VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

EFEKTIVNÍ HLEDÁNÍ PŘEKRYVŮ U NGS DAT

EFFECTIVE SEARCH FOR OVERLAPS IN NGS DATA

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. PETR MATOCHA**
AUTHOR

VEDOUCÍ PRÁCE **Ing. JANKA PUTEROVÁ**
SUPERVISOR

BRNO 2017

Zadání diplomové práce/19337/2016/xmatoc02

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2016/2017

Zadání diplomové práce

Řešitel: **Matocha Petr, Bc.**
Obor: Bioinformatika a biocomputing
Téma: **Efektivní hledání překryvů u NGS dat**
Effective Search for Overlaps in NGS Data
Kategorie: Bioinformatika

Pokyny:

1. Nastudujte základy molekulární biologie a dále se zaměřte na repetitívní elementy v DNA a sekvenovací technologie.
2. Nastudujte literaturu z dané oblasti a vytvořte přehled dostupných nástrojů a metod pro hledání překryvů DNA sekvencí.
3. Navrhněte vhodný algoritmus nebo modifikujte existující algoritmus pro hledání překryvů DNA sekvencí generovaných NGS technologiemi.
4. Po konzultaci s vedoucí implementujte navržený algoritmus a otestujte na vhodně zvoleném vzorku dat.
5. Diskutujte dosažené výsledky a další možné pokračování projektu.

Literatura:

- Davalbhakta S. 2014. Bliss: A New Read Overlap Detection Algorithm. IJBB. 1-7.
- Buhler J. 2001. Efficient large-scale sequence comparison by locality-sensitive hashing. Bioinformatics. 17:419-428. doi: 10.1093/bioinformatics/17.5.419.
- Wicker T et al. 2007. A unified classification system for eukaryotic transposable elements. Nat. Rev. Genet. 8:973-982. doi: 10.1038/nrg2165-c4.
- Zhang Z, Schwartz S, Wagner L, Miller W. 2000. A Greedy Algorithm for Aligning DNA Sequences. J. Comput. Biol. 7:203-214. doi: 10.1089/10665270050081478.
- Dále dle doporučení vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 až 3.

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

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% 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: **Puterová Janka, Ing.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 24. května 2017

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

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Hlavním tématem této práce je detekce překryvů u NGS dat. Práce obsahuje přehled sekvenovacích technologií, jenž jsou zdrojem NGS dat. V práci je obecně definován problém detekce překryvů DNA sekvencí. Následně je v práci vytvořen přehled dostupných existujících algoritmů a přístupů pro detekci překryvů u NGS dat. Jsou zde popsány základní principy těchto algoritmů. V druhé části práce je navržen vhodný nástroj pro detekci přibližných překryvů u NGS dat a popsána jeho implementace. V závěru práce jsou shrnuty a popsány experimenty provedené s tímto nástrojem a závěry, které z nich vyplývají.

Abstract

The main theme of this work is the detection of overlaps in NGS data. The work contains an overview of NGS sequencing technologies that are the source of NGS data. In the thesis, the problem of overlapping detection is generally defined. Next, an overview of the available algorithms and approaches for detecting overlaps in NGS data is created. Principles of these algorithms are described herein. In the second part of this work a suitable tool for detecting approximate overlaps in NGS data is designed and its implementation is described herein. In conclusion, the experiments performed with this tool and the conclusions that follow are summarized and described.

Klíčová slova

překryvy, detekce překryvů, sekvenování DNA, NGS data, NGS technologie, DNA, genom, read, repetitivní elementy, algoritmy pro detekci překryvů, sufixové filtry, FM-index

Keywords

overlaps, overlaps detection, DNA sequencing, NGS data, NGS technologies, DNA, genome, read, repetitive elements, algorithms for overlap detection, suffix filters, FM-index

Citace

MATOCHA, Petr. *EFEKTIVNÍ HLEDÁNÍ PŘEKRYVŮ U NGS DAT*. Brno, 2017. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Puterová Janka.

EFEKTIVNÍ HLEDÁNÍ PŘEKRYVŮ U NGS DAT

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením paní Ing. Janky Puterové. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Petr Matocha
23. května 2017

Poděkování

Velké poděkování patří mé vedoucí, Ing. Jance Puterové, za poskytnuté rady a připomínky při psaní této práce.

Obsah

Kapitola 1

Úvod

Získávání a následná práce s DNA je jedna ze základních úloh bioinformatiky. V současné době ovšem není možné získat přímo celý genom najednou. DNA je čtena nejrůznějšími technologiemi po částech. Tyto fragmenty jsou v bioinformatice označovány jako ready. Čtení DNA respektive dekódování jednotlivých symbolů DNA se označuje termínem DNA sekvenování. V současnosti nejpoužívanější technologie pro sekvenování DNA jsou NGS metody (z angl. Next-Generation Sequence Methods). Tyto metody poskytují krátké ready s vysokou propustností za přijatelnou cenu. Jsou schopny produkovat v jednom běhu obrovské množství dat (v řádech gigabajtů), jenž je zapotřebí následně zpracovat.

Po přečtení jednotlivých readů je nutné k sestavení celého genomu spojit tyto ready do sebe. K tomuto účelu vznikla celá řada nástrojů. Tyto nástroje musí být schopny se vypořádat při zpracování velkého množství NGS dat s následujícími problémy: omezenými výpočetními zdroji, chybami v datech, které vznikají při sekvenování, a také komplikacemi, jenž způsobují repetitivní elementy. Proces sestavování NGS dat se obvykle skládá ze 4 základních fází: předzpracování dat, sestrojení grafu, zjednodušení grafu, závěrečné filtrování. Jednotlivé nástroje pro sestavování NGS dat využívají různé přístupy pro sestrojení grafu. Jedním z těchto přístupů je právě hledání překryvů DNA sekvencí. Jedná se o klasický přístup, který je vhodný především pro delší ready. Současné NGS metody poskytují kratší ready, ovšem s technologickým vývojem se tato délka readů zvětšuje a dá se tedy očekávat vyšší využití tohoto přístupu.

Naivním způsobem hledání překryvů DNA sekvencí je porovnávání každé sekvence se všemi ostatními. Je ovšem zřejmé, že se jedná o $O(N^2)$ složitý problém, kde N je počet jednotlivých DNA sekvencí. Tento způsob využívá nástroj RepeatExplorer, který slouží k identifikaci a charakterizaci repetitivních elementů v DNA. Ovšem přístup porovnávání každého s každým je v důsledku své velké časové složitosti zcela nevhodný pro zpracování velkého množství dat poskytovaných NGS metodami. Tento nástroj tedy vyžaduje vylepšení způsobu detekce překryvů. Cílem této práce je proto vytvořit přehled dostupných nástrojů pro zpracování NGS dat založených na principu hledání překryvů DNA sekvencí a efektivnějších přístupů, které tyto nástroje k tomuto účelu využívají, popřípadě i dalších algoritmů a přístupů, které je možné k detekci překryvů DNA sekvencí využít. Následně zhodnotit současný stav, vhodnost a dostupnost těchto algoritmů a přístupů. Navrhnout novou metodu nebo modifikaci stávající, implementovat ji a otestovat na vhodné sadě testovacích dat.

V úvodní části této práce, kapitola 2, se nachází základní teoretické informace z oblasti bioinformatiky, především o DNA, genomu a repetitivních elementech v DNA. Kapitola 3 je zaměřena na DNA sekvenování. Obsahuje přehled a popis jednotlivých sekvenovacích

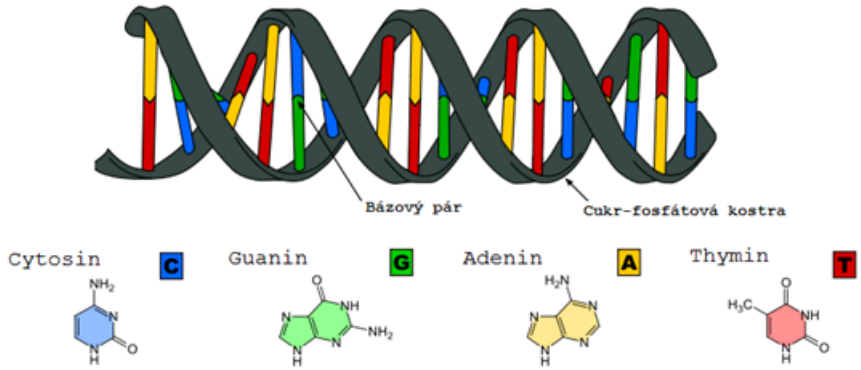
technologií, jejich výhod a nevýhod. Kapitola 4 se pak věnuje přehledu dostupných algoritmů a přístupů pro hledání překryvů DNA sekvencí. V této kapitole jsou uvedeny nástroje založeny na hledání překryvů DNA sekvencí a především algoritmy, které tyto nástroje využívají, a jejich časová a paměťová náročnost. Jsou zde také uvedeny další možné algoritmy vhodné pro detekci překryvů DNA sekvencí. Také je zde obecně definován tento problém detekce překryvů DNA sekvencí. V závěru této kapitoly je pak shrnut současný stav dostupných algoritmů a přístupů. V kapitole 5 je navržen vhodný algoritmus pro detekci DNA překryvů a jeho vylepšení. V následující kapitole 6 je popsána implementace tohoto algoritmu. Předposlední kapitola 7 se věnuje testování a experimentům s tímto nástrojem pro detekci překryvů DNA sekvencí. Jsou zde také zhodnoceny výsledky a závěry plynoucí z těchto experimentů. V samotném závěru, kapitola 8, je shrnuta a zhodnocena celá práce. Jsou zde především zhodnoceny výsledky a závěry, jenž z ní vyplývají a nastíněn směr, kterým se může práce dále ubírat.

Kapitola 2

Základy molekulární biologie

2.1 DNA

Deoxyribonukleová kyselina nebo-li DNA je základním genetickým materiálem skoro všech žijících organismů. Molekula DNA je primárně tvořená pořadím sekvence nukleotidů. Nukleotidy jsou složeny ze sacharidu 2-deoxyribózy, fosfátové skupiny a nukleové báze. Existují celkem čtyři nukleové báze: adenin (A), cytosin (C), guanin (G) a thymin (T). Adenin a guanin jsou označovány jako purinové báze a cytosin a thymin jako pyrimidinové báze [3]. U většiny organismů se DNA vyskytuje v podobě pravotočivé dvoušroubovice tvořené dvěma antiparalelními komplementárními řetězci. Konec řetězce, který obsahuje hydroxylovou skupinu, se nazývá 3' konec. Konec s fosfátem je označován jako 5' konec. Řetězce jsou navzájem spojeny pomocí vodíkových můstků mezi jednotlivými nukleovými bázemi. Podle Watson-Crickova pravidla se páruje A s T (dvě vodíkové vazby) a C s G (tři vodíkové vazby). Struktura DNA je znázorněna na obrázku 2.1. Díky komplementaritě postačí k reprezentaci DNA pouze jeden řetězec. Z pohledu informatiky je tedy možné definovat DNA jako řetězec nad abecedou {A, C, G, T}. Velikost DNA je definována pomocí počtu párů bází. Označuje se bp (base pair).



Obrázek 2.1: Struktura DNA [1].

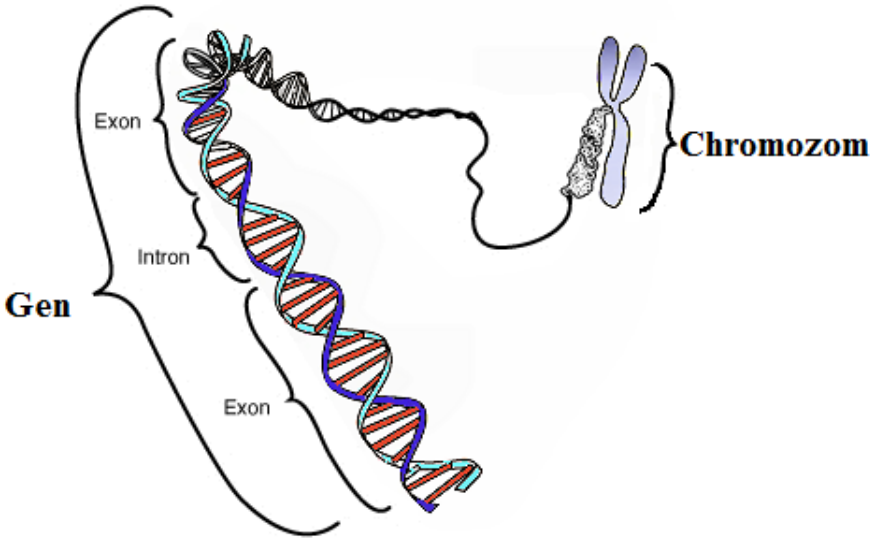
DNA byla poprvé demonstrována jakožto genetický materiál Oswaldem Theodorem Averym v roce 1944. V roce 1953 pak James D. Watson a Francis Crick popsali DNA jako dva šroubovité řetězce stočené kolem osy složené ze čtyř druhů bází.

2.2 Struktura genomu

Genomem se rozumí veškerá genetická informace uložená v DNA konkrétního organismu. Zahrnuje tedy veškeré kódující i nekódující sekvence DNA daného organismu. Kódující sekvence genomu se nazývají geny. Genom také může obsahovat velké množství repetitivních elementů (viz podkapitola 2.3). Genom se skládá ze sady chromozomů. Každý chromozom obsahuje jednu molekulu DNA. Chromozom umožňuje sdružení velkého množství DNA v organizované a kompaktní formě. Skládá se z proteinů zvaných histony, kolem nichž je molekula DNA omotána [3].

Geny

Geny jsou nejdůležitější části genomu, jelikož nesou informaci, jež je základem pro život. Dělí se na geny kódující protein, označovány jako strukturní geny, a geny nekódující protein, funkční geny, které projevují svoji funkci ve formě RNA. Lidský genom obsahuje 25-30 tisíc genů. Gen je u eukaryot tvořen exony a introny, u prokaryot pouze exony. Exony jsou kratší úseky, které určují instrukce pro tvorbu proteinů. Introny jsou naopak delší úseky, na jejichž funkci se názory liší. Avšak výskyt intronů umožňuje alternativní sestřih při přepisu na RNA a tím komplexnější tvorbu různých proteinů [3].



Obrázek 2.2: Gen - kódující oblast v segmentu eukaryotické DNA [2].

2.3 Repetitivní elementy v DNA

Repetitivní elementy v DNA jsou opakující se sekvence v DNA. Jedná se o vzory, které jsou rozšířeny napříč celým genomem v několika kopiích. Jsou velmi rozšířeny v genomech jak u eukaryotických organismů, tak i prokaryotických. Repetitivní elementy mohou být rozděleny do dvou skupin: satelitní elementy a transpozibilní elementy (transpozony). Satelitní elementy jsou tandemové repetitivní segmenty, které se v rámci genomu nepohybují.

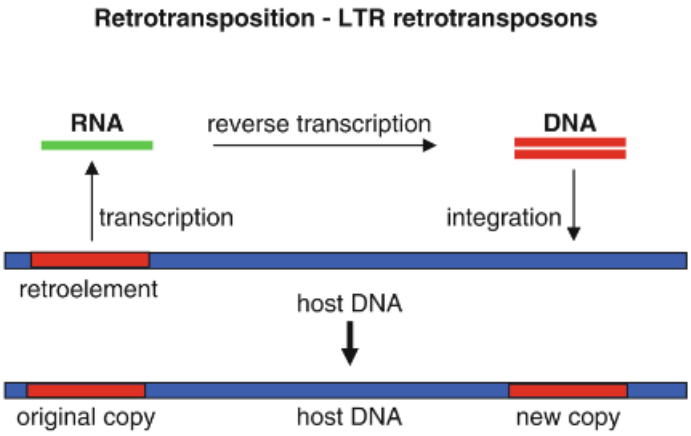
Transpozony jsou elementy, které jsou schopny se v rámci genomu pohybovat z jednoho místa na druhé. Při tomto procesu často tvoří své kopie. Touto schopností přesouvat se a replikovat se značně ovlivňují genom. V důsledku tohoto procesu transpozice jsou schopny tvořit různé genetické změny (vložení, duplikace, vyříznutí, nebo translokace v místě začlenění) v průběhu začleňování do genomu [10].

Transpozony se na základě mechanismu transpozice dělí do dvou tříd. Do třídy I patří transpozony, které se přemísťují pomocí RNA. Označují se jako retrotranspozony nebo retroelementy. Transpozony, které se přemísťují pomocí DNA, spadají do třídy II a jsou nazývány DNA transpozony. Každá skupina transpozonů obsahuje autonomní a neautonomní prvky. Autonomní elementy obsahují ORF (z angl. Open Reading Frame), jenž kóduje produkt potřebný pro transpozici. Neautonomní prvky jsou schopny se přemisťovat z důvodu zachování cis sekvence, která je nezbytná pro transpozici [22].

