Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Využitie reťazcových grafov pri hľadaní chýb v genómoch

Bakalárska práca

2020
Angelika Fedáková

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Využitie reťazcových grafov pri hľadaní chýb v genómoch

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra informatiky
Školiteľ:	doc. Mgr. Bronislava Brejová, PhD.

Bratislava, 2020
Angelika Fedáková

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Angelika Fedáková
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Využitie reťazcových grafov pri hľadaní chýb v genómoch
Use of string graphs in genome assembly error detection

Anotácia: Reťazcový graf sa využíva v bioinformatike na uloženie neúplnej informácie o študovanom reťazci alebo množine reťazcov. Sekvenovaním DNA získame čítania, ktoré sú reťazcovou reprezentáciou časti skúmaného genómu. Skúmané genómy sa potom zostavujú z takýchto čítaní výpočtovými metódami. Cieľom práce je porovnať dlhé sekvenačné čítania so zostaveným genómom prostredníctvom reťazcových grafov, a tým v zostavenom genóme detegovať chyby. Práca bude kombinovať existujúce nástroje a vlastný kód a testovať vyvinuté metódy na reálnych biologických dátach.

Vedúci: doc. Mgr. Bronislava Brejová, PhD.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: prof. RNDr. Martin Škoviera, PhD.

Dátum zadania: 23.10.2019

Dátum schválenia: 24.10.2019 doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie: Chcem sa poďakovať svojej školiteľke doc. Mgr. Bronislave Brejovej, PhD., za cenné rady a odbornú pomoc, ktoré mi poskytla pri realizácii bakalárskej práce. Zároveň sa chcem poďakovať všetkým, ktorí ma počas písania práce podporovali.

Abstrakt

Sekvencie DNA zostavované dostupnými bioinformatickými nástrojmi obsahujú veľké množstvo chýb. Cieľom našej práce bolo preskúmať možnosť využitia dlhých čítaní a reťazcového grafu na hľadanie chýb v zostavených sekvenciách. Po detailnejšom návrhu myšlienky nášho algoritmu sme vytvorili softvér, ktorý hľadá chyby pomocou zarovnania skúmanej sekvencie a dlhých čítaní ku kompaktovanému de Bruijnovmu grafu. Po využití nášho softvéru na reálnych biologických dátach sme nájdené chyby ďalej skúmali, pričom sme navrhli metódu ohodnotenia presnosti a úplnosti našich výsledkov. Našli sme aj ďalšie možnosti filtrovania chýb, ktoré zvýšili kvalitu našich výsledkov. Na záver sme porovnali naše výsledky s existujúcim softvérom na opravu chýb a navrhli sme ďalšie možnosti vylepšenia a rozšírenia nášho softvéru.

Kľúčové slová: dlhé čítania, de Bruijnov graf, hľadanie chýb

Abstract

DNA sequences assembled by accessible bioinformatics tools contain high amount of errors. The aim of our work was to examine the possibility of using long reads and string graphs for finding the errors in assembled sequences. After the detailed analysis of our proposed method, we created a software which is designed to search the errors based on the alignments of assembled sequence and long reads to compacted de Bruijn graph. After the application of our software on real biological data we further examined the errors and suggested a method for the evaluation of precision and recall of our results. We also found other ways to filter the errors which improved the quality of the outcomes. In the end we compared our results with the already existing software for finding the errors and proposed further improvements and extensions of our software.

Keywords: long reads, de Bruijn graph, error searching

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Skúmanie DNA živých organizmov je v dnešnej dobe aktívnou oblasťou výskumu. Znalosť informácie zakódovanej v DNA nám pomáha v rôznych oblastiach, ako napríklad pri diagnostike chorôb v medicíne. Poradie báz v molekule DNA nazývame sekvencia a postup získania sekvencie DNA nazývame sekvenovanie. Keďže tento proces nedokáže poskytnúť celý genóm v kuse, vzniknú nám viaceré časti, ktoré nazývame čítania.. Proces skladania čítaní do čo najväčších kusov sa nazýva zostavovanie genómu. Pri tomto procese vznikajú rôzne chyby, ktoré znižujú podobnosť zostaveného genómu reálnej postupnosti báz vo vzorke.

Na hľadanie a opravu chýb v zostavených genómoch existuje v dnešnej dobe viacero softvérových nástrojov [13, 34, 17]. Väčšina z nich spätne zarovnáva sekvenačné čítania k zostavenej sekvencii. Následne v týchto zarovnaniach hľadajú javy, ktoré by sa pri správne zostavených sekvenciách nemali vyskytovať. Cieľom našej práce bolo priniesť nový pohľad na túto problematiku a hľadať chyby pomocou dlhých čítaní a reťazcového grafu. Reťazcový graf nám pomáha združovať informácie o zdieľaných aj rozdielnych častiach genómu. Kým existujúce prístupy väčšinou využívajú dáta z technológií produkujúcich krátke a veľmi presné čítania, my využívame dlhé čítania, ktoré nám prinášajú informácie o rozložení rozsiahlejších častí genómu. Skombinovaním grafov a dlhých čítaní chceme určovať rozsiahlejšie chyby zostaveného genómu.

V kapitole 1 popíšeme základné biologické pojmy a postupy potrebné pre pochopenie tejto práce. V kapitole 2 detailnejšie popíšeme problematiku hľadania chýb, ktorou sa naša práca zaoberá. Zároveň popíšeme niekoľko existujúcich softvérov a základnú myšlienku nášho algoritmu. V kapitole 3 presne popíšeme vstupy, jednotlivé kroky, formáty výstupov nášho algoritmu, ako aj jazyky použité pri implementácii. V kapitole 4 otestujeme navrhnutý postup na reálnych biologických dátach. Taktiež popíšeme získané výsledky a ich interpretáciu.

Kapitola 1

Úvod do problematiky

V tejto kapitole zavedieme základné biologické a informatické pojmy, potrebné pre pochopenie tejto práce. Hlavná oblasť, ktorej sa týka naša práca, je hľadanie, resp. opravovanie chýb v zostavovaných genómoch. Preto potrebujeme vedomosti ohľadom sekvenovania a samotného zostavovania genómu ako jedného zo základných problémov bioinformatiky. Spomenieme vybrané algoritmy a technológie, riešiace tento problém, ako aj súvisiaci problém zarovnávania sekvencií.

1.1 DNA, chromozóm, genóm

Deoxyribonukleová kyselina (DNA) je dedičný materiál u ľudí a takmer všetkých živých organizmov, ktorý sa nachádza v každej bunke.

Základnou stavebnou zložkou DNA je nukleotid (taktiež nazývaný báza). Poznáme 4 základné typy DNA nukleotidov, a to: adenín (A), cytozín (C), guanín (G) a tymín (T) [32]. DNA je zložená z dvoch vlákien, každé pozostávajúce z reťazca nukleotidov, ktoré spolu tvoria dvojzávitnicovú špirálu. Jednotlivé nukleotidy v opačných vláknach sa vždy spájajú podľa Watson-Crick-ovho pravidla. To hovorí, že nukleotid adenín sa bude vždy spájať iba s tymínom a cytozín výhradne s guanínom. Tieto dvojice nazývame bázové páry (bp). Dve vlákna tvoriace DNA sú teda komplementárne. To organizmu umožňuje pri replikácii alebo chybe nejakej časti DNA danú časť doplniť, resp. nahradiť podľa komplementárneho vlákna. Dva konce DNA vlákna označujeme ako $5'$ a $3'$ koniec. Komplementárne vlákna sú opačne orientované, čo znamená, že $5'$ koniec sa vždy páruje s $3'$ koncom opačného vlákna a naopak. DNA zvyčajne reprezentujeme reťazcom znakov z množiny $\{A, C, G, T\}$. Stačí si pamätať jedno z DNA vlákien, keďže druhé vieme dopočítať. Keď z vlákna vytvoríme reverzný reťazec a bázy zameníme podľa komplementarity, získame komplementárne vlákno.

Genóm je genetický materiál žijúceho organizmu. Obsahuje dedičné inštrukcie pre stavbu, beh a udržiavanie organizmu a dedenie pre ďalšie generácie [14]. Genómy sú

špecifické pre jednotlivé druhy, avšak genómy každého jednotlivca daného druhu sú taktiež v menšej miere odlišné. Genóm je z informatického hľadiska množina chromozómov.

Chromozóm je jedna časť dvojvláknovej DNA v jadre bunky. Každý chromozóm je reťazec znakov so začiatkom a koncom. V prípade niektorých baktérií môžu byť chromozómy kruhové, a preto nemajú začiatok ani koniec.

1.2 Sekvenovanie genómu

DNA sekvenovanie je proces, pri ktorom sa z biologickej vzorky určuje poradie nukleotidových báz DNA. Vďaka nemu DNA špecifického organizmu vieme zapísať do formátu, s ktorým následne dokážu výskumníci a vedci ďalej pracovať pomocou rôzneho softvéru [32]. Vstupom je biologický materiál a ideálnym výstupom je pre každý chromozóm genómu celý reťazec DNA. Momentálne technológie nedokážu prečítať postupnosť celého chromozómu naraz, preto je potrebné chromozóm nasekať na menšie časti a tie následne osekvenovať. Tieto jednotlivé časti, podreťazce DNA chromozómu, nazývame čítania. Výsledkom procesu sekvenovania je multimnožina čítaní z genómu vzorky. Pri sekvenovaní nastáva viacero problémov, ktoré spôsobujú nepresnosti vo výsledkoch sekvenovania. Môže to byť nesprávne prečítaná konkrétna báza, nerovnomerné pokrytie genómu čítaniami, či kontaminácia inou DNA. Pre získanie celého genómu je následne potrebné bioinformatickými metódami čítania poskladať do pôvodných reťazcov chromozómov. Pre sekvenovanie využívame rôzne technológie, z ktorých každá má svoje výhody aj nevýhody. V tejto práci budeme používať dáta z dvoch technológií, ktoré teraz bližšie predstavíme.

1.2.1 Sekvenovanie Illumina

V počiatkoch sekvenovania prevládalo Sangerovo sekvenovanie, ktoré sa dnes nazýva aj sekvenovanie prvej generácie. Toto sekvenovanie vyžadovalo rozsiahle prípravy DNA vzorky a veľa ľudskej práce. Čas potrebný na sekvenovanie bol vysoký a to spôsobovalo aj vysokú cenu. Dĺžka čítaní pri tomto prístupe je okolo 800bp a chybovosť menšia ako 1% [32]. Najväčšou nevýhodou je čas, ktorý toto sekvenovanie zaberá. Napríklad osekvenovanie celého ľudského genómu trvalo niekoľko rokov. Preto sa pracovalo na vývoji sekvenačných technológií a vzniklo sekvenovanie druhej generácie. Poznáme viacero rôznych technológií spadajúcich do tejto kategórie a jednou z nich je Illumina. Je jednou z prvých technológií sekvenovania druhej generácie a v našej práci ju budeme využívať ako zdroj krátkych čítaní s nízkou chybovosťou.

Pri technológií Illumina sa znížil čas potrebný na prípravu vzorky. Vzorka sa počas procesu mnohonásobne namnoží. Následne sú ku vzorke pridávané zafarbené bázy,

ktoré sa podľa komplementarity postupne viažu na bázy DNA vo vzorke. Tieto farby sú skenované a podľa nich sú následne určované bázy čítaní. Tento proces vyprodukuje obrovské množstvo čítaní paralelne. Dĺžka čítaní sa pohybuje v rozsahu 50bp až 300bp [18]. Illumina je často využívaná hlavne vďaka svojej nízkej chybovosti, ktorá sa pohybuje pod 1% [15].

1.2.2 Nanopórové sekvenovanie

Následníkom druhej generácie sekvenovania je tretia generácia sekvenovania. Ako jej zástupcu spomenieme technológiu firmy Oxford Nanopore, ktorú v tejto práci využívame ako zdroj dlhých čítaní.

Základom tejto technológie je prechod skúmaného vlákna DNA cez nanopóry v membráne, ktorými prechádza prúd. Bázy postupne prechádzajú cez tento pór a ovplyvňujú hodnotu prúdu. Táto zmena elektrického signálu je zaznamenaná a ďalej spracovaná. Vďaka tomu, že jednotlivé bázy rôzne menia hodnoty prúdu, dokážeme z týchto hodnôt vytvoriť postupnosť báz sekvencie. Výhodou tejto technológie je dĺžka čítaní. Nanopórové sekvenovanie dokáže prečítať aj dlhú molekulu ako jedno čítanie. Problém je ale pri príprave materiálu, lebo DNA vlákna sú náchylné na zlomenie. Preto je zložité vytvoriť vzorku, v ktorej by boli celé chromozómy v kuse. Výsledné čítania sú napriek tomu oproti Illumine oveľa dlhšie, dosahujú dĺžku aj 50kbp [7]. Nevýhodou týchto čítaní je ale ich chybovosť, ktorá sa pohybuje až do 12% [15]. Tieto čítania však nebudeme využívať na samotné zostavenie genómu, ale na hľadanie chýb, kde sa do istej miery znižuje potreba presnosti jednotlivých báz.

