpočet skóre pro algoritmus Smith-Waterman*

Tato třída obsahuje dvě základní metody: `Naplneni()` a `Prochazeni()`, metoda `Inicializace()` je ve třídě `Algoritmy`.

5.2.7.1 Metoda Inicializace()

V této funkci se v první matici, která obsahuje skóre, nastaví hodnoty prvního řádku a prvního sloupce na 0, což také odpovídá vzorci *(2): Výpočet skóre pro algoritmus Smith-Waterman*. Tedy maximum ze všech možností je nula.

Ve druhé matici, která obsahuje řetězce, jenž plní funkce ukazatele, ze které buňky bylo dané skóre vypočítáno, se první sloupec a první řádek vyplňovat nemusí. Je to z důvodu, že při zpětném procházení, samotné procházení končí, pokud se narazí na buňku, která má skóre 0.

5.2.7.2 Metoda Naplneni()

V této funkci dochází k počítání skóre, dle vzorce *(2): Výpočet skóre pro algoritmus Smith-Waterman* a vyplňování obou matic. Do první matice se vyplní samotné skóre a do druhé řetězec písmen, které označují, ze kterého směru bylo dané skóre vypočítáno. Do pomocné třídy se také zaznamenávají veškerá skóre, která jsou maximální a zároveň jsou větší než uživatelem zadané, a souřadnice x a y buňky, kde se tyto skóre nacházejí.

5.2.7.3 Metoda Prochazeni()

Tato metoda je velice podobná metodě `Prochazeni()` ve třídě `Needleman_Wunsch`. To znamená, že se opět prochází maticí, ve které jsou písmena reprezentující ukazatel na buňku, ze které bylo skóre vypočítáno. Dílčí podsekvence (písmene, později řetězce) se ukládají do pomocné třídy i se souřadnicemi, kde se skončilo.

Rozdíl v této metodě a metodě `Prochazeni()` ve třídě `Needleman_Wunsch` je takový, že procházení začíná u buňky, která má skóre maximální ze všech skór a je zároveň větší než uživatelem zadané. Toto skóre je uloženo do pomocné třídy, což bylo provedeno v metodě `Naplneni()`. Tato třída obsahuje souřadnice, kde se skóre nachází a jeho hodnotu. Procházení maticí končí ve chvíli, kdy se narazí na buňky, jejichž skóre je nula. Toto procházení maticí se děje pro všechny buňky (skóre, souřadnice x a y), které byly uloženy do pomocné třídy v metodě `Naplneni()`.

5.2.8 Třída Vysledky - Zobrazování výsledků

Třída `Výsledky` dědí od třídy `Form` *viz popis u 5.2.2.1 Třída MainWindow*. Výsledky se zobrazují přímo ve `WinForms` okně. Veškeré údaje jakožto skóre, délka sekvencí, počet otevřených mezer a další jsou zobrazeny pomocí komponenty `DataGridView`. Při posunu na další řádek se nám pod touto tabulkou vykreslí zarovnané sekvence a zároveň i celá dotazovaná sekvence se sekvencí, vůči které byla porovnávána.

Funguje to tím způsobem, že je vytvořen přetížený konstruktor, který má vstupní parametry `DataSet`, ten obsahuje výsledná data a string, který nám označuje jaký algoritmus byl použit. Dále se dle dat – začátek sekvence, konec sekvence a počet otevřených mezer (u algoritmu Blast se používá tabulka gap_list, která označuje kde mezera začíná a jak je dlouhá), počítá výsledné zarovnané zobrazení sekvencí.

Jak bylo napsáno výše, při posunu v `DataGridView` na další řádek se vykreslí další sekvence. Toto je ošetřeno tím způsobem, že veškeré vykreslování náleží metodě `OnPaint()`. Tato metoda se volá při každé změně velikosti okna, při posunutí scrollbarem,... Nicméně již zmíněné posunutí na další řádek v `DataGridView`, je ošetřeno tak, že voláním odpovídající metody se plocha zneplatní. To znamená, že se zavolá funkce `Invalidate()` a ta zapříčiní vyvolání metody `OnPaint()`.

5.2.8.1 Úprava sekvencí

Jak již bylo naznačeno výše, třída `Vysledky` dostává informace: první sekvence, druhá sekvence, začátek a konec první sekvence a druhé sekvence (podobná část), tabulku s mezerami, délku podobné sekvence, t_frame, q_frame – výstupní parametry funkcí Blast
Výsledná zarovnaná sekvence se vytváří následovně

- vložením mezer na pozice dle tabulky s mezerami
- ořízntutím sekvence dle informací o počátku a konce, pokud je q_frame(první sekvence) nebo t_frame(druhá sekvence) záporný, sekvence se převrací
- porovnáním jednotlivých písmen pro správné vykreslení shodnosti písmen

Pokud se používá funkce TBLAST, což znamená, že alespoň jedna sekvence se musí překládat, délka podobné sekvence je myšlena jako délka konečné proteinové podobné sekvence. To znamená, že délka nukleotidové sekvence je 3x delší. Dále pro lepší představu nalezených sekvencí tímto algoritmem je vytvořena třída `XMLAmino`, která překládá jednotlivé kodony na aminokyseliny. A následně jednotlivé aminokyseliny jsou vykresleny a porovnány s druhou proteinovou sekvencí.

5.3 Testování

V této části naleznete testy algoritmů, jejich vstupních parametrů a porovnání výsledků. Testovány zde budou algoritmy či funkce Blast (funkce BlastP_align, BlastN_align, TBlast_align), dále Smith-Waterman a Needleman-Wunsch

5.3.1 BLASTP (protein - protein)

Při testování tohoto algoritmu, tedy za použití funkce blastp_align(), vytvořenou aplikací bylo zjištěno, že databáze Oracle má implementovány pouze substituční matice BLOSUM62, BLOSUM80 a PAM70. A pro tyto substituční matice jsou možné pouze následující vstupní parametry:

	BLOSUM62	***BLOSUM80***	***PAM70***
wordsize	0, 2 nebo 3	0, 2 nebo 3	0, 2 nebo 3
open cost gap	0 nebo 9 – 13	0 nebo 9 – 11	0 nebo 9 -11
extend cost gap	0 - 2	0 - 1	0 - 1

Při zadání jiných parametrů než výše popsaných vznikne výjimka: neplatný vstupní parametr. Dále při délce slova (wordsize) 2 musí být sekvence dlouhá alespoň 4 a při délce slova 3 musí být délka vstupní sekvence minimálně 6. Opět při zadání vstupní sekvence, která nesplňuje minimální délku, vznikne výjimka.