Třída I - Retrotranspozony

Transpozony třídy I se v rámci genomu pohybují pomocí RNA. Při každém cyklu replikace vytváří jednu novou kopii. Tento mechanismus se označuje jako "copy-and-paste"a je znázorněn na obrázku 2.3. Na základě dlouhých ukončovacích repeticích LTRs (z angl. Long Terminal Repeats), které obklopují tělo retroelementu, se dělí na dvě skupiny:

- **LTR retrotranspozony** - mají dlouhé ukončovací repetice.
- **Non-LTR retrotranspozony** - mají dlouhé ukončovací repetice.

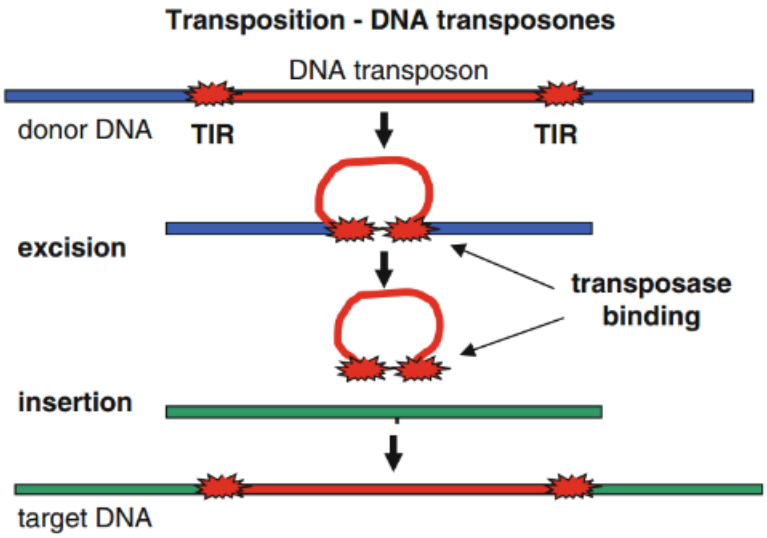


Obrázek 2.3: Retro transpozice LTR retrotranspozonů mechanismem "copy-and-paste"[9].

Třída II - DNA transpozony

Transpozony třídy II využívají k transpozici DNA. Mechanismus transpozice se u DNA transpozonů označuje "cut-and-paste", jelikož dochází k vyříznutí transpozonu z původní pozice a začlenění do pozice nové. Tento mechanismus je znázorněn na obrázku 2.4. DNA transpozony se skládají z genu transpozázy, jenž obklopují dvě obracené repetice nazývané TIRs (z angl. Terminal Inverted Repeats). Transpozáza rozpozná repetice TIRs a vyřeže

tělo DNA transpozonu, které je poté vloženo na nové místo v genomu. Následně je duplikované cílové místo DNA a výsledkem jsou TSDs (z angl. Target Site Duplications), které predstavují jedinečný rys pro každý DNA transpozon.



Obrázek 2.4: Transpozice DNA transpozonů mechanismem "cut-and-paste"[9].

DNA transpozony se dělí do dvou podtříd v závislosti na sekvencích TIRs a TSDs. Podtřída I obsahuje rodiny: Tc1/mariner, PIF/Harbinger, hAT, Mutator, Merlin, Transib, P, piggyBac and CACTA. Podtřída II se skládá z rodin: Helitron a Maverick, které využívají jiný mechanismus transpozice [16].

Kapitola 3

DNA sekvenování

Celá tato kapitola vychází a byla volně převzata z [5] a z [13].

DNA sekvenování je proces analýzy biologického materiálu pomocí biochemických metod. Tyto metody určují pořadí nukleových bází v sekvencích DNA. Tedy DNA sekvenování je proces stanovení primární struktury DNA. První metody sekvenování vznikly v sedmdesátých letech dvacátého století, kdy Sanger a jeho spolupracovníci a Maxam a Gilbert vynalezli metody sekvenování DNA. Tento objev měl velký vliv na současnou biologii a medicínu, jelikož tyto metody umožnily dešifrování celých genů a později i celých genomů. Tyto dvě metody jsou nyní označovány za metody první generace též jako klasické metody. V následujících čtyřiceti letech prošlo sekvenování DNA význačným pokrokem. Postupem času začaly vznikat takzvané NGS metody (z angl. Next-Generation Sequencing Methods), označované jako metody druhé generace. Tyto metody jsou v současnosti nejvíce využívány, a proto se tato kapitola bude zaměřovat především na ně. V nedávné době pak byly představeny PGM (z angl. Personal Genome Machines) a metody třetí generace (angl. Third-generation sequencing). Základními požadavky, které jsou kladeny na sekvenovací metody, jsou rychlost, přesnost, propustnost, snadná použitelnost a cena. Každá metoda má své výhody a nevýhody, jež budou popsány v následujících podkapitolách.

3.1 Klasické metody sekvenování

Mezi klasické metody sekvenování patří dvě nejstarší metody sekvenování. První z nich je známá pod názvem Sangerovo sekvenování. Tuto metodu představil v roce 1977 Fredirick Sanger a v roce 1980 za ni získal Nobelovu cenu. Tato metoda je založena na detekci ukončení prodlužujícího se vlákna DNA.

Sangerova metoda byla po dlouhou dobu nejpoužívanější sekvenovací metodou a v roce 2004 umožnila první dokončení lidského genomu v rámci projektu The Human Genom Project. Délka čtených fragmentů (readů) dosahuje velikosti 400 až 900 bází. Metoda tedy poskytuje dlouhé ready vysoké kvality. Mezi její nevýhody patří finanční náročnost a nízká propustnost.

Druhou klasickou metodou je Maxam-Gilbertova metoda, jež byla vymyšlena souběžně, ovšem nezávisle na Sangerově metodě ve stejném roce (1977). Maxam-Gilbertova metoda byla stejně jako Sangerova metoda oceněna Nobelovou cenou. Tato metoda je založena na chemické modifikaci DNA a následném štěpení specifických bází. Tato metoda ovšem není běžně používaná z důvodu práce s chemicky toxickými látkami a radioizotopy. Její laboratorní postup je náročnější nežli u Sangerovy metody.

3.2 NGS metody

The Human Genome Project, kde byla využívána Sangerova metoda, poukázal na omezení Sangerovi metody, jelikož vyžadoval velké množství času a zdrojů. Bylo zřejmé, že je zapotřebí rychlejších a levnějších metod s vyšší propustností. To vedlo na začátku 21. století k vývoji a komercializaci NGS metod.

Tyto nové sekvenovací metody sdílejí tři základní zlepšení oproti klasickým metodám. První z nich je, že nevyžadují bakteriální klonování DNA fragmentů, na místo toho se spoléhají na přípravu NGS knihoven v bezbuněčném systému. Dalším zlepšením je paralelní zpracování velkého množství vzorků. Tyto metody umožňují na místo stovek sekvenčně prováděných sekvenovacích reakcí provádět tisíce až miliony reakcí paralelně. Třetím zlepšením je schopnost přímo detekovat výstup sekvenování bez potřeby elektroforézy. NGS metody jsou tedy schopny generovat velké množství readů, což umožňuje sekvenování celého genomu vysokou rychlostí. Zásadním nedostatkem NGS metod je, že produkují relativně krátké sekvence, tzv. ready. Tento fakt komplikuje sestavování genomu, a proto tento proces vyžaduje speciální nástroje a algoritmy.

Nejpoužívanějšími NGS metodami jsou 454 Life Roche (dříve Science), SOLiD Systém a Illumina Genom Analyzer. Mezi NGS metody může být také zařazena technologie PGM (z angl. Personal Genome Machine) s názvem Ion Torrent. Přehledné srovnání nejpoužívanějších NGS metod a metody třetí generace PacBio, též SMRT, je znázorněno na 3.1.

454 Roche

454 Roche byla první úspěšnou komerčně využívanou NGS metodou. Tato technologie byla vytvořena v srpnu roku 2005. Metoda je založena na principu pyrosekvenování.

Délka readů technologie Roche 454 byla původně v roce 2005 100 až 150 bp. V roce 2008 vznikl systém 454 GS FLX Titanium. Délka readů u tohoto systému dosahovala až 700 bp s přesností 99.9% (po filtrování). Jeden běh poskytoval výstup o velikosti 0,7 Gb a proběhl během 24 hodin. V roce 2009 společnost Roche představila systém, který zjednodušil přípravu knihovny a zpracování dat a také zlepšil výstup samotný na 14 Gb na jeden běh.

Hlavní výhodou technologie Roche je ovšem jeho vysoká rychlost. Čas potřebný k dokončení sekvenování od jeho započetí se pohybuje kolem 10 hodin. Za výhodu této technologie je též považována délka readů, která se, jak už bylo zmíněno, pohybuje kolem 700 bp. Delší ready se snadněji mapují na referenční genom a také jsou výhodou pro *de novo* sestavování genomu. Nevýhodou této technologie ovšem je velká cena činidel, která vychází na 12 dolarů na milión bází. Tato metoda také poskytuje relativně nízkou propustnost. V důsledku konkurence jiných technologií, které jsou značně levnější, není tato metoda v současnosti příliš využívána a to i navzdory zvýšení délky readů až na 1000 bp.

SOLiD

SOLiD (z angl. Sequencing by Oligonucleotide Ligation Detection) byl vytvořen společností Applied Biosystems v roce 2007. Sekvenátor využívá technologii dvou bázového sekvenování založenou na ligázovém sekvenování.

Původní délka readů metody SOLiD byla 35 bp. Výstup na jeden běh pak 3 Gb. Díky dvou bázové metodě sekvenování byla dosažena vysoká přesnost 99,85% (po filtrování). První systém byl vydaný v roce 2007, později v roce 2010 byl vytvořen systém SOLiD 5500xl. Tento nový systém zlepšil délku readů na 85 bp, přesnost na 99,99% a velikost

datového výstupu na 30 Gb dat na jeden běh. Jeden běh pak může být dokončen během 7 dnů. Cena sekvenování se pohybuje kolem 0,04 dolaru na milión bází.

SOLiD metoda poskytuje druhou nejvyšší propustnost na trhu. Metoda také poskytuje vysokou přesnost způsobenou faktem, že každá báze je čtena dvakrát. Největší nevýhodou této metody je velmi krátká délka readů a relativně dlouhá doba běhu. Také se nehodí k sestavovaní genomu *de novo*.

Ilumina Genom Analyzer

V roce 2006 společnost Solexa vytvořila Genom Analyzer, v roce 2007 byla tato společnost koupena společností Illumina, proto název Illumina Genom Analyzer. Tato metoda využívá technologie sekvenování pomocí syntézy.

Výstup prvního Solexa GA byl 1 Gb/běh. Díky vylepšením se tato hodnota postupně zvyšovala. Poslední verze Illumina Genome Analyzeru GAIIx poskytuje výstup okolo 85 Gb na jeden běh. V roce 2010 Illumina vytvořila nástroj HiSeq 2000 založený na stejné sekvenovací technologii. Výstup tohoto nástroje byl 200 Gb na jeden běh. V dnešní době pak poskytuje výstup o velikosti 600 Gb na jeden běh a sekvenování může být dokončeno během 8 dní. V blízké budoucnosti se očekává, že by tyto hodnoty mohly dosáhnout 1 Tb na jeden běh. V porovnání s 454 a SOLiD je HiSeq 2000 nejlevnější metodou sekvenování, tato cena se se pohybuje kolem 0,02 dolaru na milion bází. Délka readů se pohybuje okolo 300 bp.

V současnosti je Illumina lídrem v NGS průmyslu, proto většina protokolů pro přípravu knihoven je kompatibilní s Illumina systémem. Illumina poskytuje nejvyšší propustnost ze všech metod a nejnižší cenu na jednu bázi. Délka readů je kompatibilní téměř se všemi aplikacemi. Nevýhodou je technická náročnost přípravy vzorku, jenž vyžaduje velkou kontrolu. Další nevýhodou je požadavek na složitost sekvenované sekvence.

PGM – Ion Torrent

V roce 2010 společnost Ion Torrent (nyní Life Technologies) představila první PGM systém (z angl. Personal Genome Machine), čili nástroj s vysokou rychlostí, nízkou cenou a malými rozměry. Tato technologie pracuje na stejném principu jako metoda 454 Roche. Jediným rozdílem je, že na místo pyrofosfátu je uvolněn proton a ten je detekován pomocí ionového senzoru. Jedná se tedy o první zařízení, které ke své činnosti nevyžaduje optické obrazové zařízení a fluorescenční nukleotidy. Je schopno dosáhnout kvalitních výsledků ve vysoké rychlosti, za nízkou cenu a s využitím zařízení malých rozměrů. V současnosti poskytuje ready o velikosti 400 bp. Doba běhu se pohybuje pouze okolo 2 hodin. Nevýhody této technologie jsou ovšem podobné jako u 454 Roche. Tedy například vysoká chybovost a nízká propustnost.

3.3 Metody třetí generace

Metody třetí generace přinášejí dvě základní vylepšení oproti NGS metodám. Nevyžadují pro přípravu sekvenovacího materiálu PCR a tedy doba přípravy se výrazně zkracuje. Druhou výhodou je, že umožňují zachycení signálu v reálném čase. V současnosti existují dvě metody. Metoda SMRT (3.3) a metoda Nanopore (3.3).

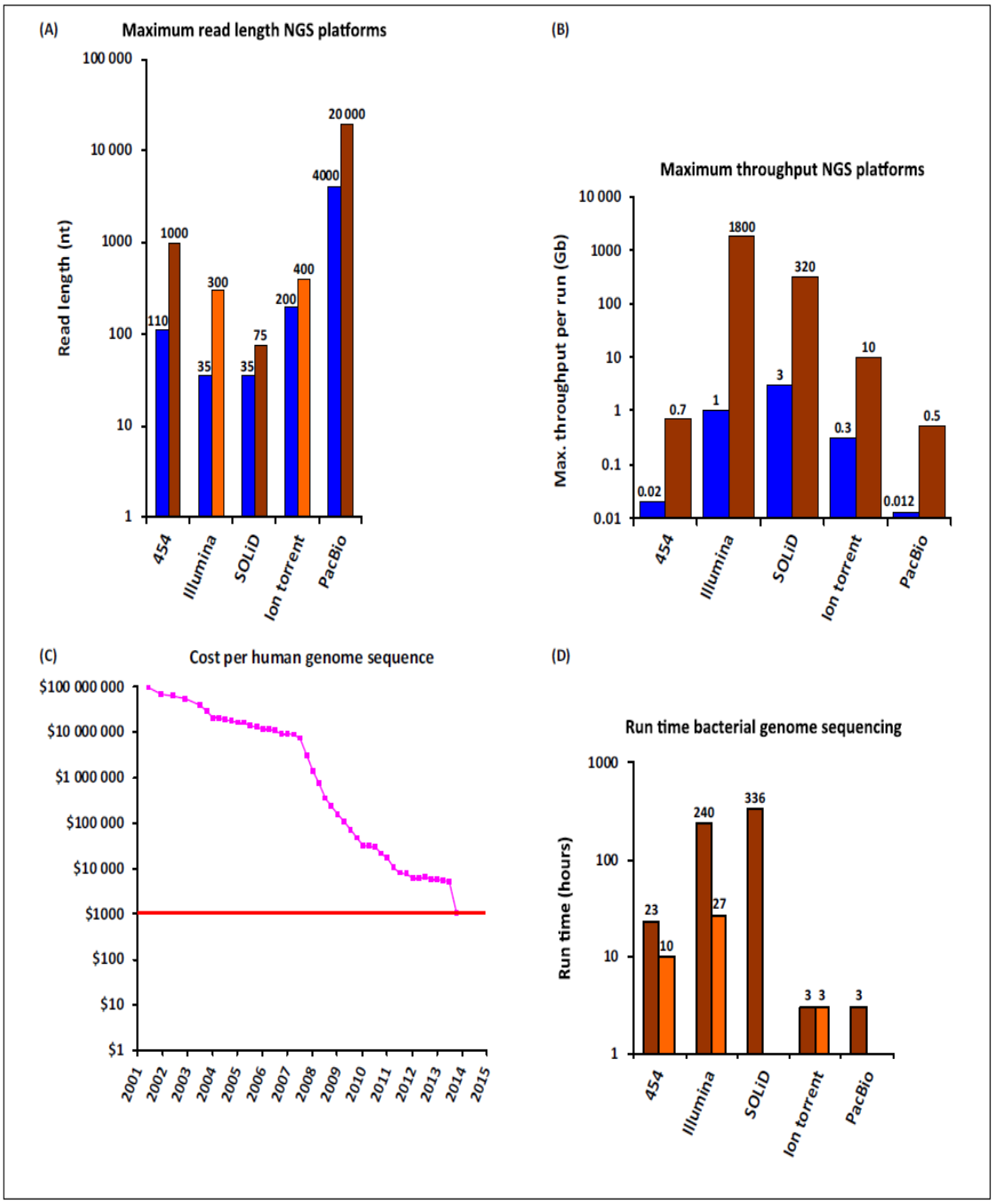
Single molecule real-time (PacBio)

Metoda SMRT (z angl. Single Molecule Real-time) byla vytvořena firmou Pacific Biosciences. Je založena na real-time zobrazování fluorescenčně označených nukleotidů v průběhu jejich syntézy. Sekvenování je realizováno na čipu, který je pokryt tisíci malými dírkami o poloměru několika nanometrů. Průměrná délka readů se pohybuje okolo 3 kb až 20 kb, což je značně větší délka oproti NGS metodám. Další výhodou je rychlost. Metoda SMRT je schopna produkovat 1 až 3 nukleotidy za sekundu. Nevýhodou ovšem je nízká propustnost. Přesnost se pohybuje pouze okolo 80%.