1.3 Zostavovanie genómu

V momentálnej dobe nepoznáme technológie, ktoré by dokázali osekvenovať napríklad celý ľudský chromozóm v jednom čítaní. Preto musíme genóm získať nepriamo. Proces, pri ktorom sa z kratších čítaní, ktoré vznikli sekvenovaním, určuje poradie nukleotidov celého genómu, sa nazýva zostavovanie genómu. Ak ide o zostavovanie vybranej časti genómu, hovoríme o zostavovaní sekvencie. Je to jeden zo základných problémov bioinformatiky. Vďaka náhodnému vzorkovaniu vytvorené čítania pokrývajú genóm pomerne rovnomerne. Keďže je čítaní veľké množstvo, čítania sa prekrývajú. Genóm je tak pokrytý viacnásobne [32].

Pri zostavovaní genómu sa stretáme s rôznymi problémami, pričom niektoré závisia od použitých technológií, spomínaných v kapitole 1.2.

- Lokálne chyby - už pri samotnom sekvenovaní sa môže stať, že sa niektoré bázy osekvenujú chybne (voči reálnej DNA sekvencii) a teda vznikajú chyby na úrovni báz (inzercie, delécie, substitúcie).

- Opakované čítania - niektoré kusy DNA sa môžu v genóme opakovať aj na viacerých miestach. V prípade čítaní, ktoré sa opakujú, je následne ťažké určiť, či ide o viacnásobné pokrytie rovnakej časti, alebo o časť, ktorá sa v DNA opakuje.
- Dĺžka čítania - čím sú čítania kratšie, tým je ťažšie DNA zostaviť, keďže máme menšiu informáciu o tom, ktoré vzdialené kusy k sebe patria.
- Dĺžka zostavovanej sekvencie - podobne ako pri dĺžke čítaní, čím je zostavovaná sekvencia kratšia, tým je jednoduchšie ju zostaviť.
- Rozlišná priemerná hĺbka pokrytia - niektoré časti DNA môžu byť pokryté čítaniami viac ako ostatné, dokonca niektoré časti nemusia byť pokryté vôbec, a preto sú pre nás v tomto prípade nezistiteľné.
- Kontaminácia inou DNA - v našich čítaniach sa môžu vyskytnúť aj čítania, ktoré nepochádzajú z daného genómu, ak sa do vzorky chybou spracovania primieša napríklad bakteriálna alebo ľudská DNA.
- Komplementarita - pri sekvenácii získavame dáta z oboch vlákien DNA, preto je potrebné vždy uvažovať aj nad orientáciou čítania.

Pri skladaní genómu niektoré časti nevieme jednoznačne pospájať alebo nám o nich chýba informácia. Namiesto toho, aby sme ich umelo dopĺňali nie nutne správnymi dátami, tieto časti necháme nespojené. Súvislé časti DNA, ktoré sa nám podarilo poskladať, nazývame *kontigy*. Vzniká nám takto rozsekaná finálna sekvencia pozostávajúca z jedného alebo viacerých kontigov.

Poznáme dva základné prístupy k zostavovaniu genómu: Overlap-Layout-Consensus a reťazcové grafy.

1.4 Overlap-Layout-Consensus

Prístup Overlap-Layout-Consensus je nazvaný podľa troch hlavných fáz zostavovania: overlap (nájdenie prekryvov), layout (rozostavenie), a consensus (súhlas) [30]. Vo fáze overlap hľadáme prekryvy čítaní, zisťujeme teda pozície, ktorých čítaní sa potenciálne prekrývajú v hľadanej sekvencii. Dve čítania sa prekrývajú, ak sa sufix jedného čítania rovná prefixu iného čítania. Dĺžka tohto sufixu (prefixu) je dĺžka prekryvu čítaní. V druhej fáze, layout, pospájame jednotlivé prekryvy do čo najväčších kontigov. Posledná fáza, consensus, slúži na zostavenie výslednej sekvencie. Každá báza v kontigu môže byť (a takmer vždy aj je) pokrytá viacerými čítaniami. Z báz jednotlivých čítaní na danej pozícii vyberáme jednu finálnu bázu sekvencie. Tieto vybrané bázy tvoria finálne kontigy sekvencie. Týmto výberom sa snažíme eliminovať chyby, ktoré počas sekvenovania mohli vzniknúť v jednotlivých čítaniach.

1.5 Reťazcové grafy

Reťazcové grafy sú jedným z významných prístupov k zostavovaniu genómu. Všeobecne najskôr z daných čítaní zostavíme graf (pričom tomuto kroku môže predchádzať nejaká úprava čítaní) a následne sa v grafe snažíme hľadať sledy s istými vlastnosťami, ktoré budú zodpovedať výsledným kontigom. Tieto grafy majú väčšinou čítania, resp. ich podreťazce ako vrcholy a tie sú prepojené, pokiaľ nejako súvisia (buď to sú časti čítaní, ktoré majú spoločný prekryv, alebo to sú časti rovnakého čítania). Jeden z týchto grafov je graf prekryvov a druhý prístup, ktorý budeme využívať v tejto práci, sú de Bruijnove grafy.

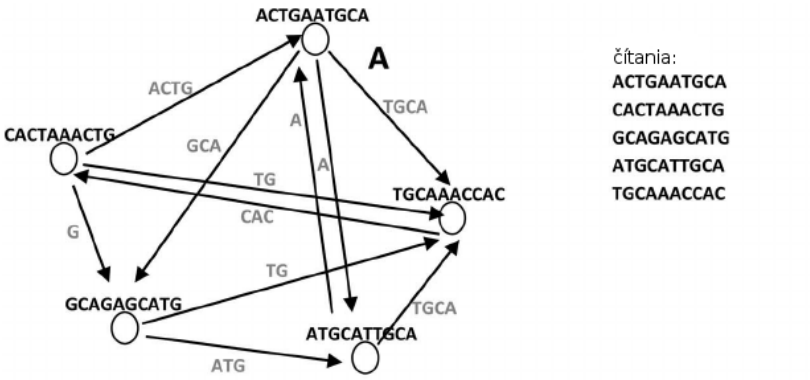
1.5.1 Graf prekryvov

Graf prekryvov a operácie nad ním patria medzi časté implementácie prvých dvoch fáz Overlap-Layout-Consensus prístupu. V najjednoduchšom grafovom modeli je každé sekvenačné čítanie vrchol grafu a pár vrcholov je spojený hranou, ak sa prekrývajú [30]. Hrana je označená číslom počtu prekrývajúcich sa báz, alebo reťazcom, ktorý tvorí prekryv (ako na obrázku 1.1). Ako reťazec sledu vrcholov označujeme zreťazenie reťazcov jednotlivých vrcholov sledu, pričom prekryvy medzi danými vrcholmi musíme do reťazca vložiť iba raz. V takomto grafe sa nám často stáva, že v ňom vznikajú tranzitívne hrany. To znamená, že nejaký reťazec sledu vrcholov hovorí o rovnakom reťazci ako iný sled s menším počtom vrcholov. Hrany, ktoré prepájajú jednu skupinu vrcholov sú voči hranám, ktoré prepájajú druhú skupinu vrcholov tranzitívne. Jednu zo skupín hrán je preto nutné odstrániť. Po odstránení tranzitívnych hrán jednoznačné sledy bez vetvenia považujeme za kontigy. Krok nájdenia prekryvov v tomto procese trvá najdlhšie. Ak by sme využili naivný prístup porovnávania všetkých čítaní navzájom, kvôli dĺžke behu tohto algoritmu by sme ho boli schopní použiť iba pre veľmi malé počty čítaní. Preto je potrebné využívať efektívnejšie metódy, ktoré nebudeme v tejto práci rozoberať.

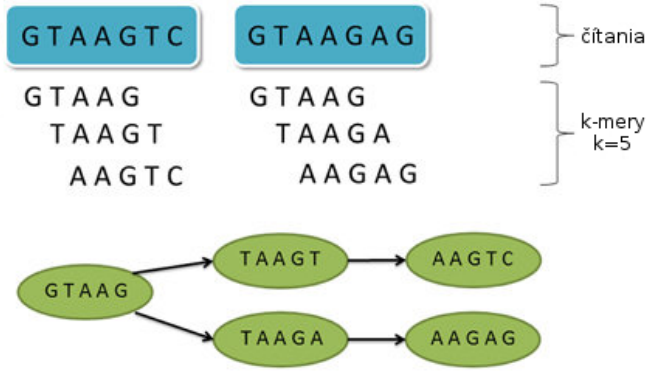
1.5.2 De Bruijnov graf

Majme množinu čítaní R, ktorú sme získali sekvenovaním a parameter k. Ľubovoľný súvislý podreťazec dĺžky k nazveme k-merom. De Bruijnov graf je graf $H_k = (V, E)$, kde V je množina všetkých k-merov čítaní z množiny R, pričom dva k-mery u a v tvoria hranu $(u, v) \in E$, ak u je prefixom dĺžky k a v je sufixom dĺžky k nejakého podreťazca z R dĺžky $k + 1$ [32]. Príklad takéhoto grafu môžeme vidieť na obrázku 1.2.

V ideálnom prípade by pre každý chromozóm genómu v grafe existoval jednoznačný ťah, ktorý by tento chromozóm reprezentoval. Takýto ťah však niekedy nemusí byť jednoznačný, alebo vôbec nemusí existovať. V takýchto prípadoch sa snažíme nájsť čo



Obr. 1.1: Príklad grafu prekryvov pre čítania $ACTGAATGCA$, $CACTAACTG$, $GCAGAGCATG$, $ATGCATTGCA$ a $TGCAAACCAC$



Obr. 1.2: Príklad de Bruijnového grafu pre čítania $GTAAGTC$ a $GTAAGAG$, $k = 5$

najväčšie súvislé časti genómu zodpovedajúce kontigom. Takýto kontig bude reprezentovaný ťahom v grafe, kde do každého vrcholu bude vchádzať a z každého vrcholu bude vychádzať práve jedna hrana.

1.6 Zarovnávanie sekvencií

Vďaka znižovaniu náročnosti a ceny sekvenovacích technológií máme v dnešnej dobe k dispozícii obrovskú databázu osekvenovaných genómov. Pri analýze genómov často chceme jednotlivé genómy porovnávať, hľadať ich podobnosti, alebo odlišnosti. To nám pomáha v kategorizácii a zisťovaní funkcií jednotlivých častí genómu. Na zisťovanie podobností genómov alebo ich častí sa využíva proces zarovnávania sekvencií. Je jedným zo základných problémov genomiky, vedy skúmajúcej genómy organizmov. Cieľom zarovnávania je vytvoriť mapovanie medzi znakmi dvoch alebo viacerých sekvencií [5]. Toto mapovanie nazývame *zarovnanie*. V prípade, že zarovnávame viac ako dve sekvencie, hovoríme o mnohonásobnom zarovnaní. Pri mutácií génov sa objavujú aj straty báz–*delécie*, pri ktorých sú niektoré bázy z genómu odstránené. Taktiež nastávajú pri-

dania báz–*inzercie*, kedy sú do genómu nové bázy pridané. Tieto javy budeme spoločne nazývať *indely*. Indely v praxi zobrazujeme tak, že do jednej zo sekvencií pridáme niekde medzi bázy pomlčku. Tá symbolizuje chýbajúcu bázu. Báza, ku ktorej sme pomlčku zarovnali, predstavuje inzerciu alebo deléciu. Keď sekvencie aj s pomlčkami zapíšeme pod seba, jednotlivé stĺpce prestavujú mapovania báz. Chceme, aby sa bázy v jednom stĺpci čo najčastejšie zhodovali.

Pri zarovnávaní zavedieme systém hodnotenia, podľa ktorého vieme dané zarovnanie ohodnotiť a určiť jeho kvalitu. Túto kvalitu nazveme *skóre* zarovnania [12]. Najjednoduchší spôsob hodnotenia je nasledovný. Ak sa bázy v stĺpci zhodujú, pripočítame bod. V prípade, že sa nezhodujú, bod odpočítame. Čím je výsledné skóre väčšie, tým považujeme zarovnanie za lepšie. Pri bodovacích systémoch sa taktiež berú do úvahy indely, často hodnotené odlišne od zhôd a nezhôd.