5.3.1.1 Příklad

porovnejme sekvence GHADGHADGHGAD a GHADFRADGFRHGAD, kde první sekvence je dotazovaná a druhá sekvence je umístěna v databázi, jako vstupní parametry mějme:

wordsize = 3

open cost gap = 11

extend cost gap = 1

funkce blastp_align() nalezla pouze jedno zarovnání při všech zmíněných substitučních matic zadaných jako parametr funkce:

```
G H A D G H A D G - - H G A D
| |   |   |     |   |   |     |   |   |
G H A D F R A D G F R H G A D
```

procentuální identita je 73,333%, což je poměr mezi počtem stejných (odpovídajících si) písmen a celkové délky zarovnání

	BLOSUM62	BLOSUM80	PAM70
skóre	48	50	47

Skóre u matice BLOSUM62 je vypočítáno jako součet hodnot z dané substituční matice u písmen, dále se výsledné skóre počítá:

$$S = skoreZeSubstitucniMatice - pocetMezer * openCostGap$$

$$- \sum_{i=1}^{pocetMezer} delkaMezery[i] * extendCostGap$$

v našem případě je skóre následující:

$$S = 61 - 1 * 11 - (2 * 1) = 48$$

Hodnota expect se mění podle sekvencí, které jsou uloženy v databázi a jsou podobné dotazované sekvenci. Pokud v databázi byla obsažena pouze jedna sekvence, která byla podobná dotazované, expect hodnota byla řádově 0,00002, což znamená že sekvence je velice významná (E – hodnota je nízká), při převodu na P-hodnotu to je 0,00002 (P-hodnota udává pravděpodobnost výskytu dané sekvence) *viz 4.2.2 Statistika*

Při výskytu více sekvencí v databázi, které byly podobné dotazované sekvenci, E-hodnota se zvětšila (řádově při přidání 3 sekvencí se E-hodnota zvětšila 10x), což znamená, že nalezená sekvence již není tolik významná.

5.3.2 BLASTN (nukleotid-nukleotid)

Při testování funkce blastn_align() bylo zjištěno, že vstupní parametry funkce mohou být pouze následující:

expect: větší jak 0

open, extend cost gap, match, x dropoff: nesmí být záporné číslo

mismatch: musí být záporné číslo

wordsize: větší jak 7

Při zadání jiných parametrů, než výše popsaných vznikne výjimka, která nám hlásí číselné přetečení (v případě záporného open, extend cost gap, match nebo x dropoff), či další, které již přímo popisují vzniklou výjimku.

5.3.2.1 Příklad č.1

mějme porovnat dvě sekvence TGATACGCTAAATCCGGGATAGAGCGCATGGCCAC a ATGTTACGCTAGATCCGGACGATAGAGCGCAATTGGCCAC, první sekvence je dotazovaná a druhá sekvence je umístěna v databázi

porovnáváme se vstupními parametry:

wordsize: 7

open cost gap: 3

extend cost gap: 1

mismatch: -1

match: 1

score: 0

výsledné zarovnání je následující:

```
T G A T A C G C T A A A T C C G G - - G A T A G A
| |   | | | | | | | |   | | | | | | |   | | | | | |
T G T T A C G C T A G A T C C G G A C G A T A G A

G C G C A - - T G G C C A C
| | | | |   | | | | | | |
G C G C A A T T G G C C A C
```

procentuální identita je 84,6154%, což je poměr mezi počtem shodných písmen a celkové délky (33/39*100),

dále skóre je 21, z čehož plyne, že skóre S se počítá následovně:

$$S = pocetShodnychPismen * match - pocetMezer * openCostGap - \sum_{i=1}^{pocetMezer} delkaMezery[i] * extendCostGap - pocetNeshodnychPismen * mismatch$$

v našem případě se S počítá následovně:

$$S = 33*1 - 2*3 - (2*1 + 2*1) - 2*1 = 21$$

E-hodnota je nyní řádově 0,00000004, což znamená, že nalezená zarovnaná sekvence je významná.

5.3.2.2 Příklad č.2

Funkce blastn_align() porovnává nukleotidy i dle principu komplementarity. Porovnejme např. tyto sekvence: TACGACGACACGCGAC a GTCGCGTGACGTCGTACGTCACGTCACCACCAAGT
Vstupní parametry:

mismatch: -3

match: 1

wordsize: 7

Výsledné zarovnání je následující:

```
C A G C G C A C A G C A G C A T
| | | | | | | |   | | | | | | |
G T C G C G T G A C G T C G T A
```

Výstupní parametry:

procentuální identita: 93,75	pozitives: 15
q_frame: -1	mismatches: 1
t_frame: 1	skóre: 12

Jelikož q_frame je záporné dotazovaná sekvence se otáčí a zarovnává se dle principu komplementarity. Skóre se počítá obdobně jako u *5.3.2.1 Příklad č.1 BLASTN*

5.3.3 TBLAST

V databázi Oracle je implementována funkce tblast_align(), která dle zadaného parametru `type` (blastx, tblastn, tblastx) porovnává přeložený nukleotid na protein s proteinem nebo s přeloženým nukleotidem na protein *viz. 5.1.2 Blast.* Proto většina vstupních parametrů je shodných s funkcí blastp_align().
V Oraclu jsou obsaženy pouze substituční matice BLOSUM62, BLOSUM80, PAM70. A pro tyto matice jsou možné pouze tyto parametry:

	BLOSUM62	***BLOSUM80***	***PAM70***
wordsize	0, 2 nebo 3	0, 2 nebo 3	0, 2 nebo 3
open cost gap	0 nebo 9 – 13	0 nebo 9 – 11	0 nebo 9 -11
extend cost gap	0 - 2	0 - 1	0 - 1

Při zadání jiných vstupních parametrů se vyvolá výjimka.