Nanopore (Oxford Nanopore Technologies)

Metoda je založena na průchodu molekuly DNA velmi úzkým místem - nanopórem. Při průchodu dochází k detekci jednotlivých nukleotidů. Délka readů se pohybuje v řádech desítek kilobází. Tuto metodu využívají tři zařízení od od společnosti Oxford Nanopore Technologies - MinION, PromethION a GridION. MinION je velmi malé zařízení, které lze k počítači připojit pomocí USB. Je určeno na vytváření stovek paralelních experimentů s celkovou dobou běhu několika hodin až dní. PromethION je stolní zařízení, které slouží pro experimenty s větší propustností, anebo pro větší počet vzorků. GridION systém pak byl navržen tak, aby umožňoval škálovatelnost.

První výhodou této technologie je délka bází, která přesahuje 5 kb. Druhou výhodou je, že nevyžaduje fluorescenční nukleotidy. V důsledku sekvenování jednotlivých vláken DNA není zapotřebí složitá příprava sekvenovacího vzorku a tedy doba sekvenování je výrazně kratší. Nevýhodou je ovšem vysoká chybovost.



Obrázek 3.1: Evoluce sekvenovacích metod s vysokou propustností. (A) Srovnání maximální délky readů komerčně dostupných sekvenovacích nástrojů. Modré sloupce znázorňují první dostupné verze jednotlivých nástrojů. Oranžové sloupce označují verze dostupné dnes. (B) Srovnání maximální propustnosti komerčně dostupných sekvenovacích nástrojů. Modré sloupce znázorňují první dostupné verze jednotlivých nástrojů. Oranžové sloupce označují verze dostupné dnes.(C) Graf zobrazuje vývoj ceny sekvenování lidského genomu od roku 2001 do roku 2014.(D) Čas k dokončení obvyklého běhu potřebného k nasekvenování bakteriálního genomu. Tmavě oranžová znázorňuje obvyklé nástroje. Světle oranžová znázorňuje nejlepší dostupné verze nástrojů [5].

Kapitola 4

Algoritmy pro hledání překryvů DNA sekvencí

Cílem této kapitoly je shrnout existující algoritmy a přístupy pro hledaní překryvů DNA sekvencí. Především se zaměřuje na přibližné překryvy, které se od sebe liší určitým počtem chyb. Kapitola se také zaměřuje na algoritmy, které využívají existující nástroje pro sestavování genomu, jež jsou založené na OLC (z angl. Overlap-layout Consensus) principu. Nejprve je v této kapitole popsán problém detekce překryvů. Poté jsou popsány jednotlivé algoritmy a přístupy pro detekci. V závěru se nachází shrnutí a přehled těchto algoritmů.

Nejjednodušším způsobem detekce překryvů je porovnání každé sekvence s každou. Tato metoda je velmi časově neefektivní, jelikož se jedná o $O(N^2)$ složitý problém, kde N je počet sekvencí DNA. Tento přístup využívá nástroj RepeatExplorer. Z tohoto důvodu tento nástroj vyžaduje vylepšení detekce překryvů. RepeatExplorer také využívá velmi nízké pokrytí genomu, obvykle maximálně 5%, z kterých lze určit repetice, jelikož se v genomu vyskytují ve stovkách až tisících kopií.

4.1 Problém detekce překryvů DNA sekvencí

Nechť X a Y označují ready a $\overline{X}$ a $\overline{Y}$ označují k nim reverzní komplementární ready. Dále nechť R označuje množinu všech readů získaných sekvenováním určitého genomu. Problémem sestavení sekvence genomu se rozumí rekonstrukci příslušného genomu spojením všech readů z množiny R. Dva ready X a Y, které leží v množině R, se překrývají, jestliže prefix X je shodný se sufixem Y nebo prefix Y je shodný se sufixem X. Pokud X a Y pochází z navzájem komplementárních řetězců, pak se překrývají, jestliže převracený komplementární read jednoho z nich překrývá druhý. Tedy jestliže $\overline{X}$ překrývá Y nebo $\overline{Y}$ překrývá X. Problémem detekce překryvů DNA sekvencí se pak rozumí nalezení všech překryvů v množině R (obrázek 4.1). Tato definice předpokládá, že množina readů R obsahuje ready, jenž jsou dokonalou reprezentací genomu, tedy neobsahují žádné chyby vzniklé sekvenováním [20].

Je zřejmé, že pro sestavení genomu je žádoucí detekovat pouze překryvy určité délky. Pro tento účel se zavádí práh detekce překrytí τ , který stanovuje minimální přijatelnou délku překrytí dvou readů [20].

Ve výše uvedené definici se očekávají dokonalé ready, tedy ready neobsahující chyby. Ovšem při sekvenování může docházet ke vzniku chyb. Chyby mohou být též způsobeny velkým množstvím repetitivních elementů. Tyto opakující se sekvence obsahují velké množ-

Obrázek 4.1: Zobrazení tří překrývajících se readů R_1, R_2, R_3 z množiny R [20].

ství mutací. S těmito chybami je nutné při detekci překryvů počítat. Nechť je tedy zavedena hammingova vzdálenost dvou řetězců $h(T, T')$, která je definována jako počet neshod mezi dvěma stejně dlouhými řetězci. Jelikož ovšem řetězce nemusí mít stejnou délku, tak je vhodné mluvit o editační vzdálenosti $ed(T, T')$, která je definována jako minimální počet insercí, delecí a nahrazení symbolů k transformování řetězce T na T'. S využitím této vzdálenosti je poté zaveden práh maximální chyby překrytí e (angl. error rate), který nám udává maximální počet akceptovatelných neshod dvou překrývajících se sekvencí. Může být definován jako maximální povolená fixní editační vzdálenost k nebo maximální relativní editační vzdálenost, jež závisí na délce překrytí.

4.2 Algoritmy využívané nástroji pro sestavování genomu

Existuje celá řada nástrojů pro sestavování genomu. Některé tyto nástroje jsou založeny na detekci překryvů DNA sekvencí. Z tohoto důvodu se tato podkapitola zaměřuje na volně dostupné nástroje pro sestavování eukaryotického i prokaryotického genomu a popisuje algoritmy a přístupy, které tyto nástroje pro detekci překryvů využívají. Tyto algoritmy na svém vstupu očekávají ready, které několika násobně pokrývají genom a neobsahují chyby. Tudíž tyto algoritmy detekují pouze přesné překryvy.

4.2.1 CABOG – Algoritmus přesné shody

Nástroj CABOG využívá pro detekci překryvů algoritmus přesné shody (z angl. Exact match algorithm). Tento algoritmus umožňuje rychle nalezení překrývajících readů pomocí nalezení přesných shod jader (angl. seeds). Tyto shody jader jsou hledány v komprimovaných sekvencích, ve kterých jsou posloupnosti stejných bází redukovány na jednu bází reprezentující tuto posloupnost. Každé jádro je k-mer (podřetězec k bází, defaultně se volí $k = 22$). Algoritmus spočítá počet instancí každého k-meru ve vstupní komprimované sekvenci. K vyhnutí se vysoce opakujících se k-merů dojde k dynamickému výpočtu prahu M s využitím předem vypočítaných tabulek. Následně jsou k detekci překrytí brány v potaz pouze k-mery s výskytem ležícím v intervalu od 2 do M. Algoritmus identifikuje dva ready jako pravděpodobně se překrývající, jestliže spolu sdílí velké množství těchto k-merů. Pomocí dynamického prahu jsou akceptovány vzácně se vyskytující k-mery, ale také často se vyskytující k-mery, které pokrývají značnou oblast. Mezi dvěma identifikovanými ready se podle pozice nejvzácněji se vyskytujícího k-meru určí kotva (anchor). Poté se od této kotvy provede zarovnání, které detekuje chyby v sekvencích a tím umožní jejich opravu. Nakonec jsou na výstup dány opravené překrývající se sekvence, jejichž konce se překrývají alespoň s rozpětím 40 bází a mají alespoň 94% identitu určenou zarovnáním. Zarovnání se provádí pomocí iterativně a paralelně prováděného párového zarovnání [15].

Časová ani prostorová složitost tohoto algoritmu nebyla uvedena. Dá se ovšem z popisu tohoto algoritmu očekávat lineární časová složitost.

4.2.2 SGA – FM-index

SGA pro detekci překryvů DNA sekvencí sestaví obecné sufixové pole z množiny vstupních readů. Následně se z tohoto sufixového pole vytvoří BWT (z angl. Burrows and Wheeleer transformation) a na základě těchto dvou struktur vznikne FM-index. Poté se pomocí tohoto FM-index rekurzivně naleznou všechny překryvy daného readu splňující práh, který určuje délku překrytí. Tento postup se opakuje pro všechny ready ze vstupní množiny readů. Tato metoda pracuje s časovou složitostí $O(N + C)$, kde N je celková délka všech vstupních readů a C je celkový počet nalezených překryvů [20].

Sufixové pole

Sufixové pole je datová struktura, která slouží ke zkrácené reprezentaci všech lexikograficky seřazených sufixů řetězce. Sufixové pole řetězce X, označené SA_X, je permutací celých čísel $1, 2, \ldots, |X|$ takových, že $SA_X[i] = j$ pokud $X[j, |X|]$ je i-tý lexikograficky nejmenší sufix řetězce X. Například pokud $X = AAGTA\$$ pak $SA_X = [6, 5, 1, 2, 3, 4]$. Jelikož je sufixové pole seřazenou datovou strukturou, tak startovní pozice všech instancí vzoru Q v X se bude nacházet na intervalu v SA_X. Tento interval se nazývá interval sufixového pole a je označován dvojicí celých čísel $[l, u]$, kde l je první index v SA_X a u je poslední index v SA_X, jenž odpovídají pozici instance Q v řetězci X. S využitím binárního vyhledávání Q je pak snadno možné určit l a u [20].

FM-index

Jedná se o metodu indexování textu podobnou sufixovému poli, tato metoda ovšem potřebuje značně méně paměti. Metoda umožňuje vypočítat l a u v čase $O(|Q|)$ bez závislosti na velikosti prohledávaného textu. Základem FM-indexu je BWT. BWT řetězce X, označovaná B_X, je permutací symbolů řetězce X takovou, že platí [20]:

$$B_X[i] = \begin{cases} X[SA_X[i-1]], & SA_X[i] > 1 \\ \$, & SA_X[i] = 1 \end{cases}$$

Jinak řečeno $B_X[i]$ je symbol předcházející prvnímu symbolu sufixu začínajícím na pozici SA_X. FM-index rozšiřuje reprezentaci řetězce pomocí BWT přidáním dvou dodatečných datových struktur. Pole C_X, pro které platí, že $C_X(a)$ je počet symbolů v X, které jsou lexikograficky menší než symbol a. A pole Occ_X, pro které platí, že $Occ_X(a, i)$ je počet výskytů symbolu a v $B_X[1, i]$. Jestliže máme řetězec S, pro jehož sufixový interval jsou známy hodnoty l a u. Pak interval pro řetězec aS lze spočítat z $[l, u]$ s využitím polí C_X a Occ_X následovně:

$$l = C_X(a) + Occ_X(a, i-1) \tag{4.1}$$

$$u = C_X(a) + Occ_X(a, u) - 1 \tag{4.2}$$

S využitím rovnic (4.1) a (4.2) lze poté nalézt řetězec Q v řetězci X v čase $O(|Q|)$.

4.2.3 Fermi – FMD-index

Metoda pro detekci překryvů u nástroje Fermi vychází z metody použité v nástroji SGA. Ovšem na rozdíl od předchozí metody tato metoda nevyužívá rekurzi a místo FM-indexu

využívá FMD-index. Rozdíl mezi FMD-indexem a FM-indexem je následující. Metoda použita v nástroji SGA sestaví FM-index pro vstupní množinu readů, tedy pro řetězec $Z = R_0R_1 \dots R_{n-1}$, kde R_0 až R_{n-1} leží v množině vstupních readů R. Metoda použita v nástroji Fermi sestaví FM-index pro řetězec $T = R_0\overline{R_0}R_1\overline{R_1} \dots R_{n-1}\overline{R_{n-1}}$, kde R_0 až R_{n-1} leží v množině vstupních readů R a $\overline{R_0}$ až $\overline{R_{n-1}}$ jsou jejich reverzní komplementární ready. Tento FM-index sestavený pro řetězec T se nazývá FMD-index. Časová i prostorová složitost je velmi podobná předchozí metodě a je lineární [12].

4.2.4 Readjoiner – SPM-relevantní sufixy

Metoda nejprve ze vstupní množiny readů určí množinu readů, které jsou prefix a sufix volné (free), tedy nejsou prefixem nebo sufixem jiných readů. K tomuto účelu dojde nejprve k lexikografickému seřazení všech readů pomocí metody radix sort. Následně se spočítá pole nejdelších společných prefixů lcp (z angl. longest common prefix) tohoto seřazeného seznamu. Podle této hodnoty a délky jednotlivých readů se určí ready, které jsou prefixem sousedního readu. Obdobně se pak ošetří i reverzní komplementy readů a sufixy [7].

Následně se pro tuto sadu readů určí kandidáti na možné shody sufixů a prefixů, které dosahují alespoň požadované délky l_{min}. Vytvoří se pole SPM-relevatních sufixů (z angl. Suffix-Prefix matching relevant suffixes). K tomuto účelu slouží algoritmus, jenž se skládá ze dvou částí: výpočetní fáze a vkládací fáze.

Nejprve se předpočítají lexikograficky seřazené tabulky k-merů, které slouží jako filtry. K-merem rozumíme prefix readu s o délce k. Dále se určí tyto počáteční k-mery všech readů, lexikograficky se seřadí a uloží do tabulky. Následně se odstraní duplicitní k-mery a pro každý k-mer se spočítá počet výskytů. Vzniknou tak dvě tabulky. Tabulka K k-merů, která neobsahuje duplicity. Druhou tabulkou je pak tabulka C, která obsahuje počty výskytů jednotlivých k-merů. Dále se vytvoří dvě tabulky P a Q. P obsahuje počáteční k'-mery, kde pro k' musí platit $k' \leq k$. Q obsahuje všechny k-mery $r[k - k'' + 1 \dots k]$, kde r leží v R (množina readů) a $k'' \leq k$.

Další částí je výpočet všech sufixů všech readů r začínajících na pozici j, kde pro j platí $2 \leq j \leq |r| - l_{min} + 1$, r má $|r| - l_{min}$ takovýchto sufixů. Pro každý takovýto sufix s se určí řetězce $v = s[1 \dots k']$ a $w = s[k - k'' + 1 \dots k]$. Pokud v neleží v P, pak není prefixem žádného readu a tedy s není potenciálním kandidátem a může být vyřazen. Pokud w neleží v Q pak $w \neq [k - k'' + 1 \dots k]$ a tedy s není potenciálním kandidátem a může být vyřazen. Posledním krokem výpočetní fáze je kontrola všech sufixů s, jenž prošly filtry P a Q. Jestliže $u = s[1 \dots k]$ je obsažen v tabulce K, pak s je potenciálním kandidátem a dojde k inkrementaci příslušného počitadla. V druhé vkládací fázi se s využitím výše uvedených tabulek a vyfiltrovaných sufixů za pomoci bufferu a rozdělování do košů vytvoří seznam SPM-relevatní sufixů. Na základě tohoto seznamu se detekují shody prefixů a sufixů a tedy určují se překrývající se ready.

Časová a prostorová složitost jednotlivých částí algoritmu je vždy buď lineární, nebo konstantní. Celková časová i prostorová složitost této metody je lineární. Přesněji je časová složitost $O(m + n + g)$, kde m je počet všech k-merů, n je celková délka všech readů a g je počet počet SPM-relevantních sufixů.