Poznáme dva druhy zarovnávania–globálne a lokálne. Pri globálnom zarovnaní je nutné namapovať všetky bázy oboch sekvencií tak, aby bolo maximalizované skóre tohto zarovnania. Vieme, že pri zostavovaní sekvencií môže dôjsť k chybám. Zároveň sekvencie podliehajú evolúcii, pri ktorej sa jednotlivé bázy, alebo celé úseky môžu meniť. Sekvencie sa preto nie vždy snažíme zarovnať celé do poslednej bázy. Pri lokálnom zarovnaní chceme zarovnať ľubovoľné súvislé podpostupnosti dvoch sekvencií tak, aby bolo skóre maximalizované. Na nájdenie lokálneho aj globálneho zarovnania sa využíva dynamické programovanie. Do tabuľky, kde riadky označujú bázy jednej sekvencie a stĺpce bázy druhej, budeme zaznačovať skóre jednotlivých prípadov. Pole so súradnicami i a j nám hovorí, aké je maximálne skóre zarovnania prvých i báz prvej sekvencie a prvých j báz druhej sekvencie. Začíname pri primitívnom prípade, pre $i = 0$ alebo $j = 0$ kedy je jedna zo sekvencií prázdna. Ich skóre je potom súčtom bodovania indelu pre každú pozíciu druhej sekvencie. Túto tabuľku postupne vypĺňame a na záver máme v pravom dolnom rohu tabuľky hodnotu maximálneho skóre globálneho zarovnania. Pre lokálne zarovnanie je možné použiť podobný prístup s istými úpravami [12].

1.6.1 Zarovnávanie ku grafu, GraphAligner

V podkapitole 1.5 sme popisovali reťazcové grafy. V tejto podkapitole sa pozrieme na to, ako dokážeme zarovnávať sekvencie ku reťazcovému grafu. Zároveň popíšeme program GraphAligner [29], ktorý túto činnosť vykonáva.

Reťazcové grafy sa používajú pri zostavovaní genómov, ale aj pri zarovnávaní sekvencií, mnohonásobných zarovnávaní sekvencií, či pri oprave chýb. Slúžia na reprezentáciu množiny sekvencií, napr. genetické variácie v populácii organizmov. Reťazcový graf nám ponúka možnosť jednoduchej reprezentácie zdieľaných aj unikátnych oblastí sekvencií.

Nech každý vrchol reťazcového grafu G obsahuje reťazec znakov. Reťazec sledu

vrcholov $(v_1, ..., v_n)$ je taká postupnosť znakov, ktorá je zreťazením reťazcov vrcholov v_1 až v_n v danom poradí. Presným zarovnaním sekvencie S ku grafu G nazývame taký sled vrcholov $(v_1, ..., v_k)$, ktorého reťazec je totožný so sekvenciou S. V prípade, že sa medzi reťazcom sledu a sekvenciou vyskytujú rozdiely, hovoríme o zarovnaní sekvencie ku grafu. Pri takomto zarovaní je, rovnako ako pri zarovnávaní sekvencií, možné určovať skóre zarovnania. Cieľom je vybrať zarovnanie sekvencie ku grafu tak, aby malo zo všetkých možných zarovnaní maximálne skóre.

V našej práci pre účel zarovnávania sekvencií ku grafu používame softvér GraphAligner [28]. Využíva paralelizáciu bitových operácií, čo prináša zrýchlenie voči iným algoritmom na zarovnávanie [29]. Vstupom pre tento algoritmus je de Bruijnov graf a sekvencia, ako množina kontigov alebo čítaní, ktoré chceme ku grafu zarovnať. Výstupom je zarovnanie sekvencie ku grafu ako postupnosť vrcholov s ďalšími informáciami o zarovnaní.

Kapitola 2

Popis problému a stav problematiky

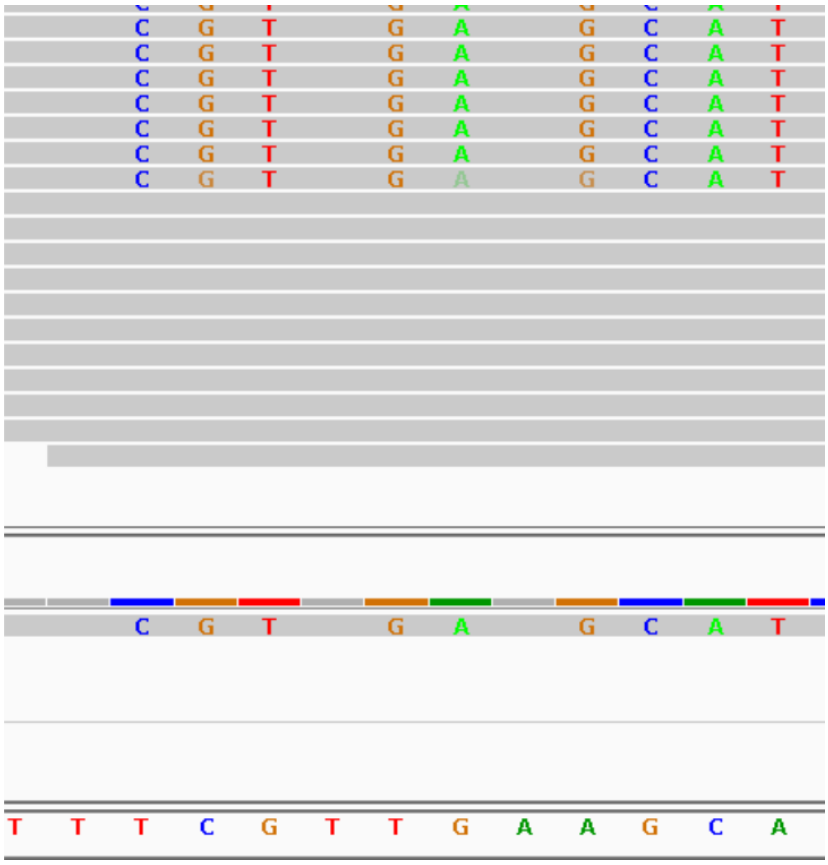
V tejto kapitole sa budeme venovať popisu problému detekcie chýb v sekvenciách. Popíšeme momentálny stav tejto problematiky a vybrané algoritmy, ktoré tento problém riešia. Zároveň popíšeme hlavnú myšlienku nášho algoritmu.

2.1 Chyby v zostavených genómoch

Množstvo osekvenovaných genómov exponenciálne rastie. S rozmachom sekvenovania novej generácie je len malé množstvo zostavených genómov aj detailne skúmaných, či podrobených dôslednej oprave chýb. Sekvencie, ktoré prešli takýmto procesom, budeme považovať za správne aj napriek chybám, ktoré sa v nich môžu vyskytovať. Budeme ich nazývať referenčné sekvencie. Jedným z príkladov genómu podrobeného detailnému skúmaniu je ľudský genóm. Vďaka jeho verziám s rôznymi opravami chýb je možné sledovať vylepšovanie sekvencie postupom času.

Existujú desiatky nástrojov na zostavovanie genómov. Výsledky týchto nástrojov však môžu obsahovať chyby, aj keď je k dispozícii vysoké pokrytie genómu krátkymi či dlhými čítaniami. Je ťažké predpovedať správanie sa týchto nástrojov na danom genóme, bez predošlej znalosti kompozície báz, opakovaných sekvencií, či veľkosti tohto genómu. Často je riešením spustiť viacero nástrojov na zarovnávanie s viacerými parametrami a vybrať z nich najlepší výsledok. Problémom pri určovaní chybovosti týchto genómov je, že vo väčšine prípadov nepoznáme referenčnú sekvenciu, oproti ktorej by sme zostavené sekvencie mohli porovnávať.

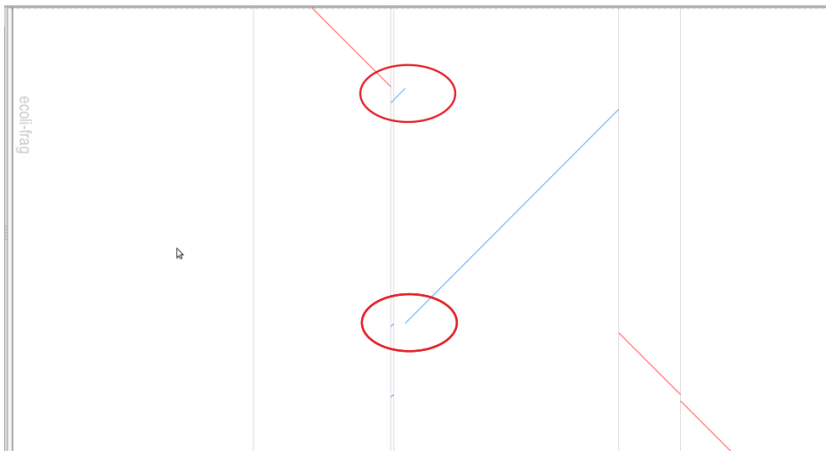
Pri sekvenovaní a zostavovaní genómu sa stretávame s viacerými druhmi chýb. Najzákladnejšie sú chyby na úrovni báz. Už pri samotnom sekvenovaní genómu sa môže stať, že sú niektoré bázy zle prečítané. Taktiež sa môže stať, že sa niektoré čítania pri zostavovaní zarovnajú na nesprávne miesto, kvôli podobnostiam rôznych častí genómu. Väčšina báz zle zarovnaného čítania sa bude zhodovať s bázami referenčnej sekvencie. Zvyšné bázy tohto čítania ale negatívne ovplyvnia výber správnej bázy na dané miesta.



Obr. 2.1: Sivé oblasti na obrázku zobrazujú krátke čítania zarovnané k sekvencii. Zostavenú sekvenciu ako reťazec báz vidíme v spodnom riadku. Nad ňou máme k tejto sekvencii zarovnanú referenciu. Farebne označené písmená v čítaniach a referencii sú bázy, ktoré sa nezhodujú s bázami v sekvencii. Vidíme, že aj napriek tomu, že v niektorých čítaniach bola báza určená rovnako ako v referencii, finálna báza sekvencie bola na viacerých miestach vybraná nesprávne.

Príklad takýchto čítaní môžeme vidieť na obrázku 2.1. V prípade, že je takto zle zarovnaných čítaní alebo zle osekvenovaných báz na jednom mieste v rôznych čítaniach viac, môže to vyústiť v chybnú bázu v zostavenom genóme.

Poznáme aj chyby rozsiahlejšieho charakteru. Jeden takýto problém vzniká pri opakovaných sekvenciách v genóme. Opakované sekvencie sú postupnosti báz, ktoré sa v genóme opakujú viackrát. Pri čítaniach obsahujúcich tieto postupnosti máme problém určiť, v ktorej oblasti genómu sa skutočne nachádzajú. Preto sú niektoré tieto časti pokryté vysoko nadpriemerne a iné podpriemerne. Málo pokryté oblasti je potom náročnejšie zostaviť, kvôli nedostatku informácií. Niektoré miesta kvôli nedostatočnej informácii nemusia byť zostavené vôbec. Tento problém je pri dlhých čítaniach menej výrazný. Je to z dôvodu, že sú tieto čítania dlhšie, a preto je málo pravdepodobné, že sa takto dlhé kusy sekvencie budú v genóme opakovať. Preto je jednoduchšie určiť, kde dlhé čítania patria. Ďalšou chybou je nesprávne určenie poradia väčších kusov sekven-



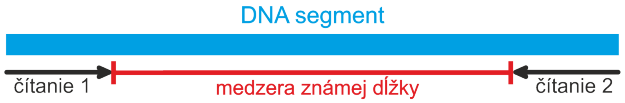
Obr. 2.2: Na tomto obrázku môžeme vidieť príklad rozsiahlejšej chyby. Na obrázku vidíme graf, kde y-ová os predstavuje referenčnú sekvenciu. Os x predstavuje zostavenú sekvenciu. Zvislé sivé čiary rozdeľujú sekvenciu na jej jednotlivé kontigy. Šikmé čiary na obrázku zobrazujú zarovnania jednotlivých kontigov ku referencii. V prípade, že je čiara modrá, kontig sa zarovnal ku referencii v normálnom smere. Ak je červená, zarovnal sa v opačnom smere. V ideálnom prípade by mal byť každý kontig zarovnaný jednou šikmou čiarou, čo znamená, že je ku referencii zarovnaný od začiatku do konca. Na obrázku sú červenou zakrúžkované dve chybové miesta. V spodnom krúžku môžeme vidieť, že sa kontig ku referencii nezarovanal od začiatku (prázdne chýbajúce miesto v krúžku hneď pri sivej čiare). V druhom krúžku vidíme, že začiatočná časť kontigu sa zarovnala až na neskoršie miesto, za predchádzajúce časti kontigu.

cie. Príklad takejto chyby môžeme vidieť na obrázku 2.2. Za chybu by sme mohli označiť aj miesta v skutočnom genóme, ktoré zostavenou sekvenciou vôbec neboli pokryté kvôli nedostatku alebo nejednoznačnosti dát.