5.3.3.1 Příklad č.1 - BLASTX

Porovnejme algoritmem blastx tyto sekvence:
GGACACGCAAACGCAGCAAACCACGGAGGACACGCAAACGGACACGGA
a GHANGANHGGHANGANHG, kde první sekvence je sekvence nukleotidů a druhá je sekvence proteinů. Vstupní parametry jsou:

open cost gap: 9

extend cost gap: 1

word: 3

Funkce nám vrátila tyto informace:

- začátek nalezené podobné sekvence je 1 a konec 48, to znamená, že nalezená podobná sekvence je celá vstupní sekvence
- začátek cílové sekvence je 1 a konec je 18, což je také celá sekvence
- t_frame a q_frame jsou kladné – sekvence se nebudou otáčet
- gap_openings = 1 – byla nalezena jedna mezera
- pozitives = 15 – shodných písmen, jejichž skóre ze substituční matice je větší jak 0, je 15
- mismatches = 2 - počet neshodných písmen (skóre ze substituční matice je menší nebo rovno 0) + počet mezer, kromě počáteční mezery, která úsek mezer otevírá
- pct_identity = 83,333, což je poměr pozitives/délka podobné sekvence (15/18*100)

Vstupní sekvence se musí překládat, to znamená, že se postupně překládá trojice písmen na aminokyselinu dle popisu pro kódování aminokyselin *viz. 1.4 Aminokyseliny*

Výsledné zarovnání:

```
G H A N A A N H G G H A N G - - H G
| | | |   | | | | | | | | |     | |
G H A N G A N H G G H A N G A N H G
```

5.3.3.2 Příklad č.2 – TBLASTX

Porovnejme tyto sekvence nukleotidů pomocí algoritmu tblastx, který nukleotidy překládá na proteiny:
GGACACGCAAACGGAGCAAACCACGGAGGACACGCAAACGGAGCAAACCACGGA a
GGACACGCAAACGGAGCAAACCACGGAGGACACGCAAACGGAGCAAACCACGGA

Výsledkem je mnoho úspěšných porovnání, kde z největším skórem nám vyšlo toto zarovnání:

- q_seq_start: 53 – začátek první porovnávané sekvence
- t_seq_start: 53 – začátek druhé porovnávané sekvence
- q_seq_end: 3 – konec první porovnávané sekvence
- t_seq_end: 3 – konec druhé porovnávané sekvence
- q_frame: -2 – rámec, určující posunutí počátku porovnávání a směr, u dotazované sekvence
- t_frame: -2 - rámec, určující posunutí počátku porovnávání a směr, u cílové sekvence

Záporné q_frame t_frame znamená, že se sekvence budou otáčet, tedy sekvence, které se dále překládají na proteiny jsou:
GGCACCAAACGAGGCAAACGCACAGGAGGCACCAAACGAGGCAAACGCACA
GGCACCAAACGAGGCAAACGCACAGGAGGCACCAAACGAGGCAAACGCACA
Výsledné zarovnání s přeloženými sekvencemi:

```
G T K R G K R T G G T K R G K R T
| | | | | | | | | | | | | | | |
G T K R G K R T G G T K R G K R T
```

Výsledné skóre za použití matice BLOSUM62 je 115. Toto skóre neodpovídá součtu jednotlivých skór párů písmen, jak tomu bylo u funkce blastp_align(). Součet jednotlivých skór párů písmen je:

6 + 4 + 5 + 5 + 6 + 5 + 5 + 4 + 6 + 6 + 4 + 5 + 5 +6 + 5 + 5 + 4 = 86

5.3.4 Smith-Waterman

Algoritmus Smith-Waterman je implementován přímo v aplikaci. Ideální vstupní parametry jsou:

match: kladné číslo

mismatch: záporné číslo

gap: záporné číslo

skóre: větší než 1

při hodnotách jiných než výše doporučených nevznikne výjimka, ale je možné, že výpočet bude dlouho trvat.

5.3.4.1 Příklad č. 1

Porovnejme sekvence GTGCGTATACCTGA a GTACGTATATGCCTGA, kde první sekvence je vstupní sekvence a druhá sekvence je umístěna v databázi
vstupní parametry jsou:

match: 1

mismatch: -1

gap: -2

výsledné zarovnání je následující:

```
G T G C G T A T A - - C C T G A
| |   | | | | | |       | | | | |
G T A C G T A T A T G C C T G A
```
výsledné skóre je 8

V tomto případě je skóre S počítáno následovně:

$$S = pocetShodnychPismen * match + pocetNeshodnychPismen * mismatch + pocetMezer * gap$$

tedy výsledné skóre vychází:

$$S = 13 * 1 - 1 * 1 - 2 * 2 = 8$$

5.3.4.2 Příklad č. 2

porovnejte tyto sekvence GTGCGTATACCTGA a GTACGTATATGCCTGA algoritmem Smith-Waterman a Blastp_align()

Výsledné zarovnání Smith-Watermanem je:

```
G T G C G T A T A - - C C T G A
| |   | | | | | |       | | | | |
G T A C G T A T A T G C C T G A
```

a funkcí Blastp_align():

```
G T G C G T A T - - A C C T G A
| |   | | | | | |       | | | | |
G T A C G T A T A T G C C T G A
```

Jak již bylo napsáno v kapitole *3.3.3*, která popisovala algoritmy Smith-Waterman a Blast, i tento příklad nám potvrzuje, že algoritmus Smith-Waterman je více citlivý a přesný. Sice v tomto případě algoritmus Blast používá substituční matici a algoritmus Smith-Waterman skórový systém, nicméně by tento fakt neměl mít vliv na výsledné zarovnání. Zvláště pokud podobnost písmen A-G je ohodnocena 0 a podobnost písmen A-A je ohodnocena 4 (při použití matic BLOSUM62). Při změně matice na BLOSUM80 nebo PAM70 se výsledné zarovnání nezmění.

5.3.5 Needleman-Wunsch

Algoritmus Needleman-Wunsch je také implementován v aplikaci, proto doporučené vstupní hodnoty jsou:

match: kladné číslo

mismatch: záporné číslo

gap: záporné číslo

při hodnotách jiných než výše doporučených se může stát, že výpočet bude dlouho trvat.