4.3 Další algoritmy pro detekci překryvů

Nástroje pro sestavování genomu provádí na začátku předzpracování vstupních readů, ve kterém dojde k vyfiltrování chyb a repetitivních elementů. Z tohoto důvodu algoritmy, které

využívají k detekci překryvů, pracují s již předzpracovanými ready. Tedy tyto algoritmy neberou zřetel na možné chyby ve vstupních readech, a proto také nejsou schopny detekovat překryvy u readů, které je obsahují. Tyto algoritmy tudíž nejsou příliš vhodné pro možné úpravy a využití pro detekci přibližných překryvů. V následující podkapitole jsou popsány algoritmy, které naopak uvažují možný výskyt chyb a je tedy možné parametricky nastavit práh maximální povolené chyby a to buď fixní nebo relativní. Cílem těchto algoritmů je tedy nalézt všechny překryvy ve vstupní množině readů s ohledem na zadanou minimální délku překryvu a prahu maximální povolené chyby překrytí.

4.3.1 q-gram filtry

Jedním z možných přístupů, který by mohl být využit pro detekci překryvů DNA sekvencí, je přístup založený na q-gramových filtrech. Tyto filtry slouží k nalezení všech lokálních zarovnání mezi dvěma sekvencemi, které jsou delší než-li je daná délka a zároveň nepřekračují práh maximální chyby překrytí e. Tento přístup lze přirozeně rozšířit a aplikovat na problém detekce překryvů DNA sekvencí či-li k řešení sufix/prefix problému.

Filtr je algoritmus, který rychle vyřadí velké části textu (řetězce) s využitím filtračního kriteria. Přičemž zanechá důležité části, které se potenciálně shodují. Ty jsou dále ověřeny na shodu pomocí přesného algoritmu. Tyto dvě fáze se nazývají filtrační fáze a verifikační fáze. Filtr je bezztrátový pokud nikdy nevyřadí skutečnou shodu.

Podřetězcem řetězce délky $q > 0$ nazýváme q-gram. Filtrování vyžaduje předzpracování, kdy je pro cílovou sekvenci vytvořen q-gram index. Tento index se skládá ze dvou tabulek. Z tabulky výskytů, která obsahuje všechny pozice výskytů q-gramů, a z vyhledávací tabulky. Vyhledávací tabulka obsahuje počáteční pozice seznamu výskytů všech q-gramů. Prostorová náročnost indexu závisí na délce indexované sekvence a na základě velikosti slovníku q-gramů. Přesněji je zapotřebí $n + |\Sigma|^q$ integerů, kde n je délka sekvence a Σ je vstupní abeceda [19].

Nalezení lokálního zarovnání mezi dvěma sekvencemi pomocí q-gramových filtrů je založeno na identifikaci oblastí mezi dvěma řetězci, které se přibližně shodují (v závislosti na prahu maximální chyby překrytí e). Tento přístup vychází z předpokladu, že dva přibližně shodující se podřetězce obsahují určité množství společných q-gramů. Základní q-gramová metoda funguje následovně. Za prvé se naleznou všechny q-gramové shody mezi hledaným řetězcem a cílovým řetězcem. Za druhé se identifikují všechny oblasti mezi řetězci, které obsahují dostatečné množství shod. Tyto oblasti jsou označovány jako rovnoběžníky (angl. parallelograms). Následně jsou porovnány pomocí matice editačních vzdáleností a posuvného okénka. Takto nalezení kandidáti jsou dále podrobněji zpracováni ve fázi verifikace. Časová i prostorová složitost tohoto algoritmu je lineární.

4.3.2 Backward backtracking algoritmus

Stejně jako přístup využívaný metodou SGA, který byl popsán v podkapitole 4.2.2, i tento algoritmus staví na FM-indexu místo klasického sufixového stromu. FM-index je totiž prostorově efektivnější strukturou. Základní backtracking algoritmus je schopný nalézt překryvy DNA sekvencí s fixní editační vzdáleností. Pomocí FM-indexu je možné rekurzivně nalézt výskyt vzoru v řetězci bez nutnosti přímého uložení sufixového pole. Jak již bylo zmíněno v podkapitole 4.2.2 k tomuto účelu je využito pole C_X a pole $Occ_X(a, i)$. Tento algoritmus využívající rovnic popsané v podkapitole 4.2.2 se nazývá backward search algoritmus. A je schopný nalézt výskyt řetězce v čase $O(|Q|)$, kde $|Q|$ je délka hledaného vzoru.

Backward backtracking algoritmus, též označovaný kmismatches algoritmus, rozšiřuje backward search algoritmus, tak aby umožnil vyhledávání přibližných výskytů vzoru v řetězci a ne jen přesných výskytů. Tedy rozšiřuje jej tak, aby bylo možné využití fixní editační vzdálenosti. Rozdíl mezi kmissmateches a přesným hledáním je ten, že rekurze zohledňuje inkrementálně, zleva doprava, všechny rozdílné způsoby, jakým může být vzor pozměněn s maximálně k substitucemi. Současně, rekurze udržuje sufixový interval, kde se sufixy shodují s aktuálním modifikovaným sufixem hledaného vzoru [21].

Časová složitost tohoto algoritmu je $O(nt+r)$, kde n je délka všech readů, t je minimální délka překrytí a r je počet všech nalezených překryvů. Existují různé návrhy, jak snížit prostorovou náročnost tohoto algoritmu, avšak ani jeden z nich nelze aplikovat na problém shody sufixu s prefixem.

4.3.3 Sufixové filtry

Sufixové filtry je nová filtrační metoda pro hledání přibližné shody. Metoda sufixových filtrů vychází z dobře známého filtračního algoritmu zvaného faktorové filtry. Faktorové filtry jsou jedním z nejlepších algoritmů pro hledání přibližné shody řetězců, které využívají textový index. Sufixové filtry jsou ovšem silnější metodou, jelikož produkují méně chybných shod než-li faktorové filtry. Také bylo experimentálně dokázáno, že jsou sufixové filtry rychlejší v praxi [21].

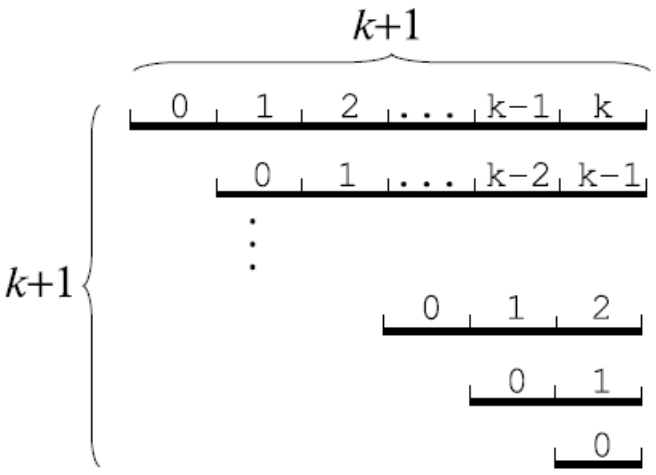
Faktorové filtry jsou založeny na faktorizaci hledaného vzoru. Faktorizací řetězce S rozumíme sekvenci řetězců nazývaných faktory, jejichž konkatenace tvoří řetězec S. V nejjednodušším případě existuje $k+1$ faktorů a výskyt jednoho z nich v prohledávaném řetězci(textu) signalizuje potenciální shodu [8].

Sufixové filtry rozdělí hledaný řetězec na $k+1$ faktorů. Přesněji, nechť je řetězec A rozdělen na neprázdné faktory $A_1A_2...A_{K+1} = A$. Nyní pro libovolný řetězec B existuje optimální faktorizace $B_1B_2...B_{K+1} = B$ taková, že editační vzdálenost $ed(A, B) = \sum_{i\in[i,k+1]} ed(A_i, B_i)$. Interval faktorů $A_iA_{i+1}...A_j$ označujeme $A[i, j]$ pro libovolné $0 < i \leq j \leq k + 1$. Řetězce A a B se shodují na intervalu $[i, j]$ pokud $ed(A[i, j], B[i, j]) \leq j - i$. Dále řetězce A a B se silně shodují na intervalu $[i, j]$, pokud se shodují ve všech prefixech $[i, j']$ kde $j' \in [i, j]$.

V průběhu vyhledávání sufixovými filtry je řetězec B považován za kandidáta pro shodu, jestliže alespoň jeden sufix splňuje podmínku silné shody. Tedy každý sufix $A[i, k + 1]$ je hledán tak, že první faktor A_i se musí přesně shodovat (editační vzdálenost je rovna 0). Když hledání přechází z jednoho faktoru na druhý, tak počet povolených chyb je zvýšen o jedna. Na obrázku 4.2 je znázorněno (kumulativní) maximální počet chyb povolených během vyhledávání. Filtry vloží na výstup kandidáty, u kterých alespoň jeden z $k + 1$ filtrů určí výskyt A se vzdáleností do k. Na závěr musí všichni potenciální kandidáti projít fází ověřování, jelikož filtrovací fáze může nalézt kandidáty, jejichž editační vzdálenost je větší než k.

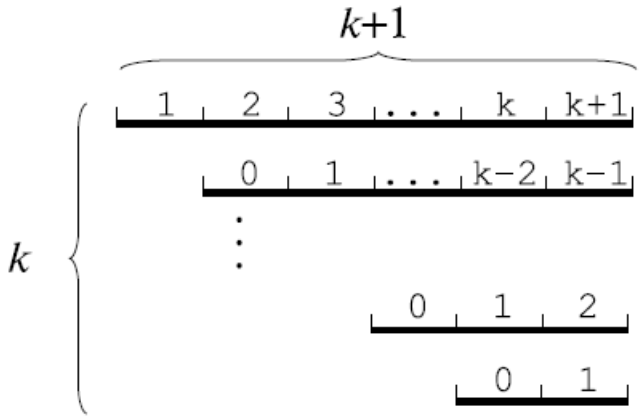
Bylo dokázáno, že sufixové filtry jsou jedny z nejsilnějších filtrů. Trik spočívá ve volbě posledního faktoru tak, aby byl delší než-li ostatní a tím bylo zajištěno, že podmínka silné shody nebude na nejkratším sufixu $[k + 1, k + 1]$ produkovat příliš mnoho kandidátů. Avšak sufixové filtry nejsou přímo vhodné pro detekci přibližných překryvů. Právě z důvodu nemožnosti kontrolovat délku posledního faktoru, který může mít v nejhorším případě délku 1. Sufixové filtry je ovšem možné upravit, tak aby hledaly přibližné překryvy sufixů a prefixů, jejichž editační vzdálenost závisí na délce překryvu, a to následovně.

Nechť jsou opět dány řetězce A a B, interval $[i, j]$ se slabě shoduje, jestliže platí $ed(A[i, j'], B[i, j']) \leq ij' - i + 1$ pro všechny $j' \in [i, j]$. Řetězec B je považován za kan-



Obrázek 4.2: Sufixové filtry rozdělí řetězec na k + 1 faktorů. Každý sufix faktoru je poté hledán odděleně. Čísla odpovídají (kumulativnímu) maximálnímu počtu chyb povolených v průběhu hledání [21].

didáta, jestli shoda splňuje slabou podmínku shody na intervalu $[1, k + 1]$ nebo silnou podmínku shody na intervalu $[i, k + 1]$ pro libovolné $i \in [2, k]$ [21]. Obrázek 4.3 znázorňuje (kumulativní) maximální počet chyb povolených během vyhledávání s těmito podmínkami. Slabá podmínka schody odpovídá nejdelšímu sufixu (nejvyšší filtr v obrázku) a silná podmínka shody odpovídá zbývajícím $k - 1$ filtrům v obrázku. Slabá podmínka shody se může zdát jako chybná, jelikož povoluje k + 1 chyb pro poslední faktor. Ovšem je nezbytná pro inkrementální hledání přibližných překryvů.



Obrázek 4.3: Sufixové filtry modifikované pro hledání překryvů. Hledání prvního sufixu již umožňuje jednu chybu od samého začátku. Poslední sufix není zahrnut do hledání [21].

Časová složitost je velmi podobná algoritmu backward backtracking. Závisí tedy jak na délce všech vstupních readů a prahu minimální délky překrytí, tak i na počtu detekovaných

překryvů. Tato závislost je lineární. Prostorová složitost je poté také lineární, ovšem dle [21] lepší nežli u q-gramů.

4.4 Shrnutí algoritmů pro detekci překryvů

V tabulce 4.1 je uveden přehled nástrojů s příslušnými metodami a jejich časová složitost (pokud byly tyto informace dostupné). Jsou zde uvedeny pouze volně dostupné nástroje, které slouží k sestavování genomu prokaryotických i eukaryotických organismů pomocí overlap-layout consensu, jenž využívá při sestavování překryvy readů. Přehled těchto nástrojů vychází z článku [6]. Je zde také uveden nástroj RepeatExplorer [17], který slouží k rekonstrukci repetitivních segmentů v DNA. Jelikož algoritmy využívané těmito nástroji pracují s ready, které neobsahují chyby a repetice, tak nejsou vhodné pro další využití či úpravy. V tabulce 4.2 je tedy uveden přehled algoritmů, u kterých je možné parametricky nastavovat minimální práh překrytí a práh maximální povolené chyby překrytí, jenž může být zadán přímo počtem neshod nebo relativní hodnotou určují editační vzdálenost v závislosti na délce překrytí. Jsou zde o nich uvedeny následující informace: jejich časová složitost, způsob zadání prahu maximální chyby překrytí a indexová struktura, kterou využívají.

Název	**Sekvenovací platforma**	**Metoda**	**Časová složitost**	
RepeatExplorer	-	Každý s každým	Kvadratická	[17]
Celera assembler CABOG wgs-assembler	Sanger, Illumina, 454, Ion Torrent, Pacific Biosciences	Algoritmus přesné shody	Neuvedeno	[15]
Forge	Hybridní - Sanger, 454 a Illumina	Hashovací tabulky, statistický model, kvantitativní metoda	Neuvedeno	[4]
SGA	Illumina	FM-index	Lineární	[20]
Readjoiner	Illumina	SPM-relevatní sufixy	Lineární	[7]
Fermi	Illumina	FMD-index	Lineární	[12]

Tabulka 4.1: Přehled nástrojů pro sestavování genomu a jejich metod, které využívají pro detekci překryvů.

Název	**Editační vzdálenost**	**Indexová struktura**	**Časová složitost**	
q-gram filtry	Relativní	q-gram index	Lineární	[19]
Backward backtracking	Fixní	FM-index	Lineární	[21]
Sufixové filtry	Relativní nebo fixní	FM-index	Lineární	[21]

Tabulka 4.2: Přehled algoritmů pro detekci přibližných překryvů.

Hlavní výhodou všech uvedených přístupů a algoritmů (s výjimkou nástroje RepeatExplorer) je, že pracují s lineární časovou složitostí. Liší se především ve způsobu práce s pamětí a s ready. U nástrojů pro sestavování genomu prochází množina readů fází předzpracování, která z této množiny odstraní ready, jenž obsahují chyby. Nevýhodou výše uvedených metod proto je, že na svém vstupu očekávají množinu readů, které neobsahují žádné chyby, a tedy

na rozdíl od nástroje RepeatExplorer nejsou schopny detekovat překryvy u readů, které je obsahují. Sufixové filtry, q-gramy a algoritmus Backward backtracking ovšem umožňují detekci překryvů i u readů s chybami.

Z tohoto důvodu jsou vhodnější pro další možné využití a vylepšení. Nejnovějším algoritmem jsou sufixové filtry. V článku [21] bylo experimentálně dokázáno, že jsou prostorově efektivnější než-li q-gramové filtry. Navíc na rozdíl od algoritmu backward backtracking umožňují využití editační vzdálenosti závislé na délce překrytí. Z těchto důvodu je vhodné se zaměřit právě na sufixové filtry a provést jejich vylepšení. Především by bylo vhodné se zaměřit na filtrující podmínky a způsob rozdělení hledaného vzoru na faktory. Filtrující podmínky, velikost a počet faktorů ovlivňuje počet určených potenciálních kandidátů na shodu. Množství potenciálních kandidátů má výrazný vliv na prostorovou složitost, ale i na časovou, jelikož každý kandidát musí být ve fázi verifikace ověřen, aby nedocházelo k chybné detekci překryvů.