2.2 Softvér na analýzu a opravu chýb

V dnešnej dobe existuje viacero nástrojov na analýzu a opravu chýb zostavených genómov. Ponúkajú možnosti detekcie a opravy malých ako aj väčších chýb. Základná myšlienka, ktorú mnohé z týchto softvérov zdieľajú, je využívanie krátkych spárovaných čítaní, ktoré sú namapované opätovne na genóm. Pomocou tohto mapovania následne hľadajú javy, ktoré by sa v správne zostavenom genóme nemali vyskytovať.

Spárované čítania (*pair-end reads*) sú čítania, ktoré sú osekvenované z oboch koncov toho istého segmentu DNA, pričom majú medzi sebou neosekvenovanú medzeru, ktorej dĺžka je približne známa, ako môžeme vidieť na obrázku 2.3. Toto párovanie pomáha napríklad pri určovaní správnej dĺžky repetitívnych sekvencií. V prípade, že niektoré čítanie obsahuje opakovanú sekvenciu, nevieme, kde presne je potrebné ho zarovnať.



Obr. 2.3: Ukážka spárovaných čítaní

Na to slúži jeho párové čítanie. Keďže je toto párové čítanie zo vzdialenejšieho miesta, repetitívna sekvencia sa v ňom už nenachádza. Keď zarovnáme toto čítanie, podľa medzery známej dĺžky vieme približne povedať aj to, kde sa má zarovnať prvé čítanie.

Ďalšia informácia, ktorú tieto softvéry využívajú, je skóre kvality pre jednotlivé bázy. Toto skóre nám hovorí, aká je pravdepodobnosť, že konkrétna báza v čítaní bola vybraná chybne. Pri sledovaní jednotlivých báz je pri oprave užitočné brať do úvahy aj toto skóre, keďže nám hovorí o relevantnosti vybranej bázy v konkrétnom čítaní. Tým vieme presnejšie určovať, ktorá báza sa na danom mieste mala reálne nachádzať.

2.2.1 REAPR

REAPR je jeden z nástrojov na detekciu chýb v sekvenciách bez toho aby potreboval referenčnú sekvenciu [17]. Jeho cieľom je ohodnotiť presnosť každej bázy a detegovať zle zostavené časti sekvencie. Rovnako ako iné nástroje využíva na určenie väčších chýb spárované čítania. Každé čítanie je najskôr zarovnané nezávisle od svojho párového čítania, aby nevznikali umelo zarovnané páry, ktoré by znižovali presnosť zarovnania. Základným princípom je počítanie pokrytia fragmentami, kde za fragment považujú oblasť medzi koncami správne spárovaných čítaní. Takéto pokrytie vypočíta pre každú bázu ako počet fragmentov, ktoré túto bázu pokrývajú. Následne hľadá odklony tejto hodnoty od teoreticky očakávaného pokrytia fragmentami. Pri príliš výraznej odchýlke u nejakej bázy ju označí za problémovú. Algoritmus následne hľadá miesta, kde sa nachádzajú dlhšie série takýchto problémových báz za sebou. Tie označí za chybné oblasti. Na určovanie presnosti jednotlivých báz využíva iba spárované čítania, ktoré sú zarovnané jednoznačne (nemôžu mať viacero možných miest zarovnania v genóme) a presne do poslednej bázy. Následne je báza označená za bezproblémovú, ak ju pokrýva aspoň istý počet takýchto čítaní (štandardné nastavenie je päť spárovaných čítaní).

2.2.2 Pilon

Cieľom nástroja Pilon je nielen ohodnotiť kvalitu sekvencie, ale zároveň opraviť miesta, o ktorých predpokladá, že sú nesprávne [34]. Tento nástroj by mal zvýšiť kvalitu zostaveného genómu opravou báz, opravou zle zostavených miest a dopĺňaním dier v genóme.

Vstupom pre Pilon je okrem opravovanej sekvencie ešte súbor, ktorý vznikne zarovnaním spárovaných čítaní ku genómu. Tieto čítania pred použitím filtruje a využíva iba správne zarovnané čítania so správnymi medzerami medzi pármi. Taktiež vynecháva informáciu z krátkeho úseku na koncoch čítaní, keďže tieto dáta považuje za menej dôveryhodné. Pre každú pozíciu v sekvencii zoberie údaje z výsledných čítaní. Zbiera informácie nielen o počte báz A, C, G, T na tejto pozícii v čítaniach, ale berie do úvahy aj ich skóre kvality. Podľa týchto informácií zaradí každú bázu skúmaného genómu do jednej z kategórií: potvrdené, zmenené, nejasné, nepotvrdené. V prípade, že je báza označená za zmenenú, vo výslednom opravenom genóme sa táto báza upraví na inú, podľa informácií z čítaní.

2.2.3 Tapestry

Softvér Tapestry je určený hlavne na analýzu menších genómov s menším počtom kontigov [13]. Jeho vstupom je zostavený genóm a dlhé čítania. Dlhé čítania podľa dĺžky filtruje, aby príliš krátke čítania, ktoré sa nezarovnajú unikátne, negatívne neovplyvnili výsledky. Tieto vyfiltrované čítania zarovnáva ku genómu a zhromažďuje bližšie informácie o jednotlivých kontigoch, ako napríklad veľkosti kontigov alebo hĺbku pokrytia čítaniami. Tieto informácie sa následne dajú vizuálne zobraziť a pomocou interaktívnej stránky, ktorú softvér vytvorí triediť, filtrovať alebo označovať. Vytvorený súbor kontigov sa následne dá vyexportovať. Softvér teda nehľadá konkrétne chybové miesta, ale ponúka možnosť jednoduchého zobrazenia štatistík o kontigoch a prácu s nimi.

2.3 Náš prístup k hľadaniu chýb pomocou dlhých čítaní

V tejto práci sa budeme sústreďovať na hľadanie chýb väčšieho charakteru, na čo chceme využiť informácie z dlhých čítaní a zarovnávanie ku de Bruijnovému grafu. Dlhé čítania majú väčšiu chybovosť a preto sa nebudeme sústreďovať na hľadanie malých lokálnych chýb. Na druhej strane nám poskytujú informáciu o rozložení väčších kusov sekvencie. Množstvo týchto čítaní a náhodnosť ich rozmiestnenia by mali zabezpečiť, že tieto čítania budú pomerne rovnomerne pokrývať celú skúmanú sekvenciu. Ak sa teda nejaké, nie príliš dlhé kusy zostavenej sekvencie nachádzajú v reálnej sekvencii, mali by sa aj dostatočne veľakrát nachádzať v dlhých čítaniach. Problémom pri samotnom zarovnávaní čítaní k zostavenej sekvencii je to, že niekedy existuje niekoľko možností, kam dané sekvencie zarovnať. V prípade nesprávneho výberu budú tieto miesta obsahovať chybné dáta a v iných miestach budú tieto dáta chýbať. Tento problém by mal čiastočne zjednodušiť práve spomínaný de Bruijnov graf. Ten nám prináša spôsob, ako

jednoducho reprezentovať opakované rovnako ako jednoznačné úseky sekvencie. Keďže opakované sekvencie sú v de Bruijnovom grafe reprezentované jedným vrcholom, odpadáva potreba rozhodovania sa, ku ktorej časti dané čítanie zaradíme. Táto výhoda grafu bola motiváciou jeho využitia v tejto práci.

Náš proces hľadania chýb bude vyzerať nasledovne. Z krátkych čítaní vytvoríme de Bruijnov graf. Ku vytvorenému grafu zarovnáme skúmanú sekvenciu a dlhé čítania. Takto získame sledy vrcholov grafu pre kontigy sekvencie a dlhé čítania. Predpokladáme, že ak bola zostavená sekvencia správna, tak sa podsledy jednotlivých kontigov budú nachádzať aj v dostatočnom počte dlhých čítaní. V prípade, že sa nejaké časti nenašli, môže to mať dva dôvody. Buď bola sekvencia na danom mieste zle zostavená, a tak túto časť nie je možné v dlhých čítaniach nájsť, keďže by tam mala byť iná. Druhý prípad je, že kontrolujeme príliš dlhé podsledy, ktoré presahujú aj dĺžku samotných dlhých čítaní. Budeme teda postupne prehľadávať kratšie podsledy zo skúmanej sekvencie a zisťovať, či sú dostatočne pokryté dlhými čítaniami. Ak nie, vyznačíme tieto miesta za potenciálnu chybu.

Kapitola 3

Návrh a implementácia softvéru

V tejto kapitole opíšeme detaily nášho algoritmu na hľadanie chýb v zostavených genómoch, ktorého hlavnú myšlienku sme popísali v podkapitole 2.3. Presne popíšeme vstupy, jednotlivé kroky, formáty výstupov nášho algoritmu a jazyky použité pri implementácii.

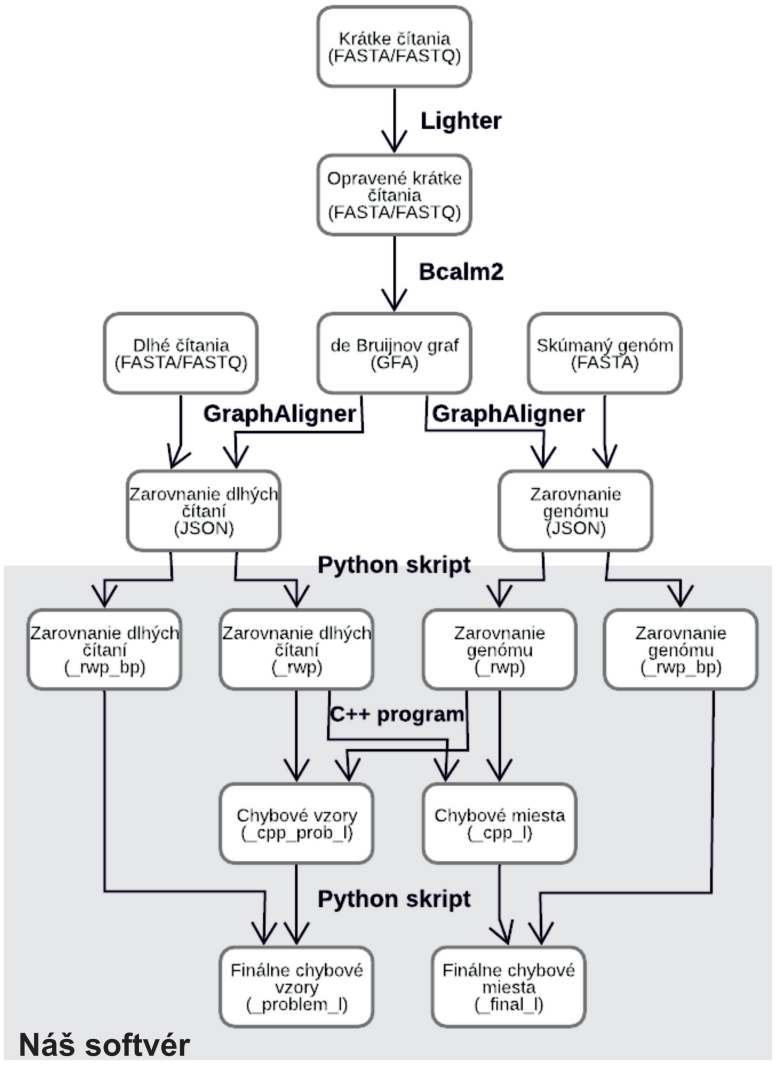
3.1 Jazyk

Pre naprogramovanie nášho algoritmu sme sa rozhodli použiť dva programovacie jazyky, Python a C++. Prvou voľbou bol Python, keďže v tejto práci bolo nutné spracovanie rôznych dátových formátov. Python obsahuje veľké množstvo knižničných funkcií, ktoré nám uľahčili prácu s reťazcami. Kroky algoritmu, pri ktorých sme využívali Python, zaberali lineárny čas v závislosti od veľkosti vstupného súboru. Preto nám aj napriek väčším dátam nižšia rýchlosť programovacieho jazyku Python neprekážala. Pri hlavnej časti algoritmu sme zvolili C++, keďže algoritmus vykonáva viacnásobné vyhľadávanie vo veľkej sade reťazcov a teda zásadná bola rýchlosť a veľkosť potrebnej pamäte. Finálne filtrácie a úpravy formátov sme opäť vykonávali v jazyku Python.