5.3.5.1 Příklad

mějme vstupní sekvence GTGCGTATACCTGA a GTACGTATATGCCTGA a vstupní parametry:

match: 2

mismatch: -2

gap: -1

score: 5

výsledná zarovnání se skórem 22 je následující:

```
G T G C G T A T A - - C C T G A
| |   | | | | | |       | | | | |
G T A C G T A T A T G C C T G A
```

s procentuální identitou 81,25%

```
G T - G C G T A T A - - C C T G A
| |     | | | | | |       | | | | |
G T A - C G T A T A T G C C T G A
```

s procentuální identitou 76,4706%

```
G T G - C G T A T A - - C C T G A
| |     | | | | | |       | | | | |
G T - A C G T A T A T G C C T G A
```

s procentuální identitou 76,4706%

dále vyzkoušejme změnit vstupní parametry na:

match: 2 – zůstává

mismatch: -1

gap: -2

při této změně bude výsledné pouze jedno zarovnání a to:

```
G T G C G T A T A - - C C T G A
| |   | | | | | |       | | | | |
G T A C G T A T A T G C C T G A
```

se skórem 21 a procentuální identitou 81,25

Skóre je počítáno jako u algoritmu Smith-Waterman tedy:

$$S = pocetShodnychPismen * match + pocetNeshodnychPismen * mismatch + pocetMezer * gap$$

v posledním případě to vychází

$$S = 13 * 2 - 1 * 1 - 2 * 2 = 21$$

5.3.6 Chybové stavy

Jde o stavy, kdy při zadání chybné vstupní sekvence se očekává, že příslušná funkce algoritmu BLAST vrátí výjimku (chybu), ale vrátí chybné údaje.

5.3.6.1 Příklad

Mějme vstupní sekvence aminokyselin: GHADGHADOGHGAD a GHADFRADGFRHGAD, kde evidentně ve vstupní sekvenci je chybný znak O. Nicméně výsledky, které nám vrátí funkce blastp_align() jsou následující:

mismatches: 3	q_seq_id: 1
pozitives: 10	q_seq_end: 12
pct_identity: 71,429 %	t_seq_start: 1
gap_openings: 1	t_seq_end: 14

Zarovnaná sekvence:

```
G H A D G H A D O - - G H G
| | | |       | |
G H A D F R A D G F R H G A
```

Dle zarovnané sekvence by hodnoty mismatches a pozitives měly být následující[17]:

mismatches: 7

pozitives: 6

pct_identity: 42,857 %

nebo by příslušná funkce měla vrátit chybu (výjimku), protože písmeno O není obsaženo v substituční matici, která se používá pro určování jednotlivých skór párů písmen.

Výjimka se vyvolá pouze pokud vstupní sekvence je sestavena pouze z písmen, která nejsou obsaženy v substituční matici (např. OOOXXX). Pokud vstupní sekvence obsahuje alespoň jeden správný znak, výjimka se nevyvolá.

[17] do pozitives se započítávají pouze páry písmen, které mají ohodnocení větší jak 0, stejně tak se započítávají i do mismatches. V tomto případě není žádný pár započítán zároveň v mismatches a pozitives.

6 Závěr

Cílem této práce bylo seznámení se s oblastí biologie, která pracuje s genomickými daty, jejich analýzou pomocí algoritmu BLAST a následné zhodnocení výsledků.

Aplikace, která byla k analýze těchto dat vytvořena, slouží i pro testování samotných uložených funkcí algoritmu BLAST. Je to dáno tím způsobem, že vstupní parametry funkcí nejsou přísně omezeny. Samozřejmě datové typy se kontrolují. Toto mělo za následek zjištění, že při zadání vstupní sekvence, která obsahovala neplatné písmeno (písmeno, které není obsaženo v substituční matici), při porovnávání proteinů vůči proteinům, funkce nevrátila chybu či výjimku, jak by se očekávalo, ale vrátila chybné informace o podobnosti sekvencí.

V aplikaci je také implementován algoritmus Smith-Waterman a Needleman-Wunsch, který slouží pro porovnávání sekvencí. S výsledky z implementovaných algoritmů Smith-Waterman a Needleman-Wunsch můžeme porovnávat výsledky funkcí algoritmu BLAST. V tomto směru je zajímavé, že při porovnávání dvou sekvencí algoritmem Smith-Waterman a funkcí blastp_align(), se ukázal algoritmus Smith-Waterman přesnější, respektive je možné, že funkce blastp_align() nevyhledává podobné sekvence zcela přesně *viz 5.3.4.2 Příklad č.2 algoritmu Smith-Waterman.* Toto nám potvrzuje předpoklad uvedený v popisu jednotlivých algoritmů, že právě na rozdíl od algoritmů Smith-Waterman a Needleman-Wunsch algoritmus BLAST nevyhledá veškeré podobné a ne vždy optimální sekvence. Nicméně při porovnávání vůči databázi, kde je umístěn velký počet sekvencí, je tento algoritmus velice rychlý.

Je to dáno tím, že při porovnávání jedné sekvence s druhou sekvencí ještě nemusí být rychlost algoritmu BLAST patrná, ale pokud se porovnává jedna sekvence vůči tisícům sekvencí v databázi, je již rozdíl v rychlosti velmi znatelný, protože se nemusí porovnávat se všemi sekvencemi, ale pouze s vhodnými sekvencemi vybranými v prvních krocích. Narozdíl od algoritmu BLAST jsou techniky dynamického programování (Smith-Waterman a Needleman-Wunsch) odkázané porovnávat jednu sekvenci za druhou, což je velmi časově náročné a není to mnohdy vůbec realizovatelné, i když víme, že bychom po skončení porovnávání dostali optimální výsledek.

Za další vývoj této práce bych považovala vylepšení aplikace, kde jako vstupní parametr algoritmů Smith-Waterman a Needleman-Wunsch by byla penalta za otevření mezery, penalta za rozšíření mezery a implementace substitučních matic, které by se používaly při porovnávání proteinů vůči proteinům umístěných v databázích (v tabulkách). Také by mohlo být zajímavé implementovat tyto zmíněné algoritmy jako uložené funkce v databázi, což by značně mohlo urychlit porovnávání. Se samotnou implementací substitučních matic by souviselo vykreslování zarovnaných sekvencí, kde by aminokyseliny, které si jsou podobné (mají kladné skóre ze substituční matice), byly označeny "+".

Zajímavé mně také přišly Karlin-Altschul statistiky, které v této práci zaujímají příliš malou část na podrobné vysvětlení. Nicméně tato část by po stránce rozsahu a náročnosti byla jako velice zajímavé hlavní téma nějaké práce.

Literatura

[1] Dan K. Krane, Michael L. Ryamer: Fundamental Concepts of Bioinformatics, ISBN: 0-8053-4633-3, Benjamin Cummings 2003.