Kapitola 5

Návrh algoritmu

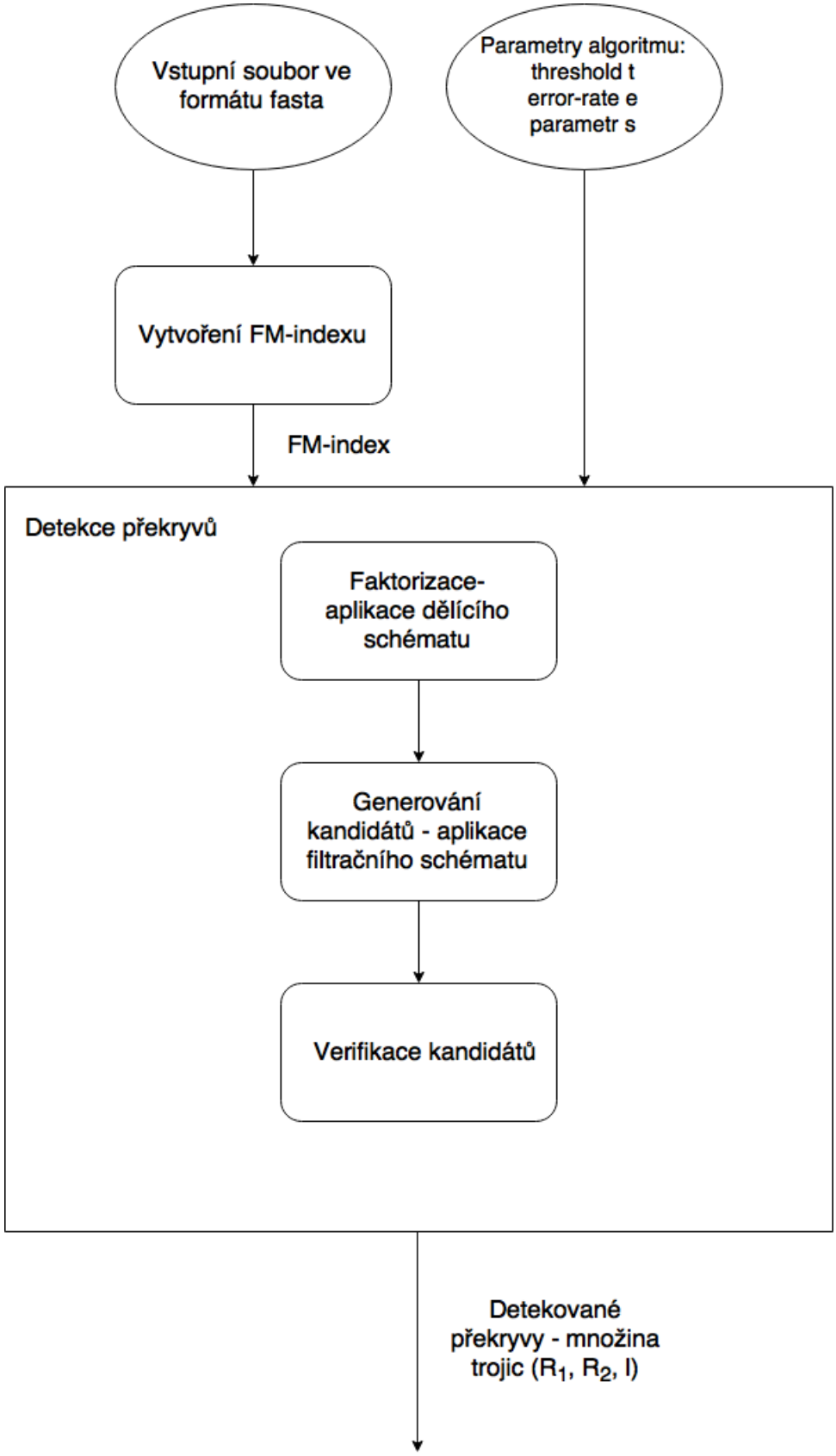
Nejvhodnějším algoritmem z algoritmů pro detekci překryvů popsaných v kapitole 4 se jeví být sufixové filtry. Sufixové filtry umožňují parametricky nastavit velikost minimálního prahu překrytí a také zadat práh maximální povolené chyby překrytí. Práh maximální chyby překrytí může být zadán přímo v podobě počtu neshod, tedy fixní editační vzdálenosti nebo v podobě relativní editační vzdálenosti, hodnoty v rozmezí 0 až 1, z které se následně dopočítává editační vzdálenost v závislosti na délce překrytí.

Algoritmus založený na principu sufixových filtrů popsaný v podkapitole 4.3.3 lze dle článku [19] dále upravit a tím zmenšit počet generovaných kandidátních řešení. Menší počet kandidátních řešení vede jak k lepší prostorové složitosti, tak i k časové. Každého potenciálního kandidáta je nutné ve fázi verifikace ověřit na shodu, aby se zabránilo potenciální chybě při detekci překryvů, nižší počet kandidátu tedy znamená nižší časové nároky na ověřování shody. Tyto úpravy jsou popsány v následujících podkapitolách 5.3 a 5.5. Jedná se celkem o dvě změny. První změnou je nové filtrační schéma. Druhou změnou je nové schéma dělení vzoru na faktory, kdy na rozdíl od původní varianty je počet a velikost faktorů různá v závislosti na parametru s a editační vzdálenosti.

5.1 Schéma algoritmu

Na obrázku 5.1 je popsáno schéma navrženého algoritmu pro detekci přibližných překryvů DNA sekvencí využívající sufixové filtry s novým filtračním a faktorizačním schématem. Na vstupu celého algoritmu je očekáván soubor ve formátu fasta, který obsahuje množinu vstupních readů. Formát fasta je podrobněji popsán v podkapitole 5.2. Tento vstupní formát je zpracován a z množiny vstupních readů je vytvořen FM-index (viz podkapitola 4.2.2). Po vytvoření FM-indexu je možné již přikročit k samotné detekci překryvů. Samotný algoritmus pro detekci překryvů na svém vstupu kromě již vytvořeného FM-indexu očekává několik důležitých parametrů. Parametr nazývaný threshold (t) čili práh minimální délky překrytí, error-rate (e) nebo-li práh maximální povolené chyby překrytí a parametr s, který jak již bylo zmíněno ovlivňuje délku a počet faktorů při faktorizaci.

Prvním krokem při samotné detekci překryvů je rozdělení vstupních readů na části (faktory) dle níže popsaného dělícího schématu. Následně jsou pomocí filtrů a kontroly filtračních podmínek generováni potenciální kandidáti na schodu. V samotném závěru algoritmu je fáze verifikace, v niž jsou ověření potenciální kandidáti. Na výstupu celého algoritmu je poté množina detekovaných překryvů v podobě (R_1, R_2, l), kde R_1 a R_2 jsou překrývající se ready a l je délka samotného překryvu.



Obrázek 5.1: Schéma algoritmu pro detekci překryvů DNA sekvencí.

5.2 Vstup algoritmu

Vstupní množina readů, které mohou být i různé délky, tedy množina DNA sekvencí, je vložena na vstup algoritmu v podobě vstupního souboru. Nejvhodnějším a nejjednodušším formátem pro uložení množiny DNA sekvencí je formát FASTA. Formát FASTA je bioinformatický textový formát pro reprezentaci jak nukleotidových sekvencí, tak i peptidových, v kterých jsou nukleotidy nebo aminokyseliny reprezentovány jednoznakovými kódy. Původně tento formát pochází ze softwarového balíku FASTA, ovšem díky své jednoduchosti se stal standardem. Záznam ve formátu FASTA se skládá ze dvou řádků. První řádek začíná znakem '>'. Za tímto znakem následuje název nebo unikátní identifikátor sekvence. Také se zde mohou nacházet doplňující informace. Na druhém řádku se poté nachází samotná sekvence. Příklad záznamů ve formátu FASTA je znázorněn níže. Formát FASTA má celou řadu rozšíření, nejvýznamnější je formát FASTAQ obsahující navíc i skóre kvality. Příklad vstupu:

```
>SEQUENCE_1
ATGAGAGCCCTCACACTCCTCGCCCTATTGGCCCTGGCCGCACTTTGCATCGCTGGCCAGGCAGGTGAGTGCCCC
CACCTCCCCTCAGGCCGCATTGCAGTGGGGGCTGAGAGGAGGAAGCACCATGGCCCACCTCTTCTCACCCCTTTG
GCTGGCAGTCCCTTTGCAGTCTAACCACCTTGTTGCAGGCTCAATCCATTTGCCCCAGCTCTGCCCTTGCAGAGG
>SEQUENCE_2
GATCTCCGACGAGGCCCTGGACCCCCGGGCGGCGAAGCTGCGGCGCGGCGCCCCCTGGAGGCCGCGGGACCCCTG
GCCGGTCCGCGCAGGCGCAGCGGGGTCGCAGGGCGCGGCGGGTTCCAGCGCGGGGATGGCGCTGTCCGCGGAGGA
CCGGGCGCTGGTGCGCGCCCTGTGGAAGAAGCTGGGCAGCAACGTCGGCGTCTACACGACAGAGGCCCTGGAAAG
CTCTCGCAGGACCTTCCTGGCTTTCCCCGCCACGAAGACC
>SEQUENCE_3
CGCGCTGTCCGCGCTGAGCCACCTGCACGCGTGCCAGCTGCGAGTGGACCCGGCCAGCTTCCAGGTGAGCGGCTG
CCGTGCTGGGCCCCTGTCCCCGGGAGGGCCCCGGCGGGGTGGGTGCGGGGGGCGTGCGGGGCGGGTGCAGGCGAG
TGAGCCTTGAGCGCTCGCCGCAGCTCCTGGGCCACTGCCTGCTGGTAACCCTCGCCCGGCACTACCCCGGAGACT
```

5.3 Nové filtrační schéma

Původní filtrační schéma vynechává z filtračního schématu filtry $filter_{k,k}$. Z důvodu pokrytí všech případů rozdělení, které byly pokryty těmito filtry, jsou upraveny filtry $filter_{k,1} = 01 \ldots k$ na filtry $filter_{k,1} = 12 \ldots (k+1)$. V důsledku této úpravy jsou poté pokryty všechny kombinace rozdělení. Filtry $filter_{k,1}$ tedy začínají již s jednou povolenou chybou, zatímco ostatní filtry na začátku nepovolují žádnou chybu. Také musely být upraveny (zpřísněny) filtrační podmínky (viz kapitola 4.3.3) .

Nové filtrační schéma se skládá z filtrů $filter_{k,1}, \ldots, filter_{k,k-1}$, kde filtry $filter_{k,i}$ jsou sekvencí délky $k-i+1$. Prvních $k-i$ elementů těchto filtrů je $0, 1, \ldots, (k-i-1)$ a posledním elementem je $k-i-1$. Například pro $k = 4$ filtry jsou $filter_{4,1} = 0122$, $filter_{4,2} = 011$ a $filter_{4,3} = 00$. Tyto filtry pokrývají všechny možné vzdálenosti rozdělení pro případ 4 částí (k = 4) a povolené chybě překrytí 2 ([el] = 2). Filtr 0122 pokrývá případy 0020, 0002, 0101 a 0110. Filtr 011 pokrývá případy 0011, 1001 a 1010. Na závěr filtr 00 pokrývá případy 2000, 0200 a 1100. Tedy opravdu tyto filtry pokrývají všechny možné vzdálenosti rozdělení. Toto nové filtrační schéma vyžaduje rozdíl alespoň 2 mezi počtem částí v rozdělení řetězce $S[1..l]$ a $[el]$. V původním filtračním schématu byl vyžadován rozdíl alespoň 1 [11].

V původním filtračním schématu je filtr $filter_{k,i}$ prefixem filtru $filter_{k',i}$ pro všechny $k < k'$ a i. Toto nové filtrační schéma ovšem nemá tuto vlastnost a tak je nutné filtrační podmínky doplnit o další podmínku, která ověří jestli skutečně řetězec S patří do množiny kandidátu na shodu. Pro zjednodušení definice nové podmínky nechť je zavedena vzdálenost rozdělení $pd(A, B)$ definovaná jako sekvence celých čísel o velikosti k, ve které $pd(A, B)[i] = ed(A_i, B_i)$. Dále pak je zavedeno nové označení akumulovaná vzdálenost rozdělení $apd(S, S')$, která je tvořena sekvencí $PrefixSum(pd(A, B))$. Tedy $apd(A, B)[i]$ je celkový počet neshod mezi prvními i částmi řetězců A a B. S touto nově definovanou vzdáleností lze poté novou podmínku definovat následovně. Za předpokladu, že jsou splněny podmínky předchozího schématu (slabá a silná podmínka shody viz kapitola 4.3.3) a zároveň platí $apd(S', B) \leq filter_{k,i}$, kde S' je prefix délky $|B|$ a k je počet částí v rozdělení řetězce S' indukovaného rozdělením řetězce (vzoru) S, pak řetězec (sufix) B leží v množině kandidátu na shodu.

5.4 Nové dělící schéma

Efektivita algoritmu využívající sufixové filtry závisí na velikosti jednotlivých části řetězce (faktorů). Větší faktory redukují menší počet kandidátů na shodu. Pro správný chod algoritmu musí rozdělení řetězce S (readu) splňovat následující vlastnost. Pro každé $l \geq t$ musí počet částí řetězce $S[1..l]$ indukovaných rozdělením řetězce S být alespoň $[el] + s$, kde s je u původního schématu rovno 1, v tomto novém schématu je voleno a doporučeno 2, ale může být parametricky zadáno [11].

V původním dělícím schématu byl řetězec S s výjimkou poslední části rozdělen do stejně velkých částí velikosti p. Velikost p se volila jako nejmenší celé číslo, které splňuje výše uvedenou podmínku. V novém schématu nemají všechny části shodnou velikost a velikost většiny částí je větší než-li u původního schématu z tohoto důvodu by mělo nové schéma garantovat efektivnější chod algoritmu.

Nechť S je řetězec, který má být rozdělen. Dále nechť $l_0 < l_1 < \cdots < l_q$ jsou všechny indexy v rozmezí $l_{min}, \ldots, |S|$, pro které platí $[e(l-1)] < [el]$ a nechť $l_{q+1} = |S| + 1$. Dále $k = [el_0] + s - 1$. Řetězec S je pak rozdělen následovně. Velikost prvních k částí je zvolena tak, aby byly splněny následující dvě podmínky. Za prvé celková velikost prvních k částí byla $l_0 - 1$. Za druhé délka k-té části byla alespoň $l_0 - t$. Ke splnění těchto dvou podmínek je možné nastavit délku k-té části na $L = max([l_0 - 1/k], l_0 - l_{min})$. Velikost prvních $k - 1$ částí na p nebo $p + 1$, kde $p = (l_0 - 1 - L)/(k - 1)$. Délka zbývajících částí je pak $l_1 - l_0, l_2 - l_1, \ldots, l_{q+1} - l_q$.

Pro lepší pochopení dělícího schématu nechť je zde uveden příklad možného rozdělení. Nechť je velikost děleného řetězce S rovna 200 symbolů a parametry jsou t = 40 a e = 0,1. Pokud bude s = 1, pak původní schéma dělení rozdělí vstupní řetězec na 8 stejně velkých částí. V novém dělícím schématu bude mít prvních 5 částí velikost 8 a zbývající části velikost 10 (pro s = 2).

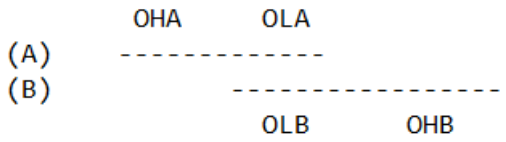
5.5 Výstup algoritmu

Algoritmus v průběhu své činnosti vypisuje na standardní výstup jeden řádek na jeden detekovaný překryv. Tento výstup má následující tvar:

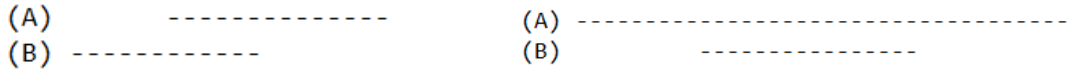
```
idA idB O OHA OHB OLA OLB K
```

kde *idA* je číslo prvního readu *A*. *idB* je číslo druhého readu *B*. Vždy platí, že $idA \leq idB$. O je znak určující orientaci. Symbol *N* označuje normální orientaci a symbol *I* označuje inverzní orientaci. Read *A* je vždy normálně orientován. Překrývá se buď s normálně orientovaným readem *B*, v takovém případě je O rovno *N*, nebo s jeho komplementární verzí, pak je *O* rovno *I*.

Sloupce *OHA, OHB, OLA, OLB* slouží k popisu překryvu readů A a B (obrázek 5.2). *OHA* určuje počet znaků readu *A* nacházejících se před překryvem. *OHA* může nabývat i záporných hodnot. Důvodem je, že vždy platí $idA \leq idB$, ovšem může nastat situace, kdy se bude shodovat prefix readu *A* se sufixem readu *B* (obrázek 5.3a). V tomto případě bude *OHA* nabývat negativní hodnoty. *OHB* je počet znaků readu *B* nacházejících se za detekovaným překryvem. Opět může nabývat i záporných hodnot a to v situaci, kdy read *B* skončí před readem *A* (obrázek 5.3b). Obvykle tato situace nastává v důsledku povolených chyb mezi překryvy.



Obrázek 5.2: Názorné zobrazení významu hodnot ve sloupcích *OHA, OHB, OLA, OLB*.



(a) (b)

Obrázek 5.3: Příklady kdy OHA (a) a OHB (b) nabývají záporných hodnot.

OLA, OLB vyjadřují samotnou délku překryvu. Délka překryvu pro read *A* a read B se může lišit. Toto je způsobeno v důsledku možných delecí a inzercí v readech. Poslední hodnota K na výstupu určuje počet chyb v zarovnání překryvů.