3.2 Proces spracovania dát

V tejto časti popíšeme celkové fungovanie nášho algoritmu a jeho hlavné súčasti a použité vstupné, výstupné a pracovné súbory. Celý tento proces je zhrnutý na na obrázku 3.1. Popis formátov súborov spomínaných v tomto texte sa nachádza v časti 3.3.

Prvotným vstupom pre náš algoritmus je viacero súborov. Prvým je skúmaná sekvencia, v ktorej chceme hľadať chybové miesta, uložená vo FASTA formáte. Ďalej potrebujeme krátke a dlhé čítania vo formáte FASTQ alebo FASTA, pričom tieto súbory môžu byť aj komprimované programom gzip.



Obr. 3.1: Diagram toku dát od vstupu po výstup.

Najprv sme využili softvér Lighter [31] na opravu krátkych čítaní odporúčaný autorom softvéru GraphAligner. Výsledkom tohto procesu sú opravené krátke čítania v rovnakom formáte ako pôvodné čítania. Následne sme opravené krátke čítania využili na vytvorenie grafu pomocou softvéru Bcalm2 [9]. Tento softvér vytvára de Bruijnov graf, ktorý následne kompaktuje. Pre vytvorenie grafu sme použili konštantu $k = 31$, čo hovorí, že vrcholy nekompaktovaného grafu reprezentujú sekvencie dĺžky 30. Pri kompaktovaní grafu sú jednoduché nevetviace sa cesty zlúčené do jedného vrcholu. V grafe sa preto nachádzajú vrcholy s reťazcami rôznych dĺžok. Tento softvér vytvorí graf vo formáte FASTA. Tento graf pre využitie v softvéri GraphAligner skriptom od tvorcov softvéru Bcalm2 pretransformujeme do formátu GFA. Nakoniec využijeme softvér GraphAligner na zarovnanie sekvencií a dlhých čítaní ku grafu [29]. Výsledkom procesu zarovnania je súbor v JSON formáte obsahujúci bližšie informácie o jednotlivých zarovnaniach ku grafu Na ďalšie spracovanie údajov už budeme používať nami vytvorený softvér.

JSON súbory sú v procese spracovania dát najobjemnejšie a obsahujú veľké množstvo nadbytočných informácií. Preto z nich pomocou skriptu v jazyku Python vyextrahujeme dáta o jednotlivých zarovnaniach, ktoré pre nás budú potrebné. Pre každý JSON súbor vytvoríme dva súbory s koncovkami _rwp a _rwp_bp. Prvý (_rwp) obsahuje pre každý kontig (respektíve dlhé čítanie) postupnosť vrcholov grafu. Táto postupnosť vrcholov predstavuje sled v grafe, ku ktorému sa sekvencia zarovnala a teda reťazec tohto sledu tvorí zarovnanú sekvenciu (s prípadnými úpravami podľa zarovnania). V prípade znamienka + sa sekvencia zarovnala k vrcholu v normálnom smere. Pri znamienku − sa sekvencia zarovnala k reverznému komplementu sekvencie vrcholu. Druhý súbor (_rwp_bp) pre každú sekvenciu obsahuje okrem vrcholov zarovnania aj hranice začiatku a konca oblasti sekvencie v bázach, ktorá sa k danému vrcholu zarovnala. Keď máme takto spracovaný vstup, JSON súbory môžeme kvôli uvoľneniu disku vymazať. Nasleduje hlavná časť hľadania chýb v jazyku C++.

Vstupom pre implementáciu algoritmu v jazyku C++ sú zarovnania dlhých čítaní a zostavenej sekvencie (_rwp). Najprv načítame informácie o dlhých čítaniach a následne hľadáme miesta zo zostavenej sekvencie, ktoré nie sú dlhými čítaniami dostatočne pokryté. Výsledkom tohto procesu sú dva súbory. Jeden obsahujúci chybové vzory (koncovka _cpp_l) a druhý obsahujúci chybové miesta (koncovka _cpp_prob_l). Bližšie informácie o tomto procese sa nachádzajú v podkapitole 3.4.

Pracovný výstup z hlavnej časti ešte dodatočne upravíme do finálneho tvaru skriptom v jazyku Python. Prvý výstupný súbor má v názve reťazec _problem a obsahuje informácie o nájdených zlých podsledoch bez ich presného výskytu v sekvencii. Druhý súbor má v názve reťazec _final a obsahuje polohy chýb v zostavenej sekvencii vo formáte BED. Ako uvidíme v kapitole 4.5, výsledné súbory sme následne detailnejšie skúmali, zisťovali ich nepresnosti a dôvod prečo sa tam tieto nepresnosti vyskytli. Tieto

informácie sme ďalej využili pre vylepšenie nášho algoritmu.

3.3 Formáty súborov

V predchádzajúcej podkapitole sme v popise procesu spracovania dát spomínali viacero využívaných formátov dát. V tejto podkapitole všetky formáty detailnejšie popíšeme.

FASTA Formát FASTA sa štandardne používa na ukladanie sekvencií DNA alebo proteínov [24]. Sekvencia vo FASTA formáte začína jednoriadkovým popisom nasledovaným jednotlivými riadkami sekvencie. Riadok s popisom je od samotnej sekvencie odlíšený znakom >, ktorý sa nachádza na začiatku riadku. V popise je uvedený identifikátor danej sekvencie a je možné, ale nie nutné, pridať aj ďalšie doplňujúce informácie o sekvencii. V tomto formáte náš softvér očakáva vstupné súbory s dlhými čítaniami a skúmanou sekvenciou.

FASTQ Formát FASTQ sa využíva na ukladanie sekvenačných čítaní spolu s informáciou o kvalite jednotlivých báz [10]. V tomto formáte sa striedajú štyri typy riadkov. Prvý riadok začína znakom @ a obsahuje popis čítania. Často sa v popise za názvom môžu nachádzať aj iné doplňujúce informácie, ako napríklad dĺžka čítania. V druhom riadku sa nachádza samotná sekvencia čítania. Tretí riadok obsahuje znak +, ktorý slúži na oddelenie sekvencie od reťazca určujúceho kvalitu. Štvrtý riadok je postupnosť ASCII znakov, ktoré určujú kvalitu jednotlivých báz, teda dĺžka tohto riadku je totožná s druhým riadkom. V tomto formáte sú niektoré súbory s krátkymi alebo dlhými čítaniami, ktoré sme použili.

GFA Formát GFA je využívaný na ukladanie reťazcových grafov [2]. V našej práci ho využívame na popis de Bruijnovho grafu vytvoreného z krátkych čítaní. Každý riadok súboru pozostáva z niekoľkých hodnôt oddelených tabulátorom. Prvý stĺpec označuje typ riadku, pričom poznáme štyri typy riadkov: hlavička, označená písmenom H, segment, označený písmenom S, prepojenie označené písmenom L a podmnožina označená písmenom C. Informácia o konštante k pre de Bruijnov graf sa nachádza v hlavičke súboru. Pre potreby našej práce sme využívali iba segmentový riadok, popis ostatných typov riadkov sa dá nájsť v špecifikácii [2]. Segmentový riadok po počiatočnom symbole S obsahuje meno segmentu, pri de Bruijnovom grafe je to meno vrcholu. V našej práci ako mená vrcholov využívame čísla, pričom vrcholy číslujeme od nuly. Za menom vrcholu nasleduje samotná sekvencia ako reťazec báz. Za ňou sa nachádza viacero nepovinných polí. Nás zaujíma pole označené $LN : i$: nasledované číslom, ktoré hovorí o dĺžke sekvencie vo vrchole.

JSON Formát JSON je určený na prenos dát, je jednoducho generovateľný a spracovateľný [3]. Je to textový, na jazyku úplne nezávislý formát. Je založený na ukladaní informácií v dvoch hlavných dátových štruktúrach . Prvou štruktúrou je neusporiadaná množina dvojíc meno–hodnota, takzvaný objekt. Objekt je uzatvorený zloženými zátvorkami, jednotlivé páry sú oddelené čiarkami a meno je od hodnoty oddelené dvojbodkou. Druhou štruktúrou je pole, ktoré je usporiadanou množinou hodnôt, pričom hodnoty sú oddelené čiarkami. V našom programe tento formát využívame ako zdroj informácií o zarovnaniach [25]. V súbore sú dáta usporiadané do nasledovnej štruktúry. Najväčšou jednotkou je záznam zarovnanej sekvencie. Začína sa kľúčom `sequence`, ktorý obsahuje zarovnanú sekvenciu ako reťazec báz. Nasleduje zložitejšia položka `path`, ktorá nám podáva informáciu o tom, ako sa daná sekvencia ku grafu zarovnala. Obsahuje jednotlivé záznamy `mapping`, ktoré popisujú poradie a orientáciu vrcholov, ku ktorým bola sekvencia zarovnaná. Záznam `mapping` ďalej obsahuje záznamy `position`, ktoré obsahujú vždy informáciu o jednom vrchole sledu. Pre každý vrchol máme informáciu o čísle vrcholu v grafe, ku ktorému sa časť sekvencie zarovnala, obsiahnutú v položke `node_id`. Nasleduje `offset`, ktorý hovorí, od ktorej bázy v sekvencii vrcholu sme sekvenciu z genómu zarovnali. Nasleduje položka `is_reverse`, ktorá hovorí o tom, či sme použili vrchol v normálnom alebo v obrátenom tvare. Nakoniec je súčasťou každého záznamu `position` množina položiek `edit`, ktorá hovorí, akými úpravami reťazcov vrcholov grafu získame zarovnanú výslednú sekvenciu. Na získanie tejto sekvencie prechádzame množinu položiek `edit`, pričom vždy berieme počet báz `from_length` z reťazca grafu a `to_length` báz zo sekvencie. V prípade zmeny je zastúpená položka `sequence`, ktorá hovorí, na akú sekvenciu sa dané bázy menia. Rôzne kombinácie týchto hodnôt reprezentujú odlišné operácie nad reťazcami (zhody, substitúcie, inzercie a delécie). Nakoniec záznam na najvyššej úrovni obsahuje položku `name`, kde sa nachádza meno zarovnanej sekvencie, položku `score`, kde je skóre danej sekvencie a položku `identity`, ktorá hovorí o tom, aké percento báz sa zarovnalo ako zhody.

BED Formát BED sa používa v bioinformatike na uloženie oblastí génov so súradnicami v bázach a ďalšími doplňujúcimi informáciami [1]. Každý riadok súboru reprezentuje jednu oblasť v sekvencii a obsahuje 3 povinné stĺpce, pričom môže obsahovať 9 ďalších nepovinných stĺpcov oddelených tabulátorom. Prvá hodnota v riadku označuje meno sekvencie, ku ktorej sa daný riadok viaže (napríklad meno kontigu). Druhý a tretí stĺpec určujú začiatok a koniec oblasti sekvencie. V ďalších stĺpcoch sa môžu nachádzať informácie ako napríklad skóre oblasti. V našom programe tento formát používame na zapísanie určenie nami nájdených chýb v zostavenej sekvencii.

_rwp Súbor _rwp je pomocný súbor, v ktorom máme uložené sledy zarovnaní jednotlivých sekvencií. Tento súbor obsahuje pre jedno zarovnanie sekvencie (kontigu alebo dlhého čítania) vždy dvojicu riadkov. V prvom riadku je meno zarovnanej sekvencie. V druhom riadku je postupnosť vrcholov grafu so znamienkom, ku ktorým sa daná sekvencia zarovnala, oddelených medzerou. V prípade znamienka + sa sekvencia zarovnala k danému vrcholu v smere, v ktorom je zapísaný v GFA súbore. Pri znamienku − sa zarovnala k reverznému komplementu sekvencie vrcholu. Tento formát nám umožňuje jednoducho načítať sledy zodpovedajúce jednotlivým zarovnaniam a pristupovať k ich podsledom.

_rwp_bp Súbor _rwp_bp je podobný súboru _rpw, pričom ale podáva bližšiu informáciu o tom, ktorá oblasť sekvencie sa k danému vrcholu grafu zarovnala. Jedno zarovnanie sekvencie je reprezentované jedným riadkom, v ktorom sú hodnoty oddelené tabulátorom. Ako prvý je reťazec obsahujúci meno kontigu, alebo čítania. Pre každý vrchol zarovnania nasleduje trojica čísel. Číslo vrcholu so znamienkom, ku ktorému sa časť sekvencie zarovnala, poradové číslo bázy, od ktorej sa sekvencia zarovnala a číslo bázy, po ktorú sa sekvencia zarovnala.