[2] *Základy molekulární genetiky.*
Dostupný na URL http://www.gjs.cz/vedy-o-zemi/Ruda/Sbi/1-SB4-zaklady-molekularni-dedicnosti.pdf (duben 2008).

[3] *Genetika, molekulární genetika.*
Dostupný na URL http://genetika.wz.cz/genetika.htm (duben 2008).

[4] *Replikace DNA.*
Dostupný na URL http://www.kbi.zcu.cz/studium/ftp/mobi3.pdf (duben 2008).

[5] *Cesta k DNA, replikace, transkripce, translace*
Dostupný na URL http://www.orko.cz/Biologie%202008/Replikacetranskripcetranslace08.ppt (duben 2008).

[6] *Proteiny, struktura proteinů.*
Dostupný na URL http://cs.wikipedia.org/wiki/Protein (duben 2008).

[7] *DNA, struktura DNA.*
Dostupný na URL http://cs.wikipedia.org/wiki/DNA (duben 2008).

[8] *DNA*
Dostupný na URL http://images2.clinicaltools.com (duben 2008).

[9] *DNA Replication and Synthesis.*
Dostupný na URL http://library.thinkquest.org/C006188/dna.htm (duben 2008).

[10] The Biology Department, UM, Univerzity of Miami, *Department of Biology*
Dostupný na URL http://fig.cox.miami.edu (duben 2008).

[11] *Obrázek průběhu výroby proteinů.*
Dostupný na URL http://ciselniky.dasta.mzcr.cz/hypertext/200620/hypertext/GOAAA_soubory (duben 2008).

[12] Catherine Hearne, Zhan Cui, Simon Parsons and Saki Hajnal, Prototyping and Genetics *Deductive database.* Dostupný na URL http://www.sci.brooklyn.cuny.edu/~parsons/publications/downloads/conferences/ismb94.ps.gz (duben 2008).

[13] Institute for Computer Science, University of Munich, *Knowledge discovery in Large Spatial Database.* Dostupný na URL http://ifsc.ualr.edu/xwxu/publications/kdlsd.pdf (duben 2008)

[14] *GenoSIS: Genome Data Interpretation Using GIS.*
Dostupný na URL http://gis.esri.com/library/userconf/proc02/pap0719/p0719.htm (duben 2008)

[15] Frédérique, G.: The Fasta and Blast programs, 2000, (17 stran).

[16] Univerzita Karlova v Praze, Přírodovědecká fakulta, *Chemické složení buněk, struktura proteinů.* Dostupný na URL http://www.natur.cuni.cz/~zdenap/lectures/ModUprProt07.pdf (duben 2008)

[17] Ian Korf, Mark Yandell, Joseph Bedell: BLAST, O'Reilly – ukázky z knihy. Dostupný na URL http://safari.java.net/0596002998/blast-CHP-7 (duben 2008).

[18] *National Center for Biotechnology Information* Dostupný na URL http://www.ncbi.nlm.nih.gov (duben 2008)

[19] Univerzity of Kentucky, *BLOSUM62 Substitution Matrix* Dostupný na URL http://www.uky.edu/Classes/BIO/520/BIO520WWW/blosum62.htm (duben 2008)

[20] *Sequence alignment.* Dostupný na URL http://www2.cs.uh.edu/~zhenzhao/Review/alignment.htm (duben 2008).

[21] Ian Korf, Mark Yandell, Joseph Bedell: BLAST, O'Reilly – ukázky ze statistik a skór. Dostupný na URL http://www.oreilly.com/catalog/blast/chapter/ch04.pdf (duben 2008).

[22] *Oracle Data Mining Application Developer's Guide 10g Release 1 (10.1).* Dostupný na URL http://www.acs.ilstu.edu/docs/oracle/datamine.101/b10699/6blast.htm (duben 2008).

Seznam příloh

Příloha č. 1

Funkce BLASTN_MATCH

```
function BLASTN_MATCH (
  query_seq CLOB,
  seqdb_cursor REF CURSOR,
  subsequence_from NUMBER default 1,
  subsequence_to NUMBER default -1,
  filter_low_complexity BOOLEAN default false,
  mask_lower_case BOOLEAN default false,
  expect_value NUMBER default 10,
  open_gap_cost NUMBER default 5,
  extend_gap_cost NUMBER default 2,
  mismatch_cost NUMBER default -3,
  match_reward NUMBER default 1,
  word_size NUMBER default 11,
  xdropoff NUMBER default 30,
  final_x_dropoff NUMBER default 50)
  return table of row (t_seq_id VARCHAR2, score NUMBER, expect NUMBER);
```

Parametry:

- query_seq – dotazovaná sekvence k porovnání
- seqdb_cursor – je charakterizován nejčastěji SQL dotazem pro vybrání části nukleotidové tabulky, SQL dotaz musí vracet pouze identifikátor sekvence a daný řetězec sekvence
- subsequence_from – označení počáteční pozice dotazované sekvence, která bude použita pro vyhledání
- subsequence_to – označení koncové pozice dotazované sekvence, která bude použita pro vyhledání
- filter_low_complexity – pokud je true, vyhledávací maska některého segmentu dotazu, které mají menší kompoziční složitost. Filtrování může eliminovat statisticky významnou ale biologicky již nevýznamnou oblast.
- mask_lower_complexity – pokud je true, můžeme specifikovat sekvenci s velkými písmeny jako dotazovanou a označit část pro odfiltrování části s malými písmeny
- expect_value – statistický významový práh pro hlášení nalezení podobnosti oproti databázi
- open_cost_gap – cena za otevření mezery
- extend_cost_gap – cena za rozšíření mezery
- mismatch_cast – penalta za nukleotidovou neshodu
- match_reward – odměna za nukleotidovou shodu
- word_size – velikost slova, používá se při rozdělní sekvence na části o dané délce word_size
- xdropoff – pokles pro BLAST rozšíření v bitech
- final_x_dropoff – konečný pokles o final_x_dropoff – parametr X

Výstupní parametry:

- t_seq_id – identifikátor nalezené sekvence
- score – skóre nalezené sekvence
- expect – očekávaná hodnota nalezené sekvence