Kapitola 6

Implementace

K článku [21], kde byly poprvé navrhnuty a použity sufixové filtry k detekci překryvů DNA sekvencí, je také volně dostupná implementace navrženého algoritmu. Tento algoritmus využívá filtrační a dělící schéma popsané v kapitole 4.3.3. V kapitole 5 je dle článku navržen algoritmus, který z této implementace vychází a dle článku [19] jej doplňuje o nové filtrační a dělící schéma. Tyto úpravy původního algoritmu byly v tomto článku ovšem popsány pouze teoreticky a nebyla k nim poskytnuta implementace. Navíc provedené experimenty nebyly provedeny na reálných datech a tudíž získané závěry v tomto článku nemusí být reálné a důvěryhodné. Z tohoto důvodu byla využita dostupná implementace sufixových filtrů pro detekci překryvů DNA sekvencí. A tato práce se zaměřuje na úpravu této implementace, kterou doplňuje o nové filtrační a dělící schéma.

Algoritmus sufixových filtrů je implementován v jazyce C++ v podobě konzolové aplikace. Skládá se ze dvou základních částí. První část sestaví ze vstupního souboru FM-index, aby nemuselo docházet před každým spuštěním detekce překryvů k opětovnému vytváření tohoto indexu ze vstupních dat. Druhá část již implementuje samotný algoritmus detekce překryvů DNA sekvencí. Obě tyto implementace využívají pro práci datové struktury z knihovny *libcds*[1].

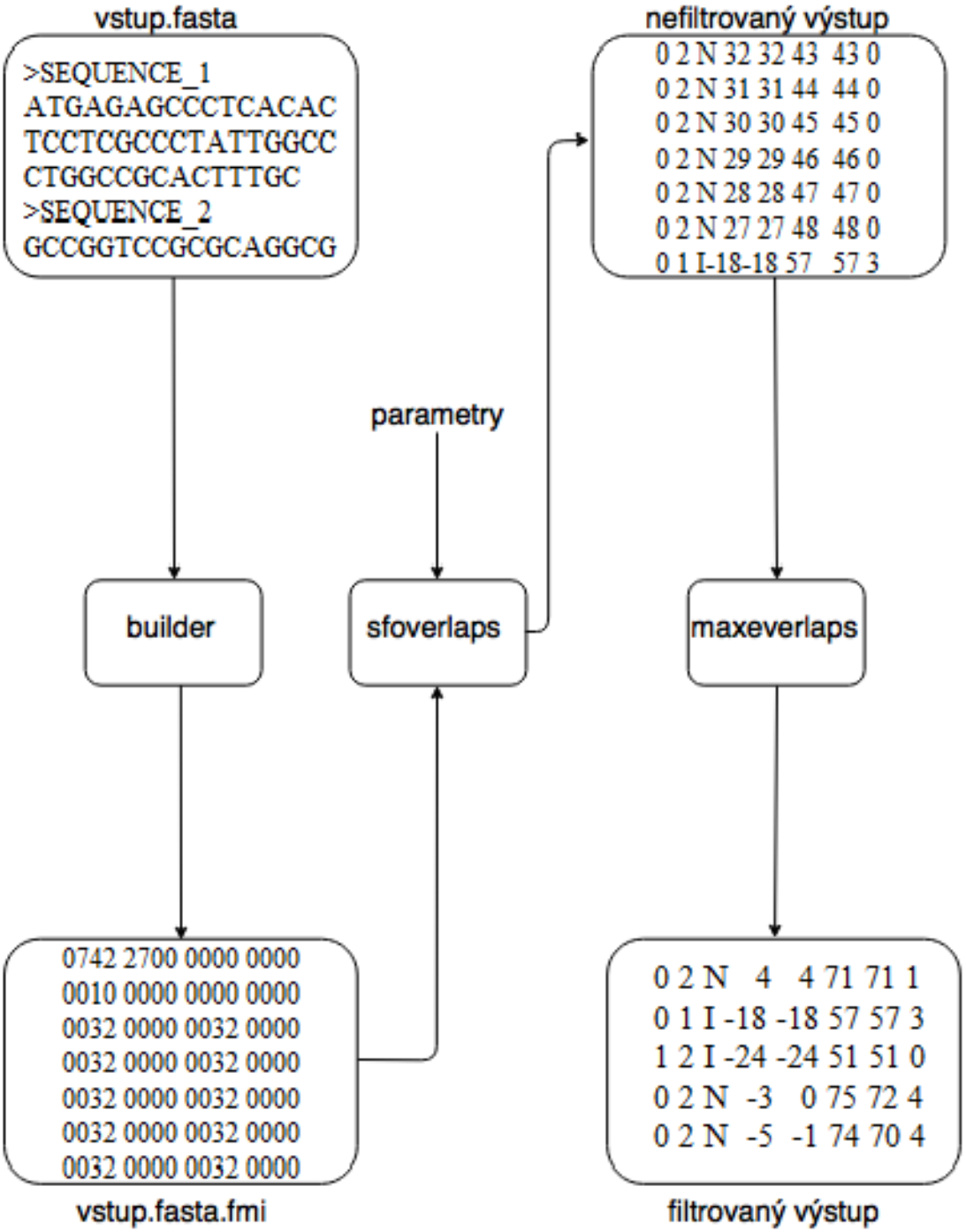
6.1 Celkové schéma nástroje pro detekci překryvů

Na obrázku 6.1 je uvedeno celkové schéma nástroje pro detekci překryvů DNA sekvencí. Nejprve je zpracován vstupní soubor ve formátu FASTA pomocí nástroje *builder*. Pomocí něj je vytvořen FM-index. Následně je pomocí nástroje *sfoverlaps* provedena samotná detekce překryvů, jenž na standardní výstup vypisuje jednotlivé překryvy. Tyto překryvy jsou poté pomocí nástroje *maxoverlaps* vyfiltrovány pouze na nejdelší překryvy pro každý pár readů.

6.2 Sestavení FM-indexu

Aby se předešlo nutnosti vždy opakovaně vytvářet FM-index ze stejné množiny vstupních readů, tak je toto zpracování vstupního souboru odděleno od samotné detekce překryvů DNA sekvencí. K tomuto účelu byl vytvořen nástroj *builder*. Tento nástroj postupně zpracovává řádek po řádku vstupní soubor ve formátu FASTA. Po načtení sekvence DNA provede transformaci této sekvence. Malá písmena převede na velká (pokud již nejsou) a neplatné

[1] http://code.google.com/p/libcds/



Obrázek 6.1: Celkové schéma nástroje pro detekci překryvů.

znaky nahradí symbolem 'N'. Následně ještě provede kontrolu zda-li se opravdu jedná o sekvenci nukleových kyselin a tím zabrání možné chybě při vytváření FM-indexu. Poté provede s využitím třídy *TextCollectionBuilder* sestavení FM-indexu. Tedy vytvoří pro vstupní sekvence BWT (Burrows-Wheelerovu transformaci) a sufixové pole z nichž následně vypočítá pole C_X a Occ_X. Tím vytvoří FM-index pro vstupní sekvence DNA (množinu readů). Tento FM-index následně uloží do binárního souboru s příponou *fmi*.

Zdrojové soubory

- **builder.cpp** - hlavní metoda programu.
- **TextCollectionBuilder.cpp a TextCollectionBuilder.h** - vytvoření FM-indexu.
- **adresář incbwt** - obsahuje všechny doplňující třídy.

Vstupní parametry

Nástroj *builder* na svém vstupu očekává jeden povinný vstupní parametr a to název vstupního souboru. Dále pak je možné zobrazit pomocí parametru *-h* nápovědu nebo pomocí parametru *-v* zapnout průběžný informativní výpis. Další volitelný parametr je *-c*, který určuje, že na vstupu jsou ready v SOLiD color kódech. Posledním parametrem je *-s*, pomocí nějž se určuje vzorkovací frekvence při sestavování FM-indexu, nižší hodnota vede k menšímu FM-indexu, ale může snížit prohledávací čas. Výchozí a doporučená hodnota je 16.

- **<vstup>** - název souboru ve formátu FASTA
- **-c,** –**color** - určuje, že vstupní ready jsou v SOLiD color kódech.
- **-v,** –**verbose** - zapne výpis informací na standardní chybový výstup.
- **-s <int>,** –**sample-rate <int>** - vzorkovací frekvence pro index (výchozí hodnota: 16)
- **-h,** –**help** - zobrazí nápovědu.

Příklady spuštění

- *./builder input.fasta*
- *./builder -h*

6.3 Detekce překryvů

Program pro detekci překryvů se nazývá *sfoverlap* a je spouštěn s parametry uvedenými v předcházející podkapitole 6.3. Hlavní metoda se nachází v souboru *sfoverlap.cpp*. Nejprve v této metodě dojde ke zpracování parametrů. Následně je otevřen vstupní soubor obsahující FM-index. Tento soubor je zpracován pomocí třídy *TCImplementation*, která implementuje rozhraní *TextCollection*. Po zpracování vstupního souboru jsou v této třídě do příslušných proměnných uloženy struktury FM-indexu a také vstupní množina readů společně s dalšími doplňujícími informacemi, jako je například velikost vstupní množiny readů. Pro uložení množiny vstupních readů je využita třída *TextStorage*.

Jakmile je zpracován vstupní soubor s FM-indexem, tak je v závislosti na vstupních parametrech vytvořena instance jedné z následujících tříd: *SuffixFilterOverlap*, *SuffixFilterOverlapEd*, *MismatchOverlap*, *EditDistanceOverlap*. Všechny tyto třídy implementují rozhraní *OverlapQuery*. V těchto třídách je implementována samotná detekce překryvů. *MismatchOverlap* a *EditDistanceOverlap* implementují detekci překryvů pouze pomocí algoritmu Backward backtracking. Důležitějšími a zajímavějšími třídami jsou třídy *SuffixFilterOverlap* a *SuffixFilterOverlapEd*. Rozdíl mezi těmito třídami je ten, že třída *SuffixFilterOverlap* detekuje překryvy na základě fixní hodnoty povolených neshod překrytí, naopak třída *SuffixFilterOverlapEd* na základě relativního maximálního prahu překrytí. Z Tohoto relativního prahu překrytí je dopočítávána editační vzdálenost v závislosti na délce překrytí.

Samotná detekce překryvů pomocí sufixových filtrů je realizována následovně. V cyklu jsou postupně zpracovány jednotlivé vstupní ready. Pro každý vstupní read je zavolána metoda *SuffixPrefixMatch*. Na začátku této metody dojde k rozdělení zpracovávaného readu na faktory, dle schématu popsaného v kapitole 5.5. Indexy pozic, na kterých dochází k dělení, jsou uloženy do pomocného pole. Poté je implementována kontrola filtračních podmínek (viz kapitola 4.3.3 a 5.3) pomocí backtracking algoritmu, tedy prohledáváním FM-indexu. Přičemž dochází mezi faktory k inkrementaci maximální povolené chyby dle schématu popsaného v podkapitole 5.3. Pseudokód backtracking algoritmu je uveden níže (algoritmus 1). Parametry této rekurzivní procedury jsou následující: *P[1,m]* je zpracovávaný vzor, m je délka vzoru, k je počet povolených chyb, který se inkrementálně mění, j je ukazatel do *P[1,m]*, *sp* a *ep* jsou indexy do sufixového pole. První zavolání pak je *backtracking(P, k, m, 1, n)* , kde n je délka prohledávaného textu (délka všech readů). Po ověření filtračních podmínek a nalezení potenciálního kandidáta je zkontrolována skutečná shoda překryvu pomocí metody *checkMatch*. Tato kontrola je realizována pomocí kontroly shody znaků znak po znaku, přičemž je počítán počet neshod a ověřen s povoleným počtem neshod. Celý tento postup je aplikován i na komplementární read.

Nástroj je také pomocí parametru možné spustit v paralelním módu. Paralelní počítání bylo implementováno pomocí API *OpenMP*.

Hlavní zdrojové soubory

- **sfoverlap.cpp** - hlavní metoda programu.
- **TextCollection.cpp a TextCollection.h** - rozhraní pro práci s FM-indexem.
- **TCImplementation.cpp a TCImplementation.h** - implementace rozhraní pro práci s FM-indexem.
- **TextStorage.cpp a TextStorage.h** - uložení vstupní množiny readů.
- **OverlapQuery.h** - rozhraní pro detekci překryvů.
- **EditDistanceOverlap.h** - detekce překryvů s fixním počtem neshod, zadané pomocí editační vzdálenosti - pouze backward backtracking algoritmus.
- **MismatchOverlap.h** - detekce překryvů s fixním počtem neshod , zadané pomocí počtu neshod - pouze backward backtracking algoritmus
- **SuffixFilterOverlap.h** - detekce překryvů sufixovými filtry- fixní počet neshod.
- **SuffixFilterOverlapEd.h** - detekce překryvů sufixovými filtry- relativní práh překrytí.

Algorithm 1 backtracking($P[1,m], k, j, sp, ep$)

```
if sp > ep then
  return
end if
if j = 0 then
  return SA[sp], ..., SA[ep]
end if
for all s ∈ Σ do
  sp' ← C[s] + rank_s(L, sp − 1) + 1
  ep' ← C[s] + rank_s(L, ep)
  if P[j] ≠ s then
    k' ← k − 1
  else
    k' ← k
  end if
  if k' > 0 then
    backtracking(P, k', j -1, sp', ep')
  end if
end for
```

Vstupní parametry

Program na svém vstupu očekává řadu parametrů. Část těchto parametrů je povinná, jelikož jsou nezbytné pro správný běh programu a je pomocí nich nastaven způsob detekce překryvů. Tyto parametry mají velký vliv na běh celého algoritmu. Nejdůležitějším parametrem je název vstupního souboru obsahující FM-index. Název souboru odpovídá pouze vstupnímu souboru ve formátu FASTA. Přípona *.fmi* je doplněna automaticky před započetím zpracování tohoto souboru. Nejdůležitějšími dvěma parametry, které jsou tudíž povinné, jsou parametry *-e* a *-k*. Tyto dva parametry se navzájem vylučují. První z jmenovaných určuje relativní maximální povolený práh překrytí, pomocí něhož se v závislosti na délce překrytí vypočítává povolená editační vzdálenost. Druhý pak určuje fixní počet povolených neshod pro překryvy všech délek. Další parametr poté určuje způsob zarovnání, tedy jestli bude povolovat pouze neshody nebo i delece a inzerce. Implicitně je nastavená druhá varianta. Důležitým parametrem je také *-s*, který ovlivňuje počet faktorů (viz kapitola 5)

Obecnými parametry poté je maximální práh překrytí *-t*, parametr *–skip*, který určuje kolik prvním readů bude přeskočeno, dále *–nreads* určující kolik readů ze vstupního souboru bude zarovnáno. Parametr *-c* indikuje situaci, kdy na vstupu jsou ready ze sekvenovací technologie SOLiD, tedy jsou v SOLiD color kódech. Posledními dvěma parametry je parametr vyvolávající nápovědu *–help* a parametr *-v*, který způsobí výpis informací o běhu algoritmu, jako je počet readů, doba běhu a podobné. Pro zpracování parametrů byla využita knihovní metoda *getopt_long*. Přehled uvedených parametrů je uveden níže.

- **<index>** - název souboru obsahujícího FM-index.
- **-e <int>** - určuje relativní maximální práh překrytí, jenž je dán 1/<int>. Například jestliže je zadáno -e 20, pak relativní maximální práh překrytí je dán 1/20 = 0,05.
- **-k <int>** - určuje fixní počet povolených chyb pro všechny překryvy. Doporučené jsou malé hodny. Nejlépe menší než 4.

- **-s <int>** - na základě parametru s je vypočítán počet faktorů. Doporučené hodnoty jsou 2 nebo 3.
- **–indels** - povoluje neshody, inzerce a delece při zarovnání.
- **–mismatch** - povoluje pouze neshody při zarovnání.
- **-t <int>, –threshold <int>** - určuje minimální práh překrytí. Pouze překryvy délky větší nebo rovny tomuto prahu jsou detekovány. Výchozí hodnota je 40.
- **–skip <int>** - algoritmus přeskočí prvních <int> readů.
- **–nreads <int>** - algoritmus zpracuje pouze <int> prvních readů. Výchozí hodnota jsou všechny ready.
- **-c, –color** - určuje, že vstupní ready jsou v SOLiD color kódech.
- **-v, –verbose** - zapne výpis informací na standardní chybový výstup.
- **-h, –help** - zobrazí nápovědu.
- **-P <int>, –parallel <int>** - počet paralelních vláken, které se využijí.