_problem Súbor _problem používame na zapisovanie informácie o nájdených chybových podsledoch. Názov tohto súboru okrem mena sekvencie a špecifického slova `problem` obsahuje na konci znak `l` nasledovaný celým číslom, ktoré značí limit na počet čítaní pre určenie chyby. Každý podsled je opísaný na jednom riadku. Prvá hodnota je samotný podsled ako postupnosť čísel vrcholov so znamienkom oddelených medzerami. Druhá hodnota označuje počet dlhých čítaní, v ktorých sa daný vzor nachádza. Na treťom a štvrtom mieste sa nachádzajú informácie o výskyte skrátených podsledov, ktoré by sa dali použiť pri ďalšej filtrácii chybových miest. Posledné hodnoty riadku obsahujú dĺžku podsledu ako počet vrcholov a dĺžku podsledu v bázach. Tú budeme počítať ako súčet dĺžok reťazcov prislúchajúcich jednotlivým vrcholom podsledu okrem prvého a posledného. Pre sledy dĺžky jedna a dva to bude nula. Prvý a posledný vrchol do dĺžky nezapočítavame, lebo nemusia byť úplne pokryté zarovnaním. Túto dĺžku teda môžeme chápať ako dolný odhad dĺžky problematickej oblasti v sekvencii.

_final Do tohto súboru zapisujeme informácie o polohe nájdených chybových miest vo vstupných kontigoch. Súbor má názov obsahujúci meno skúmaného genómu, kľúčové slovo `final` a znak `l` nasledovaný celým číslom označujúcim limit na počet čítaní pre chyby. Na zapisovanie polohy chýb používame formát BED popísaný vyššie. V našom prípade využijeme nasledovné stĺpce. Prvý stĺpec obsahuje názov kontigu. Druhý a tretí stĺpec obsahujú začiatok a koniec oblasti, kde sa chyba nachádza. Za tým nasleduje samotný zlý podsled, ako postupnosť vrcholov so znamienkami oddelených čiarkami.

Posledná hodnota označuje počet dlhých čítaní, v ktorých sa podsled nachádzal.

3.4 Hlavný algoritmus

V tejto podkapitole popíšeme detailne kľúčovú časť nášho softvéru písanú v C++. Vstupom pre túto implementáciu sú dva súbory s príponou _rwp, ktoré obsahujú sledy zarovnaných čítaní a kontigov. Najprv načítame informácie z dlhých čítaní a následne budeme podľa týchto informácií hľadať chyby v kontigoch. Pre jednoduchosť implementácie budeme sledy vrcholov ukladať ako reťazce obsahujúce postupnosť čísel vrcholov so znamienkami oddelených medzerami. Podsled reprezentujeme ako súvislý podreťazec sledu, pričom začiatok ani koniec podsledu nesmie byť vo vnútri oblasti od znamienka po koniec čísla vrcholu (opäť teda získame validný sled). Konkrétny podsled budeme nazývať *vzor*. Pri určovaní chýb chceme vyznačovať miesta, ktoré sú málo pokryté dlhými čítaniami. Limit, ktorý hovorí aspoň koľko dlhých čítaní musí konkrétny vzor pokrývať na to, aby nebol chybový, nazývame *ch_limit*. Je to teda minimum výskytov v dlhých čítaniach aby vzor nebol chybový. Keďže príliš dlhé vzory môžu presahovať dĺžku dlhých čítaní, často by boli prehlásené za chybu. Preto je vhodné určiť limit na dĺžku vzoru (v počte vrcholov vzoru), ktorú označíme *v_limit*. V našich experimentoch sme uvažovali vzory dĺžky do 5 (vrátane).

V prvom kroku vytvoríme všetky možné podsledy zarovnaní dlhých čítaní, ktoré obsahujú najviac *v_limit* vrcholov. Všetky tieto podsledy uložíme do dátovej štruktúry hašovacia tabuľka. Využili sme implementáciu z C++ STL knižnice, `unordered_map`. Túto štruktúru sme zvolili vďaka rýchlemu vkladaniu prvkov a vyhľadávaniu existencie prvkov v tabuľke. Kľúčmi hašovacej tabuľky sú samotné reťazce podsledov a hodnoty určujú počty výskytov podsledov v dlhých čítaniach. Pri vkladaní podsledu sa pozrieme, či sa už v hašovacej tabuľke nachádza. Ak nie, vložíme tam nový záznam s hodnotou jedna, keďže sa podsled v čítaniach vyskytoval prvýkrát. V prípade, že sa daný podsled ako kľúč v tabuľke nachádza, zvýšime jeho hodnotu v tabuľke o jedna.

Keď máme takto spracované dlhé čítania, ideme prehľadávať kontigy skúmanej sekvencie. Kontigy budeme spracúvať postupne jeden po druhom. V _rwp súbore môže mať jeden kontig aj viacero záznamov zarovnaní. Tieto jednotlivé zarovnania pokrývajú disjunktné (resp. obsahujúce malý zanedbateľný prekryv) časti kontigu, medzi ktorými nebolo nájdené spojenie v grafe. Keďže sú takmer nezávislé, budeme ich spracúvať postupne. Pre každé zarovnanie vytvárame podsledy (vzory) rovnako ako pri dlhých čítaniach. Vieme, že DNA má dve navzájom komplementárne vlákna, ktoré je potrebné brať do úvahy. Preto okrem vzoru musíme kontrolovať výskyt opačného vzoru (z rovnakého miesta v komplementárnom vlákne). Tento vzor vytvoríme tak, že v pôvodnom vzore obrátime poradie vrcholov a zmeníme znamienka na opačné. Výskyt

tohto opačného vzoru nám hovorí o zarovnaní komplementárneho vlákna ku grafu. Pre každý vzor a jeho opačný vzor sa pozrieme do hašovacej tabuľky, koľkokrát sa tieto vzory v dlhých čítaniach nachádzajú. Súčet týchto čísel hovorí o pokrytí vzoru dlhými čítaniami. V prípade, že sa vzor zo sekvencie v podsledoch dlhých čítaní vyskytoval viackrát ako je *ch_limit*, budeme tento vzor považovať za správny. V prípade, že sa vyskytoval v rovnakom alebo menšom počte podsledov dlhých čítaní, budeme ho považovať za chybu. Informáciu o počte jednoducho zistíme z hašovacej tabuľky pomocou vyhľadania existencie vzoru ako kľúča v tabuľke. Pri existencii kľúča v tabuľke je počet podsledov dlhých čítaní hodnota v tabuľke. V prípade neexistencie kľúču je to nula. Takto prejdeme všetky vzory všetkých kontigov a zhromaždíme informácie o chybových miestach.

Častokrát sa stáva, že jednu chybu v genóme označíme viacerými vzormi. Ako príklad si zvoľme sled $+1 + 2 + 3$ v kontigu skúmanej sekvencie. V prípade, že sa vrchol $+2$ v dlhých čítaniach nenájde dostatočne veľakrát, všetky vzory obsahujúce tento vrchol by sme označili za chybu. V našom prípade by sme vzory $+2$, $+1 + 2$, $+2 + 3$ aj $+1 + 2 + 3$ označili všetky za chybové, aj keď sa vo veľkej miere prekrývajú. Keďže ale reprezentujú rovnaký problém, rozhodli sme sa vybrať iba vzory, ktoré sú vzhľadom na postupnosť vrcholov minimálne. Ak nejaký vzor obsahuje kratší vzor, ktorý už bol označený za chybový, za chybu ho neoznačíme. Takto sa vyhneme nadbytočným opakujúcim sa chybám. Keďže vzory vytvárame od kratších ku dlhším, je jednoduché udržiavať si množinu minimálnych chybových vzorov. Opäť využijeme hašovaciu tabuľku. Ak nájdený vzor ešte v tabuľke nie je, vložíme ho tam, lebo je minimálny. V prípade, že je vzor totožný s kľúčom tabuľky, ide o rovnaký vzor, ale na inom mieste, preto tento vzor za chybu označíme. V prípade vzorov dlhších ako jeden vrchol prechádzame všetky jeho podvzory (resp. podsledy) a zisťujeme, či sa nachádzajú v hašovacej tabuľke. Ak sa ľubovoľný z podvzorov (okrem celého vzoru) nachádza v tabuľke, neoznačíme ho. V prípade, že sa tam žiadny z podvzorov nenachádza, pridáme samotný vzor do tabuľky a označíme ho za chybu. Keďže vzory testujeme od kratších ku dlhším, nemôže sa stať, že by sa nám do tabuľky dostal vzor, ktorý nie je minimálny.

Kapitola 4

Výsledky experimentov

Algoritmus na hľadanie chýb v genóme, popísaný v predchádzajúcej kapitole, sme testovali na reálnych biologických dátach. V tejto kapitole popíšeme použité dáta, nastavenia softvéru, získané výsledky a ich interpretáciu.

4.1 Použité dáta

Na testovanie nášho softvéru sme zvolili dva genómy, pričom z každého sme mali k dispozícii referenčnú sekvenciu, dlhé čítania a krátke čítania. Zároveň sme využili sekvencie zostavené rôznymi softvérmi. Bol to softvér SPAdes [6], Velvet [35], Miniasm [21] a softvér Racon [33], ktorý bol využitý pri oprave sekvencie zostavenej softvérom Miniasm. Tieto automaticky zostavené sekvencie môžu obsahovať chyby, ktoré môžeme skúmať naším softvérom. Prehľad informácií o jednotlivých sekvenciách a použitých čítaniach sa nachádza v tabuľkách 4.1 a 4.2.

Prvou vzorkou bola baktéria *Escherichia coli*, v ďalšom texte skrátene *E. coli*. Je to baktéria žijúca prevažne v črevách ľudí a väčšiny teplokrvných živočíchov. Taktiež je dlhodobo intenzívne študovaná ako modelový organizmus. Z genómu tejto baktérie sme zvolili kratší úsek dĺžky 500Kbp (kilobázových párov, teda 500 000bp). Tieto dáta sme získali z materiálov pre predmet Integrácia dátovych zdrojov. Keďže sú dáta menej rozsiahle, poskytli nám možnosť rýchleho a opätovného testovania zmien v našom algoritme a otestovanie veľkého množstva nastavení nášho softvéru. Referenčná sekvencia tohto genómu by mala byť veľmi málo chybová.

Druhou vzorkou bola sekvencia z kvasinky *Saprochaete fungicola*, ďalej iba kvasinka [8]. Kvasinky sú jednobunkové huby. Genóm tejto kvasinky má približne 20Mbp (megabázových párov, teda 2 000 000bp). Keďže je objem dát oveľa väčší, nebolo už možné testovanie toľkých možností nastavenia softvéru, ale na druhej strane tieto dáta sú realistické pre skutočné skladanie menšieho genómu. K dispozícii sme mali referenčnú sekvenciu, ktorá ale oproti referenčnej sekvencii E. coli obsahuje viac chýb. Ďalej sme

Tabuľka 4.1: Prehľad použitých zostavení genómov. Stĺpec využité čítania nám hovorí o čítaniach využitých pri zostavovaní daných sekvencií, pričom *D* predstavuje dlhé čítania a *K* krátke. V prípade veľkých písmen boli čítania využité ako primárny zdroj dát a v prípade malých ako doplňujúca informácia pri zostavovaní.

	Genóm	***Celková dĺžka (bp)***	***Počet kontigov***	***Najdlhší kontig (bp)***	***Využité čítania***
Referencia	E. coli	500 000	1	500 000	?
Spades1	E. coli	485 988	13	141 900	K
Spades2	E. coli	500 991	6	500 200	K,d
Spades3	E. coli	486 220	15	141 900	K,d
Velvet	E. coli	485 825	21	141 654	K
Velvet2	E. coli	484 371	18	142 006	K
Referencia	kvasinka	20 189 518	6	5 398 951	D,k
Spades	kvasinka	20 167 932	367	1 075 911	K,d
Miniasm	kvasinka	20 180 834	10	5 351 644	D

použili dve sekvencie zostavené z čítaní Illumina a Nanopore. Dáta pre kvasinku je možné stiahnuť z odkazu v použitej literatúre [4].

4.2 Testovanie presnosti výsledkov nášho algoritmu

Na to, aby sme vedeli náš algoritmus vyhodnotiť z hľadiska úspešnosti pri odhaľovaní skutočných chýb v zostavení genómu, potrebujeme v prvom rade množinu chýb v zostavení, ktoré budeme považovať za správnu odpoveď. Tieto chyby budeme označovať referenčné chyby. Okrem toho potrebujeme vedieť rozhodnúť, či sa referenčná chyba a chyba nájdená našim algoritmom dajú považovať za totožné a určiť vhodné numerické miery presnosti. Postupy na testovanie popísané v tejto podkapitole potom použijeme na získanie výsledkov v podkapitole 4.5.