Funkce BLASTP_MATCH

```
function BLASTP_MATCH (
  query_seq CLOB,
  seqdb_cursor REF CURSOR,
  subsequence_from NUMBER default 1,
  subsequence_to NUMBER default -1,
  filter_low_complexity BOOLEAN default false,
  mask_lower_case BOOLEAN default false,
  sub_matrix VARCHAR2 default 'BLOSUM62',
  expect_value NUMBER default 10,
  open_gap_cost NUMBER default 11,
  extend_gap_cost NUMBER default 1,
  word_size NUMBER default 3,
  x_dropoff NUMBER default 15,
  final_x_dropoff NUMBER default 25)
  return table of row (t_seq_id VARCHAR2, score NUMBER, expect NUMBER);
```

Parametry:

- query_seq – dotazovaná sekvence k porovnání
- seqdb_cursor – je charakterizován nejčastěji SQL dotazem pro vybrání části nukleotidové tabulky, SQL dotaz musí vracet pouze identifikátor sekvence a daný řetězec sekvence
- subsequence_from – označení počáteční pozice dotazované sekvence, která bude použita pro vyhledání
- subsequence_to – označení koncové pozice dotazované sekvence, která bude použita pro vyhledání
- filter_low_complexity – pokud je true, vyhledávací maska některého segmentu dotazu, které mají menší kompoziční složitost. Filtrování může eliminovat statisticky významnou ale biologicky již nevýznamnou oblast.
- mask_lower_complexity – pokud je true, můžeme specifikovat sekvenci s velkými písmeny jako dotazovanou a označit část pro odfiltrování části s malými písmeny
- sub_matrix – použitá substituční matice
- expect_value – statistický významový práh pro hlášení nalezení podobnosti oproti databázi
- open_cost_gap – cena za otevření mezery
- extend_cost_gap – cena za rozšíření mezery
- word_size – velikost slova, používá se při rozdělní sekvence na části o dané délce word_size
- xdropoff – pokles pro BLAST rozšíření v bitech
- final_x_dropoff – konečný pokles o final_x_dropoff – parametr X

Výstupní parametry:

- t_seq_id – identifikátor nalezené sekvence
- score – skóre nalezené sekvence
- expect – očekávaná hodnota nalezené sekvence

Příloha č. 3

Funkce TBLAST_MATCH

```
function TBLAST_MATCH (
  query_seq CLOB,
  seqdb_cursor REF CURSOR,
  subsequence_from NUMBER default 1,
  subsequence_to NUMBER default -1,
  translation_type VARCHAR2 default 'BLASTX',
  genetic_code NUMBER default 1,
  filter_low_complexity BOOLEAN default false,
  mask_lower_case BOOLEAN default false,
  sub_matrix VARCHAR2 default 'BLOSUM62',
  expect_value NUMBER default 10,
  open_gap_cost NUMBER default 11,
  extend_gap_cost NUMBER default 1,
  word_size NUMBER default 3,
  x_dropoff NUMBER default 15,
  final_x_dropoff NUMBER default 25)
  return table of row (t_seq_id VARCHAR2, score NUMBER, expect NUMBER);
```

Parametry:

- query_seq – dotazovaná sekvence k porovnání
- seqdb_cursor – je charakterizován nejčastěji SQL dotazem pro vybrání části nukleotidové tabulky, SQL dotaz musí vracet pouze identifikátor sekvence a daný řetězec sekvence
- subsequence_from – označení počáteční pozice dotazované sekvence, která bude použita pro vyhledání
- subsequence_to – označení koncové pozice dotazované sekvence, která bude použita pro vyhledání
- translation_type – druh zahrnuté translace – překladu, možnosti jsou BLASTX, BLASTP, TBLASTX. Standardní je BLASTX
- genetic_code – užito při translaci z nukleotidů na amino-kyseliny, genetic_code je druh mapovací tabulky, číslo – genetic_code udává jejich jméno
- filter_low_complexity – pokud je true, vyhledávací maska některého segmentu dotazu, které mají menší kompoziční složitost. Filtrování může eliminovat statisticky významnou ale biologicky již nevýznamnou oblast.
- mask_lower_complexity – pokud je true, můžeme specifikovat sekvenci s velkými písmeny jako dotazovanou a označit část pro odfiltrování části s malými písmeny
- sub_matrix – použitá substituční matice
- expect_value – statistický významový práh pro hlášení nalezení podobnosti oproti databázi
- open_cost_gap – cena za otevření mezery
- extend_cost_gap – cena za rozšíření mezery
- word_size – velikost slova, používá se při rozdělní sekvence na části o dané délce word_size

- xdropoff – pokles pro BLAST rozšíření v bitech
- final_x_dropoff – konečný pokles o final_x_dropoff – parametr X

Výstupní parametry:

- t_seq_id – identifikátor nalezené sekvence
- score – skóre nalezené sekvence
- expect – očekávaná hodnota nalezené sekvence

Příloha č. 4

Funkce BLASTN_ALIGN

```
function BLASTN_ALIGN (
  query_seq CLOB,
  seqdb_cursor REF CURSOR,
  subsequence_from NUMBER default 1,
  subsequence_to NUMBER default -1,
  filter_low_complexity BOOLEAN default false,
  mask_lower_case BOOLEAN default false,
  expect_value NUMBER default 10,
  open_gap_cost NUMBER default 5,
  extend_gap_cost NUMBER default 2,
  mismatch_cost NUMBER default -3,
  match_reward NUMBER default 1,
  word_size NUMBER default 11,
  xdropoff NUMBER default 30,
  final_x_dropoff NUMBER default 50)
  return table of row (
    t_seq_id VARCHAR2,
    pct_identity NUMBER,
    alignment_length NUMBER,
    mismatches NUMBER,
    positives NUMBER,
    gap_openings NUMBER,
    gap_list [Table of NUMBER],
    q_seq_start NUMBER,
    g_seq_end NUMBER,
    q_frame NUMBER,
    q_seq_end NUMBER,
    t_seq_start NUMBER,
    t_seq_end NUMBER,
    t_frame NUMBER,
    score NUMBER,
    expect NUMBER);
```

Parametry:

- query_seq – dotazovaná sekvence k porovnání
- seqdb_cursor – je charakterizován nejčastěji SQL dotazem pro vybrání části nukleotidové tabulky, SQL dotaz musí vracet pouze identifikátor sekvence a daný řetězec sekvence
- subsequence_from – označení počáteční pozice dotazované sekvence, která bude použita pro vyhledání
- subsequence_to – označení koncové pozice dotazované sekvence, která bude použita pro vyhledání
- filter_low_complexity – pokud je true, vyhledávací maska některého segmentu dotazu, které mají menší kompoziční složitost. Filtrování může eliminovat statisticky významnou ale biologicky již nevýznamnou oblast.
- mask_lower_complexity – pokud je true, můžeme specifikovat sekvenci s velkými písmeny jako dotazovanou a označit část pro odfiltrování části s malými písmeny