Příklady spuštění

- *./sfoverlap -e 20 -t 40 -s 2 input.fasta*
- *./sfoverlap -k 2 -t 40 -s 2 -v input.fasta*
- *./sfoverlap -mismatch -e 20 -t 40 input.fasta*
- *./sfoverlap -e 20 -t 40 -s 2 -skip 1000 -nreads 5000 input.fasta*
- *./sfoverlap -h*

6.4 Filtrování výstupu

Program během svého běhu vypisuje na standardní výstup veškeré detekované překryvy, které splňují práh překrytí a maximální povolenou chybu. Výstup tedy obsahuje množství redundantních dat, jelikož důležité jsou pouze nejdelší detekované překryvy DNA sekvencí. Tudíž je vhodné daný výstup filtrovat. Ukázka části výstupu je zobrazena na obrázku 6.2.

Toto filtrování je možné provést tak, aby obsahoval pouze nejdelší přibližné překryvy pro každý pár. K tomuto účelu slouží nástroj *maxoverlaps*. Ukázka vyfiltrovaného výstupu při detekci překryvů 3 vstupních readů o délce 75 nukleotidů je znázorněna na obrázku 6.3.

Zdrojové soubory

- **maxoverlaps.cpp** - nástroj pro filtraci nejdelších překryvů

Příklady spuštění

- *./sfoverlap -e 20 -t 40 -s 2 input.fasta | ./maxoverlaps*
- *./maxoverlaps < vystup*

```
0	2	N	32	32	43	43	0
0	2	N	31	31	44	44	0
0	2	N	30	30	45	45	0
0	2	N	29	29	46	46	0
0	2	N	28	28	47	47	0
0	2	N	27	27	48	48	0
0	2	N	26	26	49	49	0
0	1	I	-18	-18	57	57	3
0	1	I	-20	-20	55	55	3
0	1	I	-20	-19	56	55	3
0	1	I	-18	-18	57	57	3
0	1	I	-20	-20	55	55	3
0	1	I	-20	-19	56	55	3
0	1	I	-18	-18	57	57	3
1	2	I	-33	-33	42	42	0
1	2	I	-34	-34	41	41	0
1	2	I	-32	-32	43	43	0
1	2	I	-31	-31	44	44	0
1	2	I	-30	-30	45	45	0
1	2	I	-29	-29	46	46	0
1	2	I	-28	-28	47	47	0
1	2	I	-27	-27	48	48	0
1	2	I	-26	-26	49	49	0
1	2	I	-25	-25	50	50	0
1	2	I	-24	-24	51	51	0
1	2	I	-23	-23	52	52	1
1	2	I	-20	-20	55	55	2
1	2	I	-20	-20	55	55	2
1	2	I	-20	-20	55	55	2
1	2	I	-19	-19	56	56	3
1	2	I	-29	-29	46	46	0
```

Obrázek 6.2: Příklad výpisu překryvů v průběhu detekce bez závěrečného filtrování.

```
0	2	N	6	6	69	69	0
0	1	I	-18	-18	57	57	3
1	2	I	-24	-24	51	51	0
0	2	N	-3	0	75	72	4
0	2	N	-5	-1	74	70	4
```

Obrázek 6.3: Výstup programu po filtrování nástrojem maxoverlaps.

Kapitola 7

Testování a experimenty

Navržený a implementovaný nástroj pro detekci přibližných překryvů DNA sekvencí bylo zapotřebí otestovat na reálných datech. Testování se zaměřilo především na sledování doby potřebné k nalezení překryvů, ověření očekávané teoretické lineární časové složitosti a porovnání dosažené časové složitosti s původní variantou tohoto algoritmu. Testování bylo prováděno na serverech Metacentra. Nástroji pro detekci překryvů bylo poskytnuto vždy 8 procesorů a 8 GB operační paměti. K testování byly použity dvě sady reálných NGS dat získaných nástrojem Illumina. Podrobnější popis testovacích dat se nachází v podkapitole 7.1. Naměřené doby běhů jednotlivých experimentů jsou poté uvedeny v podkapitole 7.2, zhodnocení a závěry experimentů se nachází v podkapitole 7.3.

Kromě těchto testů byly s nástrojem provedeny i jednoduší testy za účelem ověření výstupu nástroje. K tomuto účelu sloužily naopak velmi malé sety vstupních data, na niž bylo možné ověřit detekované překryvy. Výstup byl vždy také porovnán s výstupem původní varianty algoritmu.

7.1 Testovací data

Původní datasety, na kterých bylo prováděno testování v článku [11] nebyly dostupné. Z tohoto důvodu bylo testování prováděno na readech z dvou rostlin. První z nich je *Silene latifolia* nebo-li knotovka bíla, též přezdívaná jako silenka širokolistá. Jedná se o ready ze samečka této rostliny. Obsah repeticí se u samečka pohybuje okolo 61.4%, u samičky je toto číslo o něco vyšší 63.3% [14]. Pokrytí genomu těmito ready je nízké, pouze přibližně 2%. Tato sada je tvořena 295 978 ready o délce 200 nukleotidů. Druhou sadu readů tvořily ready z rostliny *Hippophae rhamnoides* nebo-li rakytník řešetlákový, též přezdívaný rakytník úzkolistý. V tomto případě se jedná o ready samičky. Pokrytí genomu těmito ready je přibližně 30%. Obsah repeticí této rostliny je u samečka menší než-li u samičky. U samečka se pohybuje okolo 52.75% a u samičky okolo 47.94% [18]. Tato sada obsahuje 831 300 readů o délce 230 nukleotidů. Obě sady těchto readů byly nasekvenovány platfourmou Illumina MiSeq. Sady byly poskytnuty Biofyzikálním ústavem Akademie věd České republiky.

Testovací sady readů

- **silene_latifolia_BYS-ma.fasta** - 295 978 readů o délce 200 nukleotidů, 61.4% repeticí, pokrytí genomu - 2%

- **HRfe_230_20_831300_15_short.fasta** - 831 300 readů o délce 230 nukleotidů, 47.94% repeticí, pokrytí genomu - 30%

Tyto testovací sady byly dále pomocí skriptu napsaného v jazyce python rozděleny na testovací datasety. Rozdělení bylo provedeno následujícím způsobem. Každá testovací sada byla rozdělena na 5 částí. Přičemž první část obsahovala 1 pětinu readů, druhá část 2 pětiny readů atd., až 5 část obsahovala všechny ready z dané testovací sady. Z každého datového setu byly následně nástrojem builder vytvořeny soubory obsahující FM-indexy těchto setů. S těmito soubory bylo poté prováděno testování.

Testovací datasety

- **silene_latifolia_BYS-ma_set1.fasta** - 59 195 readů, FM-index: 21 798 kB
- **silene_latifolia_BYS-ma_set2.fasta** - 118 390 readů, FM-index: 43 782 kB
- **silene_latifolia_BYS-ma_set3.fasta** - 177 585 readů, FM-index: 65 954 kB
- **silene_latifolia_BYS-ma_set4.fasta** - 236 780 readů, FM-index: 87 939 kB
- **silene_latifolia_BYS-ma_set5.fasta** - 295 978 readů, FM-index: 110 393 kB
- **HRfe_230_20_831300_15_short_set1.fasta** - 166 260 readů, FM-index: 69 198 kB
- **HRfe_230_20_831300_15_short_set2.fasta** - 332 520 readů, FM-index: 139 009 kB
- **HRfe_230_20_831300_15_short_set3.fasta** - 498 780 readů, FM-index: 208 514 kB
- **HRfe_230_20_831300_15_short_set4.fasta** - 665 040 readů, FM-index: 279 230 kB
- **HRfe_230_20_831300_15_short_set5.fasta** - 831 300 readů, FM-index: 349 038 kB

7.2 Získané data

Celkem bylo provedeno 100 testů, 50 testů s novým schématem a 50 odpovídajících testů s původním schématem. Při každém testu byly pomocí knihovny *ctime* měřeny doby běhu, jednotlivých testů. Získané časy byly zaznamenány do následujících tabulek. Testy byly spouštěny s časovým limitem 125 hodin. První sada testů se zaměřila na vliv prahu maximální povolené chyby překrytí na dobu běhu. Práh minimální délky překrytí byl nastaven u všech běhů na 40. Postupně byl měněn práh maximální povolené chyby překrytí. Nejprve byl parametr e nastaven na hodnotu 0.025 (tabulky 7.1 a 7.2), 0.05 (tabulky 7.3 a 7.4) a 0.1 (tabulky 7.5 a 7.6). V druhé části testování byly pak spuštěny testy s konstantní hodnotou maximální povolené chyby překrytí (e = 0,05). Naopak byl postupně měněn práh minimální délky překrytí a to na 20 (tabulky 7.7 a 7.8), 40 (tabulky 7.3 a 7.4) a 60 (tabulky 7.9 a 7.10). V případě testů s e = 0.1, tedy až 10% povolenou chybou, byl v případě velkých datasetů překročen maximální časový limit 125 hodin. Z tohoto důvodu jsou v tabulkách 7.5, 7.6, 7.7 a 7.8 uvedeny časy pouze pro některé datasety.

Silene latifolia e = 0.025, t = 40 s = 2	Počet readů	Nové schéma		Původní schéma	
		Celkový čas (h)	Čas na 1 read (s)	Celkový čas (h)	Čas na 1 read (s)
Set 1	59 195	0.96	0.059	1.21	0.074
Set 2	118 390	2.81	0.086	3.32	0.101
Set 3	177 585	4.48	0.091	6.51	0.132
Set 4	236 780	7.42	0.113	10.13	0.154
Set 5	295 978	11.01	0.134	14.71	0.179

Tabulka 7.1: Časy běhů s datovou sadou Silene latifolia a s parametry: e= 0.025, t = 40, s = 2.

Hip. rham. e = 0.025, t = 40 s = 2	Počet readů	Nové schéma		Původní schéma	
		Celkový čas (h)	Čas na 1 read (s)	Celkový čas (h)	Čas na 1 read (s)
Set 1	166 260	2.92	0.063	4.29	0.093
Set 2	332 520	11.15	0.121	12.84	0.139
Set 3	498 780	20.36	0.147	24.25	0.175
Set 4	665 040	27.90	0.151	36.76	0.199
Set 5	831 300	42.25	0.183	50.34	0.218

Tabulka 7.2: Časy běhů s datovou sadou Hippophae rhamnoides a s parametry: e= 0.025, t = 40, s = 2.

Silene latifolia e = 0.05, t = 40 s = 2	Počet readů	Nové schéma		Původní schéma	
		Celkový čas (h)	Čas na 1 read (s)	Celkový čas (h)	Čas na 1 read (s)
Set 1	59 195	2.74	0.166 8	3.06	0.186
Set 2	118 390	7.39	0.225	8.49	0.268
Set 3	177 585	12.89	0.261	13.24	0.272
Set 4	236 780	19.99	0.304	22.11	0.303
Set 5	295 978	29.50	0.359	34.51	0.424

Tabulka 7.3: Časy běhů s datovou sadou Silene latifolia a s parametry: e = 0.05, t = 40, s = 2.

Hip. rham. e = 0.05, t = 40 s = 2	Počet readů	Nové schéma		Původní schéma	
		Celkový čas (h)	Čas na 1 read (s)	Celkový čas (h)	Čas na 1 read (s)
Set 1	166 260	4.95	0.107	6.14	0.133
Set 2	332 520	16.71	0.181	21.31	0.230
Set 3	498 780	32.15	0.232	40.57	0.293
Set 4	665 040	55.67	0.301	67.26	0.364
Set 5	831 300	84.28	0.365	98.02	0.425

Tabulka 7.4: Časy běhů s datovou sadou Hippophae rhamnoides a s parametry: e = 0.05, t = 40, s = 2.

Silene latifolia e = 0.1, t = 40 s = 2	Počet readů	Nové schéma		Původní schéma	
		Celkový čas (h)	Čas na 1 read (s)	Celkový čas (h)	Čas na 1 read (s)
Set 1	59 195	38.69	2.353	47.02	2.886
Set 2	118 390	109.01	3.315	123.55	3.757

Tabulka 7.5: Časy běhů s datovou sadou Silene latifolia a s parametry: e = 0.1, t = 40, s = 2. Běhy, jenž překročily časový limit 125 hodin, nejsou v tabulce uvedeny.

Hip. rham. e = 0.1, t = 40 s = 2	Počet readů	Nové schéma		Původní schéma	
		Celkový čas (h)	Čas na 1 read (s)	Celkový čas (h)	Čas na 1 read (s)
Set 1	166 260	24.79	0.537	53.89	1.167
Set 2	332 520	92.736	1.006	119,98	1,299

Tabulka 7.6: Časy běhů s datovou sadou Hippophae rhamnoides a s parametry: e= 0.1, t = 40, s = 2. Běhy, jenž překročily časový limit 125 hodin, nejsou v tabulce uvedeny.

Silene latifolia e = 0.05, t = 20 s = 2	Počet readů	Nové schéma		Původní schéma	
		Celkový čas (h)	Čas na 1 read (s)	Celkový čas (h)	Čas na 1 read (s)
Set 1	59 195	54.62	3.322	61.49	3.740

Tabulka 7.7: Časy běhů s datovou sadou Silene latifolia a s parametry: e= 0.05, t = 20, s = 2. Běhy, jenž překročily časový limit 125, hodin nejsou v tabulce uvedeny.

Hip. rham. e = 0.05, t = 20 s = 2	Počet readů	Nové schéma		Původní schéma	
		Celkový čas (h)	Čas na 1 read (s)	Celkový čas (h)	Čas na 1 read (s)
Set 1	166 260	45.53	0.986	54,68	1.184
Set 2	332 520	107.42	1.163	118.59	1.284

Tabulka 7.8: Časy běhů s datovou sadou Hippophae rhamnoides a s parametry: e= 0.05, t = 20, s = 2. Běhy, jenž překročily časový limit 125 hodin, nejsou v tabulce uvedeny.

Silene latifolia e = 0.1, t = 60 s = 2	Počet readů	Nové schéma		Původní schéma	
		Celkový čas (h)	Čas na 1 read (s)	Celkový čas (h)	Čas na 1 read (s)
Set 1	59 195	1.29	0.078	1.62	0.099
Set 2	118 390	3.38	0.103	4.21	0.128
Set 3	177 585	6.39	0.130	7.15	0.145
Set 4	236 780	11.44	0.174	12.89	0.196
Set 5	295 978	15.04	0.183	16,52	0.201

Tabulka 7.9: Časy běhů s datovou sadou Silene latifolia a s parametry: e= 0.1, t = 60, s = 2.

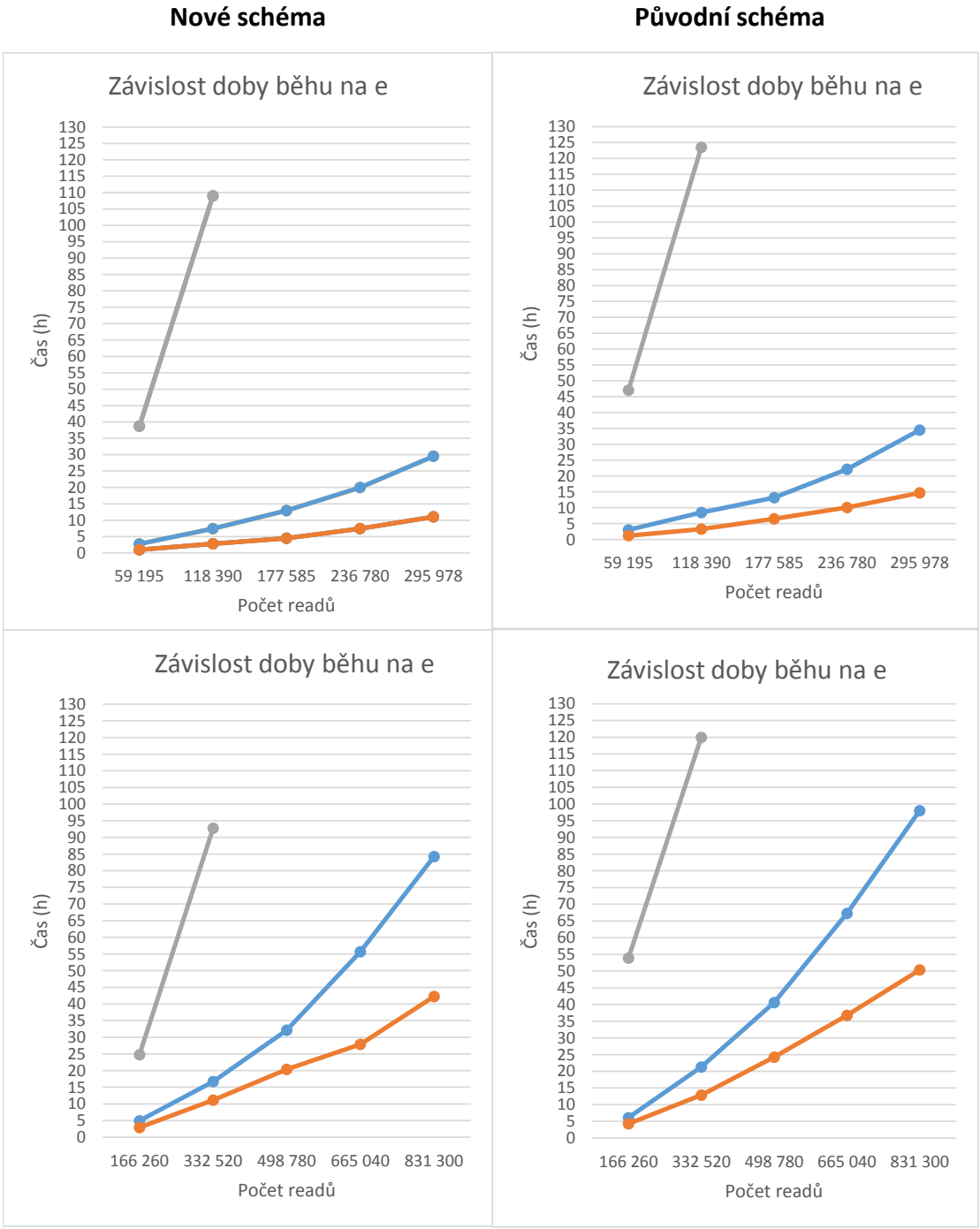
Hip. rham. e = 0.1, t = 60 s = 2	Počet readů	Nové schéma		Původní schéma	
		Celkový čas (h)	Čas na 1 read (s)	Celkový čas (h)	Čas na 1 read (s)
Set 1	166 260	3.17	0.069	3.28	0.071
Set 2	332 520	8.682	0.094	10.71	0.116
Set 3	498 780	16.90	0.122	22.45	0.162
Set 4	665 040	30.85	0.167	33.80	0.183
Set 5	831 300	40.87	0.177	48.88	0.203

Tabulka 7.10: Časy běhů s datovou sadou Hippophae rhamnoides a s parametry: e= 0.1, t = 60, s = 2.