Nakoľko sa náš nástroj zameriava iba na určité typy chýb a navyše niektoré chyby nenájde kvôli obmedzeniam použitého grafu a zarovnaní ku grafu, rozhodli sme sa porovnávať naše výsledky najmä s ideálnym výsledkom, ktorý by sme mohli dostať pri použití toho istého grafu.

Referenčné chyby sme teda zostrojili s využitím referenčnej sekvencie. Keďže jej zarovnania ku grafu považujeme za ideálne sledy, budeme porovnávať, či sa podsledy zo skúmanej sekvencie vyskytujú v referenčných sledoch. Využili sme na to náš softvér, pričom namiesto dlhých čítaní sme mu zadali referenciu. Limit pre chybovosť sme v

Tabuľka 4.2: Prehľad použitých čítaní. Položka medián hovorí o mediáne dĺžok daných čítaní. Chybovosť hovorí o editačnej vzdialenosti čítaní zarovnaných ku referencii v pomere ku počtu namapovaných báz.

Čítania	***Počet***	***Celková dĺžka(bp)***	***Najdlhšie čítanie(bp)***	***Medián (bp)***	***Chybovosť (%)***
E. coli krátke	195 938	29 586 638	151	151	0.68
E. coli dlhé	2 255	22 145 281	73 039	6 662	18.1
Kvasinka krátke	40 313 550	4 071 668 550	101	101	0.16
Kvasinka dlhé	309 350	3 007 292 812	251 379	4 159	15.89

tomto prípade nastavili na nulu, teda vzor je chybový iba v prípade, že sa vôbec v referencii nenašiel. Je to z dôvodu, že kým určitá oblasť genómu je väčšinou pokrytá viacerými čítaniami, referenčnou sekvenciou je typicky pokrytá iba raz. Chyby nájdené týmto spôsobom budeme považovať za referenčné chyby. Správnosť týchto chýb však môžu skresľovať problémy pri zarovnávaní, ktoré budú bližšie vysvetlené v podkapitole 4.3.

Okrem zostavenia množiny referenčných chýb potrebujeme rozhodnúť, či sa referenčná a nájdená chyba zhodujú. To sme vykonávali nasledovne. Vo výslednom súbore máme pre jednotlivé kontigy vypísané oblasti, ktoré sú podľa nás chybné, pričom sa tieto oblasti môžu prekrývať. Prvým krokom pri analýze bolo jednotlivé prekrývajúce sa alebo susediace úseky pospájať do jedného úseku, keďže prekrývajúce sa úseky sa pravdepodobne týkajú tej istej chyby v skladaní. Takto sme získali súbory s neprekrývajúcimi sa oblasťami referenčných aj nájdených chýb. Na skladania úsekov a neskôr na získavanie prekryvov oblastí dvoch súborov sme využili softvér BEDTools [27].

V ďalšom kroku sme skúmali, ktoré z oblastí nájdených chýb sa prekrývajú s nejakou oblasťou referenčnej chyby. Nebudeme vyžadovať presnú zhodu oblastí vrátane pozície oboch okrajov, keďže nájdené chyby môžu mať posunuté okraje kvôli rôznym nepresnostiam. Preto za správne označujeme nájdené chyby, ktoré sa prekrývajú s referenčnými chybami v aspoň jednej báze. Takúto nízku hranicu sme zvolili z dôvodu, že cieľom našej práce nie je presne určiť chybné bázy, ale poukázať na možné problémové úseky v genóme. Týmto spôsobom vieme pre každú chybu jednoznačne určiť, či je správna alebo nesprávna.

Keď máme takto ohodnotené všetky úseky, využijeme štatistické metriky na určenie presnosti našich výsledkov. Oblasti, ktoré boli správne určené za chybu (prekrývali sa s referenčnou chybou) budeme nazývať skutočne pozitívne (true positive). Oblasti, ktoré náš program nesprávne určil za chyby, označujeme falošne pozitívne (false posi-

tive). Oblasti, ktoré sme za chyby neoznačili, ale chybami v skutočnosti sú (neoznačené referenčné chyby), označujeme falošne negatívne (false negative). Pomocou týchto metrík môžeme určiť presnosť (precision) a úplnosť (recall) algoritmu. Presnosť nám hovorí, aká je pravdepodobnosť, že nájdená chyba je našim algoritmom určená správne, zatiaľ čo úplnosť hovorí, aké percento referenčných chýb sme naším algoritmom objavili. Presnosť je podielom skutočne pozitívnych oblastí a všetkých chybových (súčet skutočne pozitívnych a falošne pozitívnych). Úplnosť je všeobecne podielom skutočne pozitívnych oblastí a všetkých chybových oblastí (súčet skutočne pozitívnych a falošne pozitívnych) [16]. V našom prípade sme rátanie úplnosti potrebovali upraviť. Keďže dve rôzne nájdené chyby môžu pretínať rovnakú referenčnú chybu, v tomto prípade by bola úplnosť 2/1, čo je zrejme nezmysel. Preto sme úplnosť vyjadrili ako podiel počtu referenčných chýb, ktoré sa pretínali s ľubovoľnou nájdenou chybou ku celkovému počtu referenčných chýb. Takto vieme, aký podiel správnych (referenčných) chýb sme naším algoritmom odhalili.

4.3 Problémy softvéru GraphAligner

Pri používaní softvéru GraphAligner sme v procese testovania narazili na problém, ktorý silne ovplyvňoval výsledky našej práce, a preto ho bolo potrebné preskúmať. Týmto problémom bolo vynechávanie veľkých kusov sekvencie pri zarovnávaní. Prehľad celkovej dĺžky úsekov jednotlivých sekvencií, ktoré sa vôbec nezarovnali ku grafu, vidíme v tabuľke 4.3.

Zatiaľ čo pri E. coli sú veľkosti týchto nezarovnaných častí zanedbateľné (v najhoršom prípade dosahujú 0.06% genómu), pri kvasinke je situácia horšia. Najhoršie sa zarovnala sekvencia Miniasm, kde nezarovnané časti tvorili až 10% genómu, lepšie dopadla referencia so 6% a najmenšie problémy u kvasinky nastali pri zostavení Spades, kde nezarovnané časti tvorili 2% genómu. Za príčinu lepšieho zarovnania sekvencie Spades pre kvasinku a všetkých sekvencií pre E. coli považujeme dáta využité pri zostavovaní daných sekvencií. Keďže spomínané sekvencie boli zostavené primárne z krátkych čítaní (rovnako ako graf), predpokladáme, že podobnosť ku grafu bola vyššia. Preto bolo pre softvér jednoduchšie zarovnať tieto sekvencie ako zarovnať zostavenia získané primárne z dlhých čítaní (referenciu kvasinky a Miniasm).

Tento problém mal dva dôsledky. Prvým bolo, že sme nedokázali dostatočne presne určit sledy referenčnej sekvencie. Referenčné chyby určujeme tak, že zisťujeme, či sa vzory (konkrétne podsledy) skúmanej sekvencie nachádzajú v sledoch referenčnej sekvencie. Sledy častí referencie, ktoré sa nezarovnali, nepoznáme. Nevieme preto povedať, či by sa v týchto momentálne neznámych sledoch niektoré nami určené referenčné chyby nenachádzali. Tieto chyby by boli falošne určené referenčné chyby. Druhý prob-

Tabuľka 4.3: Prehľad nezarovnaných sekvencií. V stĺpci dĺžka nezarovnaných sekvencií sa nachádza suma dĺžok sekvencií, ktoré sa vôbec nezarovnali ku grafu. V druhom stĺpci sa nachádza percento genómu, ktoré tieto nezarovnané časti tvoria.

Sekvencia	***Genóm***	***Dĺžka nezarovnaných sekvencií(bp)***	***Časť genómu(%)***
Referencia	E. coli	5	0.001
Spades1	E. coli	317	0.065
Spades2	E. coli	317	0.063
Spades3	E. coli	317	0.065
Velvet	E. coli	5	0.001
Velvet2	E. coli	5	0.001
Referencia	kvasinka	1 332 163	6
Spades	kvasinka	423 068	2
Miniasm	kvasinka	2 076 920	10

lém nastával pri skúmaných sekvenciách. Keďže program nezarovnal celé sekvencie, nezarovnané časti nemohli byť otestované na výskyt chýb naším algoritmom. Preto sme skúmali možnosti celkovej alebo aspoň čiastočnej opravy tohto problému

Keďže boli niektoré nezarovnané úseky dosť dlhé a vyzerali byť správne zostavené, zdalo sa nám málo pravdepodobné, že by sa vôbec nedali zarovnať ku grafu. Preto sme sa rozhodli zo sekvencie vystrihnúť časti, ktoré sa nezarovnali a následne iba tieto časti opätovne zarovnať ku grafu. Časti, ktoré sa zarovnali pri prvotnom zarovnaní celej sekvencie budeme nazývať primárne zarovnané. Sekvencie zarovnané dodatočne budeme nazývať sekundárne zarovnané. Keďže sme nechceli, aby v prípade zarovnania vznikli problémy na rozhraniach primárne a sekundárne zarovnaných sekvencií, každú sekundárne zarovnávanú sekvenciu sme predĺžili z každej strany o 1000bp, aby tak vznikol medzi jednotlivými zarovnaniami prekryv.

Pri sekundárnom zarovnaní sa nám istú časť primárne nezarovnaných úsekov zarovnať podarilo. Niektoré časti ale aj napriek tomu zostali stále nezarovnané. Taktiež sme zistili, že pri zarovnaní opačného vlákna vznikajú mierne odlišnosti od zarovnania pôvodného vlákna a nezarovnané časti nie sú úplne zhodné. Z týchto pozorovaní sme dospeli k finálnym riešeniam problému. Keďže referenciu a skúmané zostavenia genómov využívame v procese hľadania chýb rôznymi spôsobmi, nepodarilo sa nám dospieť k jednotnému riešeniu tohto problému, ktoré by bolo ideálne pre oba prípady.

Keďže z referencie zisťujeme referenčné chyby, chceme ju mať čo najviac pokrytú (aby nevznikal problém falošných referenčných chýb). Postupovali sme teda nasledovným spôsobom. Spojili sme pôvodné vlákno a jeho reverzný komplement do jedného

Tabuľka 4.4: Prehľad nezarovnaných sekvencií po oprave. V tabuľke vidíme sumu dĺžok nezarovaných sekvencií, rovnako ako percento genómu, ktoré tieto časti pokrývajú po nami spomínaných úpravách. Pri referencii sme využili opačné vlákno a sekundárne zarovnanie. Pri sekvencii Miniasm sme využili iba sekundárne zarovnanie.

Sekvencia	***Genóm***	***Dĺžka nezarovnaných sekvencií(bp)***	***Časť genómu(%)***
Referencia	kvasinka	1 901	0.0094
Miniasm	kvasinka	61 536	0.3

súboru. Túto sekvenciu sme nechali zarovnať ku grafu. Tak sme pokryli časti pomocou pôvodného aj reverzného zarovnania. Časti z pôvodného aj opačného vlákna, ktoré sa v primárnom zarovnaní nenachádzali, sme vystrihli, predĺžili o 1000bp z každej strany a opätovne zarovnali. . Týmto sme väčšinu nezarovnaných miest z referencie odstránili. V prípade, že sa zarovnajú rovnaké sekvencie z opačných vlákien na rôzne miesta, jedno zo zarovnaní môže byť nesprávne a potenciálne zakryť niektoré referenčné chyby. To je však pre nás prípustnejšie ako prehlásiť falošné chyby za správne.

Pri skúmaných sekvenciách by bolo rozdielne zarovnanie opačných vlákien problematickejšie. Ak by bolo jedno vlákno zarovnané nesprávne, spôsobilo by chyby, ktoré by sa na správne zarovnanom vlákne nenachádzali. Takto by mohlo vzniknúť veľa falošných chýb spôsobených pridaním opačného vlákna. Preto sme sa v prípade skúmaných sekvencií rozhodli zarovnať iba pôvodné vlákno a následne sekundárne zarovnať nezarovnané časti (bez uvažovania o opačnom vlákne). Miesta, ktoré sa ani po sekundárnom zarovnaní nezarovnali, sme nechali nezarovnané a neskúmali ich na prítomnosť chýb.