- expect_value – statistický významový práh pro hlášení nalezení podobnosti oproti databázi
- open_cost_gap – cena za otevření mezery
- extend_cost_gap – cena za rozšíření mezery
- mismatch_cast – penalta za nukleotidovou neshodu
- match_reward – odměna za nukleotidovou shodu
- word_size – velikost slova, používá se při rozdělní sekvence na části o dané délce word_size
- xdropoff – pokles pro BLAST rozšíření v bitech
- final_x_dropoff – konečný pokles o final_x_dropoff – parametr X

Výstupní parametry:

- t_seq_id – identifikátor nalezené sekvence
- pct_identity – procentuální identita porovnávaných sekvencí
- alignment_length – délka uspořádaní
- mismatches – počet párů s negativním skórem včetně 0
- positives – počet párů s pozitivním skórem
- gap_openings – počet otevřených mezer
- gap_list – tabulka, která obsahuje informace o mezerách
  - gap_start – číslo, které udává počátek vložené mezery v sekvenci, je typu NUMBER
  - gap_type – číslo, které udává, ve které sekvenci se mezera nachází, je typu NUMBER
    - 0 – nachází se v dotazované sekvenci
    - 1 – nachází se v sekvenci obsažené v databázi
  - gap_length – číslo, udává délku mezery, je typu NUMBER
- q_seq_start, q_seq_end – indikace části dotazované sekvence, která je nalezena jako podobná
- q_frame – translace čísla rámu na dotaz (hodnoty jsou v rozsahu -3 až +3, záporné číslo značí, že se sekvence porovnává od konce)
- t_seq_start, t_seq_end – indikace části cílové sekvence, která byla nalezena jako podobná
- t_frame – translace čísla rámu na cílovou sekvenci (hodnoty jsou v rozsahu -3 až +3, záporné číslo značí, že se sekvence porovnává od konce)
- score – skóre nalezené sekvence
- expect – očekávaná hodnota nalezené sekvence

Příloha č. 5

Funkce BLASTP_ALIGN

```
function BLASTP_ALIGN (
  query_seq CLOB,
  seqdb_cursor REF CURSOR,
  subsequence_from NUMBER default 1,
  subsequence_to NUMBER default -1,
  filter_low_complexity BOOLEAN default false,
  mask_lower_case BOOLEAN default false,
  sub_matrix VARCHAR2 default 'BLOSUM62',
  expect_value NUMBER default 10,
  open_gap_cost NUMBER default 11,
  extend_gap_cost NUMBER default 1,
  word_size NUMBER default 3,
  x_dropoff NUMBER default 15,
  final_x_dropoff NUMBER default 25)
 return table of row (
    t_seq_id VARCHAR2,
    pct_identity NUMBER,
    alignment_length NUMBER,
    mismatches NUMBER,
    positives NUMBER,
    gap_openings NUMBER,
    gap_list [Table of NUMBER],
    q_seq_start NUMBER,
    g_seq_end NUMBER,
    q_frame NUMBER,
    q_seq_end NUMBER,
    t_seq_start NUMBER,
    t_seq_end NUMBER,
    t_frame NUMBER,
    score NUMBER,
    expect NUMBER);
```

Parametry:

- query_seq – dotazovaná sekvence k porovnání
- seqdb_cursor – je charakterizován nejčastěji SQL dotazem pro vybrání části nukleotidové tabulky, SQL dotaz musí vracet pouze identifikátor sekvence a daný řetězec sekvence
- subsequence_from – označení počáteční pozice dotazované sekvence, která bude použita pro vyhledání
- subsequence_to – označení koncové pozice dotazované sekvence, která bude použita pro vyhledání
- filter_low_complexity – pokud je true, vyhledávací maska některého segmentu dotazu, které mají menší kompoziční složitost. Filtrování může eliminovat statisticky významnou ale biologicky již nevýznamnou oblast.
- mask_lower_complexity – pokud je true, můžeme specifikovat sekvenci s velkými písmeny jako dotazovanou a označit část pro odfiltrování části s malými písmeny
- sub_matrix – použitá substituční matice

- expect_value – statistický významový práh pro hlášení nalezení podobnosti oproti databázi
- open_cost_gap – cena za otevření mezery
- extend_cost_gap – cena za rozšíření mezery
- word_size – velikost slova, používá se při rozdělní sekvence na části o dané délce word_size
- xdropoff – pokles pro BLAST rozšíření v bitech
- final_x_dropoff – konečný pokles o final_x_dropoff – parametr X

Výstupní parametry:

- t_seq_id – identifikátor nalezené sekvence
- pct_identity – procentuální identita porovnávaných sekvencí
- alignment_length – délka uspořádaní
- mismatches – počet párů neshodných písmen
- positives – počet párů s pozitivním skórem
- gap_openings – počet otevřených mezer
- gap_list – tabulka, která obsahuje informace o mezerách
  - gap_start – číslo, které udává počátek vložené mezery v sekvenci, je typu NUMBER
  - gap_type – číslo, které udává, ve které sekvenci se mezera nachází, je typu NUMBER
    - 0 – nachází se v dotazované sekvenci
    - 1 – nachází se v sekvenci obsažené v databázi
  - gap_length – číslo, udává délku mezery, je typu NUMBER
- q_seq_start, q_seq_end – indikace části dotazované sekvence, která je nalezena jako podobná
- q_frame – translace čísla rámu na dotaz (hodnoty jsou v rozsahu -3 až +3, záporné číslo značí, že se sekvence porovnává od konce)
- t_seq_start, t_seq_end – indikace části cílové sekvence, která byla nalezena jako podobná
- t_frame – translace čísla rámu na cílovou sekvenci (hodnoty jsou v rozsahu -3 až +3, záporné číslo značí, že se sekvence porovnává od konce)
- score – skóre nalezené sekvence
- expect – očekávaná hodnota nalezené sekvence