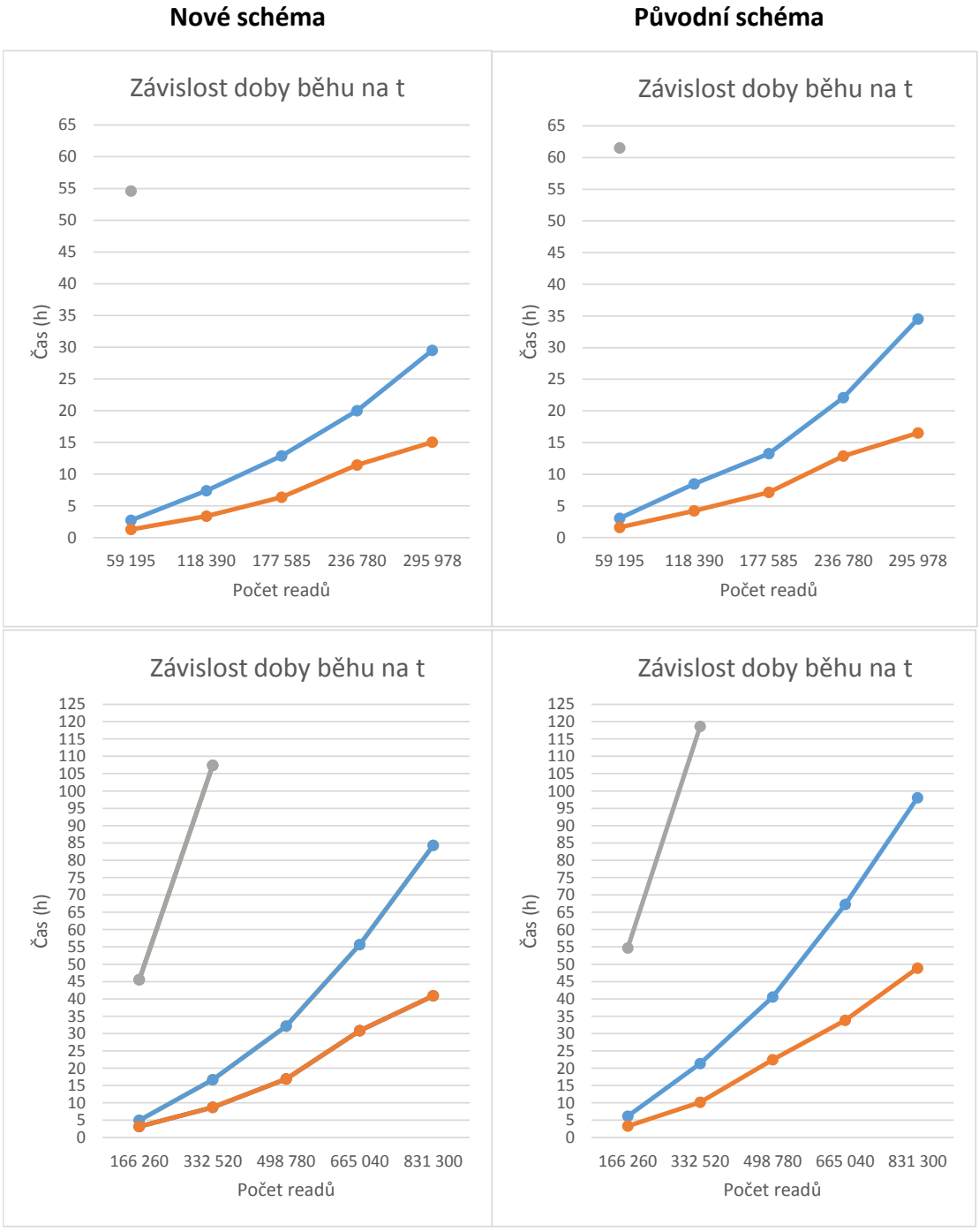
7.3 Závěry a shrnutí experimentů

Pro určení časové složitosti daného algoritmu byly tabelované hodnoty z podkapitoly 7.2 vyneseny do grafů (obrázky 7.1 a 7.2). Z grafů je patrné, že časová složitost algoritmu pro detekci překryvů (s novým i s původním schématem) se ve většině případů více blíží lineární časové složitosti. Ovšem nikdy se nejedná přímo o lineární časovou složitost. Časová složitost totiž závisí nejen na velikosti vstupní množiny, ale také na thresholdu a na počtu detekovaných překryvů. Také má na časovou složitost vliv dodatečná režie, která je nutná pro načtení FM-indexů a následné využívání této datové struktury. Nejlepších výsledků algoritmus dle očekávání dosahuje s nižšími hodnotami prahu maximální povolené chyby a s vyššími hodnotami prahu minimální délky překrytí. Toto je způsobeno nižším počtem akceptovaných překryvů a tedy i značně nižším množstvím potenciálních kandidátů na shodu. Je zřejmé tedy, že časová složitost závisí i na počtu detekovaných překryvů. Opačný efekt poté má volba vyšších hodnot prahu maximální povolené chyby a nižších hodnot prahu minimální délky překrytí. V této situaci se v případě většího datasetu blíží časová složitost složitosti kvadratické. Zde nastává opačná situace, je akceptováno daleko větší množství překryvů a algoritmus tedy generuje značně větší množství kandidátu na shodu, kteří jsou následně kontrolování. Rozdíl časů mezi podobně velkými datasety různých datových sad je pravděpodobně způsoben rozdílným množstvím detekovaných překryvů u daných datasetů, na což má také vliv různý podíl repeticí.

V porovnání nového schématu s původním dosahuje nové schéma lepších výsledků. Tento rozdíl mezi časy se zvyšuje s množstvím zpracovávaných readů a tedy i s rostoucí celkovou délkou běhu nástroje, kdy tento rozdíl se může pohybovat až v řádech desítek hodin. Tyto poznatky získané z experimentů naplnily, jak očekávání plynoucí z teoretické analýzy algoritmu, tak i z experimentů z článku [11], které se zaměřovaly na ověření nižšího počtu generovaných kandidátů. Využití sufixových filtrů pro detekci překryvů DNA sekvencí dosahuje lepší časové složitosti než-li naivní přístup využívaný nástrojem RepeatExplorer, jenž pracuje s kvadratickou časovou složitostí.



Obrázek 7.1: Grafy zobrazující časovou složitost pro různé hodnoty parametru e s datovou sadou *Silene latifolia* (horní dva grafy) a datovou sadou *Hippophae rhamnoides* (spodní dva grafy). Oranžová barva znázorňuje průběh pro e = 0.025, modrá barva průběh pro e = 0.05 a šedá barva pro 0.1.



Obrázek 7.2: Grafy zobrazující časovou složitost pro různé hodnoty parametru t s datovou sadou *Silene latifolia* (horní dva grafy) a datovou sadou *Hippophae rhamnoides* (spodní dva grafy). Oranžová barva znázorňuje průběh pro t = 60, modrá barva průběh pro e = 40 a šedá barva pro 20.

Kapitola 8

Závěr

V rámci diplomové práce byly nejprve nastudovány základy molekulární biologie, především informace o DNA, genomu, genech a repetivních elementech v DNA. Jelikož cílem této práce je detekovat překryvy DNA sekvencí získaných pomocí sekvenování NGS metodami, tak byl vytvořen přehled sekvenovacích nástrojů se zaměřením na NGS metody. Byly také zhodnoceny výhody a nevýhody jednotlivých metod. Dále byl obecně popsán problém detekce překryvů DNA sekvencí. Také byly nastudovány existující algoritmy a přístupy, které slouží k hledání překryvů DNA sekvencí. Z počátku se práce zaměřila na nástroje, které slouží k sestavování genomu a jsou založeny na principu detekce překryvů DNA sekvencí. Jednotlivé algoritmy, které jsou využívány těmito nástroji, byly popsány a byla určena jejich časová a prostorová složitost a jejich výhody a nevýhody. Tato složitost byla většinou lineární. Ovšem u některých metod v důsledku nedostatečných informací nebylo možné tuto složitost stanovit.

Jelikož při sestavování genomu je nejprve prováděna fáze filtrování a tedy jsou odstraněny ready obsahující chyby vzniklé v důsledku sekvenování a repetitivních elemntů DNA, tak nejsou tyto algoritmy vhodné pro možné úpravy. Tudíž se práce zaměřila na další možné algoritmy a přístupy vhodné pro detekci přibližných překryvů DNA sekvencí. Jako nejvhodnější algoritmus byly zvoleny sufixové filtry, jelikož umožňují pracovat s relativním prahem maximální povolené chyby závislým na délce překrytí. Tyto sufixové filtry byly nejprve popsány teoreticky, následně byl navržen algoritmus využívající sufixové filtry s novým filtračním a dělícím schématem, jež byly teoreticky popsány v článku [11]. Následně byl tento algoritmus s využitím existující implementace sufixových filtrů implementován.

V závěru práce bylo provedeno nejprve testování se zaměřením na ověření výstupu algoritmu. Následně byla provedena řada experimentů s reálnými NGS daty získanými sekvenovací platformou Illumina. Přesněji se jednalo o dvě sady readů z genomu dvou rostlin: *Silene latifolia* a *Hippophae rhamnoides*. Tyto experimenty se zaměřovaly na ověření časové složitosti algoritmu a porovnání rychlosti nového schématu s původním. Časová složitost se ve většině případů blížila lineární časové složitosti. Odchylka od této časové složitosti je nejpravděpodobněji způsobena režií nutnou pro práci s FM-indexem, na němž jsou sufixové filtry postaveny. Nové schéma poté dosahovalo o něco lepších výsledků nežli původní schéma.

Možným pokračováním projektu je integrace nástroje pro detekci přibližných překryvů DNA sekvencí do nástroje RepeatExplorer. Dalším pokračováním by pak mohlo být důslednější a důkladnější otestování na větším množství dat s různým zastoupením repeticí.

Literatura

[1] Difference DNA RNA-EN. březen 2013, [Online; navštíveno 22.12.2016].
URL https://en.wikipedia.org/wiki/File:Difference_DNA_RNA-EN.svg

[2] Gen. prosinec 2016, [Online; navštíveno 15.5.2017].
URL https://cs.wikipedia.org/wiki/Gen#/media/File:Gene.png

[3] DeWeerdt, S. E.: What's a Genome? Leden 2013.
URL
http://www.genomenewsnetwork.org/resources/whats_a_genome/Chp1_1_1.shtml

[4] DiGuistini, S.; Liao, N. Y.; Platt, D.; aj.: De novo genome sequence assembly of a filamentous fungus using Sanger, 454 and Illumina sequence data. *Genome biology*, ročník 10, č. 9, 2009: s. R94.1–R94.12.

[5] van Dijk, E. L.; Auger, H.; Jaszczyszyn, Y.; aj.: Ten years of next-generation sequencing technology. *Trends in genetics*, ročník 30, č. 9, sep 2014: s. 418–426.

[6] El-Metwally, S.; Hamza, T.; Zakaria, M.; aj.: Next-generation sequence assembly: four stages of data processing and computational challenges. *PLoS Comput Biol*, ročník 9, č. 12, 2013: str. e1003345.

[7] Gonnella, G.; Kurtz, S.: Readjoiner: a fast and memory efficient string graph-based sequence assembler. *BMC bioinformatics*, ročník 13, č. 1, 2012: str. 19.

[8] Kärkkäinen, J.; Na, J. C.: Faster filters for approximate string matching. In *2007 Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments (ALENEX)*, SIAM, 2007, s. 84–90.

[9] Kejnovsky, E.; Hawkins, J. S.; Feschotte, C.: Plant transposable elements: biology and evolution. In *Plant Genome Diversity Volume 1*, Springer, 2012, s. 17–34.

[10] Kidwell, M. G.: Transposable elements and the evolution of genome size in eukaryotes. *Genetica*, ročník 115, č. 1, 2002: s. 49–63, ISSN 0016-6707, doi:10.1023/A:1016072014259.

[11] Kucherov, G.; Tsur, D.: Improved filters for the approximate suffix-prefix overlap problem. In *International Symposium on String Processing and Information Retrieval*, Springer, 2014, s. 139–148.

[12] Li, H.: Exploring single-sample SNP and INDEL calling with whole-genome de novo assembly. *Bioinformatics*, ročník 28, č. 14, 2012: s. 1838–1844.

[13] Liu, L.; Li, Y.; Li, S.; aj.: Comparison of next-generation sequencing systems. *BioMed Research International*, ročník 2012, 2012: str. 11.

[14] Macas, J.; Kejnovskỳ, E.; Neumann, P.; aj.: Next generation sequencing-based analysis of repetitive DNA in the model dioceous plant Silene latifolia. *PLoS One*, ročník 6, č. 11, 2011: str. e27335.

[15] Miller, J. R.; Delcher, A. L.; Koren, S.; aj.: Aggressive assembly of pyrosequencing reads with mates. *Bioinformatics*, ročník 24, č. 24, 2008: s. 2818–2824.

[16] Muñoz-López, M.; García-Pérez, J. L.: DNA transposons: nature and applications in genomics. *Current genomics*, ročník 11, č. 2, 2010: s. 115–128.

[17] Novák, P.; Neumann, P.; Macas, J.: Graph-based clustering and characterization of repetitive sequences in next-generation sequencing data. *BMC bioinformatics*, ročník 11, č. 1, 2010: str. 12.

[18] Puterova, J.; Razumova, O.; Martinek, T.; aj.: Satellite DNA and Transposable Elements in Seabuckthorn (Hippophae rhamnoides), a Dioecious Plant with Small Y and Large X Chromosomes. *Genome Biology and Evolution*, 2017: str. evw303.

[19] Rasmussen, K. R.; Stoye, J.; Myers, E. W.: Efficient q-gram filters for finding all ε-matches over a given length. *Journal of Computational Biology*, ročník 13, č. 2, 2006: s. 296–308.

[20] Simpson, J. T.; Durbin, R.: Efficient construction of an assembly string graph using the FM-index. *Bioinformatics*, ročník 26, č. 12, 2010: s. i367–i373.

[21] Välimäki, N.; Ladra, S.; Mäkinen, V.: Approximate all-pairs suffix/prefix overlaps. In *Annual Symposium on Combinatorial Pattern Matching*, Springer, 2010, s. 76–87.

[22] Wessler, S. R.: Transposable elements and the evolution of eukaryotic genomes. *Proceedings of the National Academy of Sciences of the United States of America*, ročník 103, Listopad 2006: str. 17600–17601, ISSN 0027-8424, doi:10.1073/pnas.0607612103.

Příloha A

Obsah CD

- **Spustitelné soubory** - obsahuje spustitelné soubory k nástroji pro detekci překryvů.
- **Zdrojové soubory** - obsahuje veškeré zdrojové soubory včetně souboru *makefile*.
- **Testovací data** - obsahuje obě testovací sady, na kterých bylo provedeno testování.
- **README** - soubor s popisem překladu a použití nástroje pro detekci překryvů.
- **doc** - obsahuje text práce a zdrojové soubory tohoto textu včetně obrázků.
- **skript.py** - skript pro rozdělení testovacích sad na datasety.