Příloha č. 6

Funkce TBLAST_ALIGN

```
function TBLAST_ALIGN (
  query_seq CLOB,
  seqdb_cursor REF CURSOR,
  subsequence_from NUMBER default 1,
  subsequence_to NUMBER default 0,
  translation_type VARCHAR2 default 'BLASTX',
  genetic_code NUMBER default 1,
  filter_low_complexity BOOLEAN default false,
  mask_lower_case BOOLEAN default false,
  sub_matrix VARCHAR2 default 'BLOSUM62',
  expect_value NUMBER default 10,
  open_gap_cost NUMBER default 11,
  extend_gap_cost NUMBER default 1,
  word_size NUMBER default 3,
  x_dropoff NUMBER default 15,
  final_x_dropoff NUMBER default 25)
 return table of row (
    t_seq_id VARCHAR2,
    pct_identity NUMBER,
    alignment_length NUMBER,
    mismatches NUMBER,
    positives NUMBER,
    gap_openings NUMBER,
    gap_list [Table of NUMBER],
    q_seq_start NUMBER,
    q_frame NUMBER,
    q_seq_end NUMBER,
    t_seq_start NUMBER,
    t_seq_end NUMBER,
    t_frame NUMBER,
    score NUMBER,
    expect NUMBER);
```

Parametry:

- query_seq – dotazovaná sekvence k porovnání
- seqdb_cursor – je charakterizován nejčastěji SQL dotazem pro vybrání části nukleotidové tabulky, SQL dotaz musí vracet pouze identifikátor sekvence a daný řetězec sekvence
- subsequence_from – označení počáteční pozice dotazované sekvence, která bude použita pro vyhledání
- subsequence_to – označení koncové pozice dotazované sekvence, která bude použita pro vyhledání
- translation_type – druh zahrnuté translace – překladu, možnosti jsou BLASTX, TBLASTN, TBLASTX. Standardní je BLASTX
- genetic_code – užito při translaci z nukleotidů na amino-kyseliny, genetic_code je druh mapovací tabulky, číslo – genetic_code udává jejich jméno

- filter_low_complexity – pokud je true, vyhledávací maska některého segmentu dotazu, které mají menší kompoziční složitost. Filtrování může eliminovat statisticky významnou ale biologicky již nevýznamnou oblast.
- mask_lower_complexity – pokud je true, můžeme specifikovat sekvenci s velkými písmeny jako dotazovanou a označit část pro odfiltrování části s malými písmeny
- sub_matrix – použitá substituční matice
- expect_value – statistický významový práh pro hlášení nalezení podobnosti oproti databázi
- open_cost_gap – cena za otevření mezery
- extend_cost_gap – cena za rozšíření mezery
- word_size – velikost slova, používá se při rozdělní sekvence na části o dané délce word_size
- xdropoff – pokles pro BLAST rozšíření v bitech
- final_x_dropoff – konečný pokles o final_x_dropoff – parametr X

Výstupní parametry:

- t_seq_id – identifikátor nalezené sekvence
- pct_identity – procentuální identita porovnávaných sekvencí
- alignment_length – délka uspořádaní
- mismatches – počet párů neshodných písmen
- positives – počet párů s pozitivním skórem
- gap_openings – počet otevřených mezer
- gap_list – tabulka, která obsahuje informace o mezerách
  - gap_start – číslo, které udává počátek vložené mezery v sekvenci, je typu NUMBER
  - gap_type – číslo, které udává, ve které sekvenci se mezera nachází, je typu NUMBER
    - 0 – nachází se v dotazované sekvenci
    - 1 – nachází se v sekvenci obsažené v databázi
  - gap_length – číslo, udává délku mezery, je typu NUMBER
- q_seq_start, q_seq_end – indikace části dotazované sekvence, která je nalezena jako podobná
- q_frame – translace čísla rámu na dotaz (hodnoty jsou v rozsahu -3 až +3, záporné číslo značí, že se sekvence porovnává od konce)
- t_seq_start, t_seq_end – indikace části cílové sekvence, která byla nalezena jako podobná
- t_frame – translace čísla rámu na cílovou sekvenci (hodnoty jsou v rozsahu -3 až +3, záporné číslo značí, že se sekvence porovnává od konce)
- score – skóre nalezené sekvence
- expect – očekávaná hodnota nalezené sekvence

Postup pro načtení dat do databáze

- stáhnete si data *.dat
- použijete Perl skript(*viz. Příloha č. 8*) a vyexportujete data do *.txt, kde již budou formátována

```
swissprot.pl xxx.dat > yyy.txt
```

- dále vytvoříte tabulku, kam budou tyto data nahrány

```
create table swissprot(
    SEQ_ID VARCHAR2(32),
    CREATION_DATE VARCHAR2(32),
    ORGANISM VARCHAR2(256),
    SEQ_DATA CLOB
);
```

- vytvoříte soubor sprot.ctl, který bude obsahovat následující:

```
LOAD DATA
INFILE yyy.txt
INTO TABLE swissprot
REPLACE
FIELDS TERMINATED BY '|'
TRAILING NULLCOLS
(
seq_id,
creation_date,
organism,
seq_data char(100000)
)
```

- A nakonec nahrajete data pomocí SQL * Loader z yyy.txt do tabulky swissprot

```
sqlldr userid=<user_name>/<passwd> control=sprot.ctl
log=sprot.log
direct=TRUE data=yyy.txt
```

Příloha č. 8

Perl skript

```
#!/bin/perl
#swissprot.pl < input > output
#Input: protein db as provided by SWISSPROT
#
my $string = "";
my $indicator = "";
$sq = 0;
$ac = 0;

while(<>)
{
    #chop;
    if ( /^\/\// ) {
      print "\n";
      $sq = 0;
      $ac = 0;
      next;
    }
    if ($sq == 1) {
        @words = split;
        foreach $word (@words) {
          print "$word";
        }
        next;
    }
    if( /^AC(\s+)(\w+);/ ) {
      if ($ac == 0) {
        $indicator = $2;
        print "$indicator|";
        $sq = 0;
        $dt = 0;
        $ac = 1;
        next;
      }
    }
    if ( /^OS(\s+)(.*)\./ ) {
        $organism = $2;
        print "$organism|";
        next;
    }
    if ( /^DT(\s+)(\S+)/ ) {
        if ($dt == 0) {
           print "$2|";
           $dt = 1;
        }
    }
    if ( /^SQ(\s+)/ ) {
        $sq = "1";
        next;
    }
}
```