Zvýšenie pokrytia zarovnania danými postupmi sme testovali na referencii kvasinky a na sekvencii Miniasm. Výsledky zníženia dĺžky nezarovnaných častí oboch sekvencií sú v tabuľke 4.4.

Pri dlhých čítaniach sme považovali nezarovnanie niektorých častí za zanedbateľné, keďže pokrytie dlhými čítaniami by malo byť dostatočné aj v prípade nezarovnania niektorých čítaní. V prípade záujmu by používateľ mohol na čítania využiť jeden z vyššie spomínaných postupov.

4.4 Analýza pokrytia vzorov

Keďže náš algoritmus funguje na princípe hľadania podsledov (vzorov) zo sekvencií v dlhých čítaniach, rozhodli sme sa pre prvotné pozorovanie vytvoriť histogram pokrytia vzorov. Os x predstavuje počet výskytov vzoru v dlhých čítaniach a os y predstavuje počet vzorov s daným počtom výskytov. Pri správne zostavených sekvenciách očaká-

vame, že najväčšie množstvo vzorov bude mať pokrytie rovné priemernej hĺbke pokrytia genómu. Napravo a naľavo od tohto maxima by mali počty klesať, keďže sekvencia by mala byť čítaniami pokrytá približne rovnomerne. Isté výkyvy môžu byť spôsobené opakovanými sekvenciami. Tie majú za následok zarovnania opakovaných sekvencií z rôznych miest genómu k rovnakému vrcholu grafu. Vzory týchto oblastí sa následne v čítaniach nachádzajú s vyšším pokrytím. Pri nulovej hranici pokrytia však očakávame nulové, resp. veľmi nízke počty vzorov, keďže všetky vzory zo správnej sekvencie by sa mali v aspoň malom počte dlhých čítaní nachádzať. Pri väčšom počte vzorov blízko nuly tieto vzory predstavujú nami pozorované chyby.

Na obrázku 4.1 vidíme histogram pokrytia vzorov referencie E. coli. Z grafu vidíme, že každý vzor referencie bol pokrytý aspoň raz (nulový stĺpec je prázdny) a pri nízkych hodnotách pokrytia vyskytujú iba malé počty vzorov. Z toho môžeme odhadovať, že referencia by podľa nášho kritéria chybovosti mala obsahovať iba minimálne množstvo chýb. Na rozdiel od referencie vidíme na obrázku 4.2 pre zostavenie sekvencie Velvet2 na ľavom kraji v nule hodnotu okolo 30, čo nám hovorí, že 30 vzorov nebolo pokrytých žiadnym čítaním. Taktiež vidíme väčšie počty vzorov a pokrytím blízko nuly. Táto skupina vzorov vyznačuje nami hľadané potenciálne chyby v sekvencii. Pre dáta kvasinky 4.3 vidíme mierne odlišné výsledky. Keďže vieme, že aj samotná referencia má chybové miesta, rozdiel voči zostaveniu Spades pre kvasinku už nie je taký jednoznačný. Aj napriek tomu môžeme pri ľavých krajných hodnotách v jednotlivých histogramoch pozorovať dostatočné rozdiely, čo zodpovedá rozdielnej chybovosti týchto sekvencií.

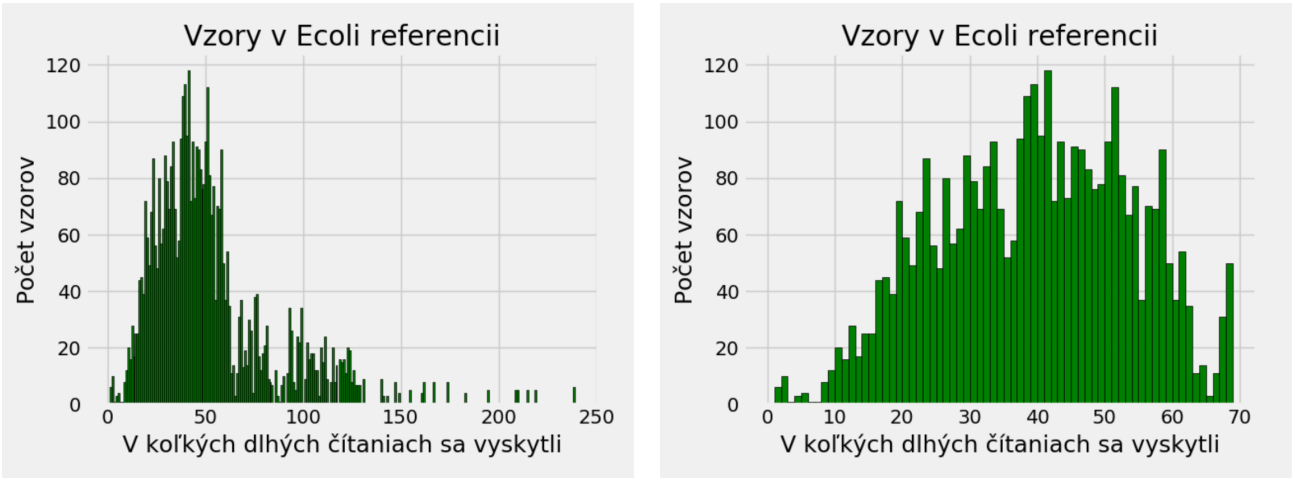
Podľa týchto histogramov je taktiež možné určovať *ch_limit* pre chybovosť vzorov podľa toho, či sú hodnoty blízko nule zvýšené. V našej práci sme testovali vždy 4 verzie limitu, aby sme videli rozdiely vo výsledkoch pri týchto rôznych hodnotách. Pre tieto limity sme naším algoritmom našli chyby a bližšie sme ich pozorovali.

4.5 Analýza výsledkov

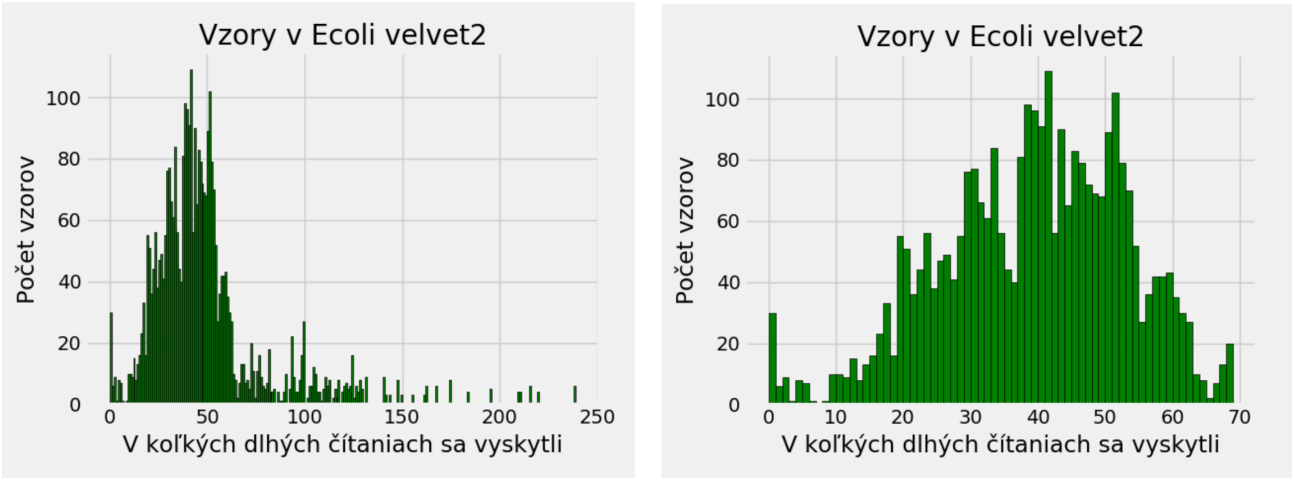
V tejto podkapitole vyhodnotíme výsledky nášho algoritmu. Taktiež sa pozrieme na potenciálne dôvody výskytu nájdených chýb a na možnosti zlepšenia algoritmu.

Prvotné výsledky sme získali postupom spomínaným v kapitole 4.2. Z výsledkov nášho algoritmu sme prekrývajúce sa alebo susedné chyby spojili do jednej chyby, ktorá bola zjednotením týchto chýb. Následne sme určovali presnosť a úplnosť našich výsledkov. Tieto údaje sú v tabuľkách 4.5 a 4.6.

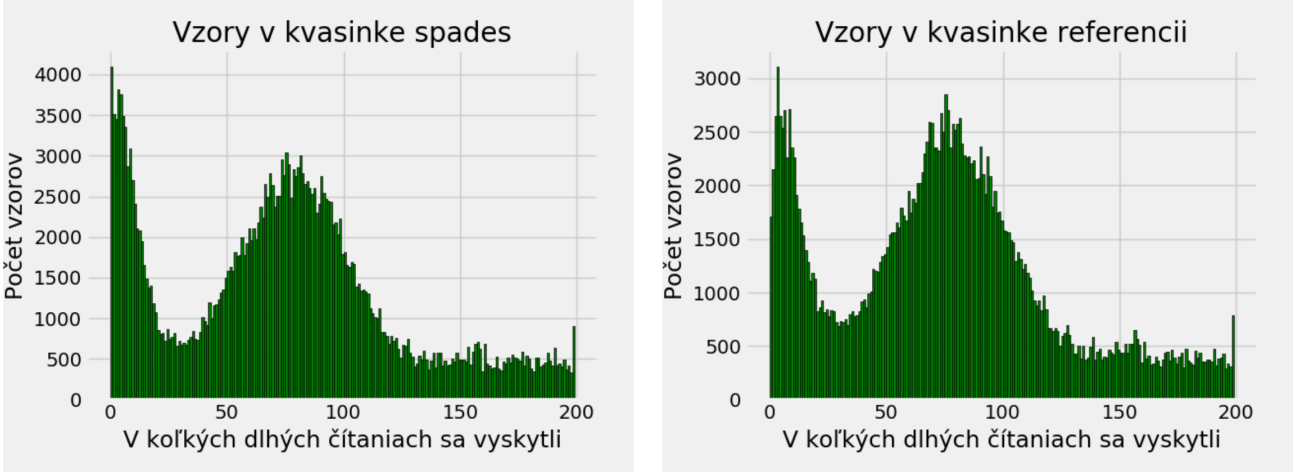
V tabuľkách vidíme veľké rozdiely vo výsledkoch jednotlivých sekvencií. Keďže v E. coli máme iba malé množstvo chýb, je ťažké z nich objektívne určiť presnosť a úplnosť. Ako môžeme v tabuľke vidieť, pri väčších počtoch chýb (Spades2 a Velvet2) je úspešnosť celkovo výrazne vyššia ako v ostatných sekvenciách, kde je počet chýb



Obr. 4.1: Pokrytie vzorov referencie E. coli dlhými čítaniami. Vľavo je úplný histogram pokrytia vzorov. Keďže sa zameriavame najmä na ľavú časť histogramu blízko nuly, vpravo je priblížená verzia histogramu.



Obr. 4.2: Pokrytie vzorov zostavenia Velvet2 pre E. coli dlhými čítaniami. Vľavo je úplný histogram a vpravo priblížená verzia histogramu.



Obr. 4.3: Pokrytie vzorov zostavenia Spades pre kvasinku a referencie kvasinky. Vľavo je priblížená verzia histogramu pre referenciu kvasinky a vpravo priblížená verzia histogramu pre zostavenie Spades pre kvasinku.

Tabuľka 4.5: Presnosť a úplnosť prvých výsledkov kvasinky. V časti tabuľky pre presnosť nám prvý stĺpec sekvencie hovorí o počte správne určených chýb a všetkých nájdených chýb a druhý o percentuálnom vyjadrení presnosti. V časti pre úplnosť hovorí prvý stĺpec o počte nájdených referenčných chýb (čo je počet referenčných chýb pokrytých ľubovoľnou nájdenou chybou) a všetkých referenčných chýb. Druhý stĺpec vyjadruje úplnosť percentuálne. Pre každú sekvenciu máme štyri riadky líšiace sa použitou hodnotou *ch_limit* (počet výskytov vzoru v dlhých čítaniach pre určenie chybovosti).

	Spades		***Miniasm***	
Presnosť				
Limit1	543/837	64.8%	325/480	67.7%
Limit3	1209/2591	46.6%	472/816	57.8%
Limit5	1310/3144	41.6%	593/1062	55.8%
Limit7	1379/3902	35.3%	662/1244	53.3%
Úplnosť				
Limit1	599/1599	37.4%	386/2513	15.3%
Limit3	1298/1599	81.2%	567/2513	22.5%
Limit5	1389/1599	86.8%	692/2513	27.5%
Limit7	1459/1599	91.2%	770/2513	30.6%

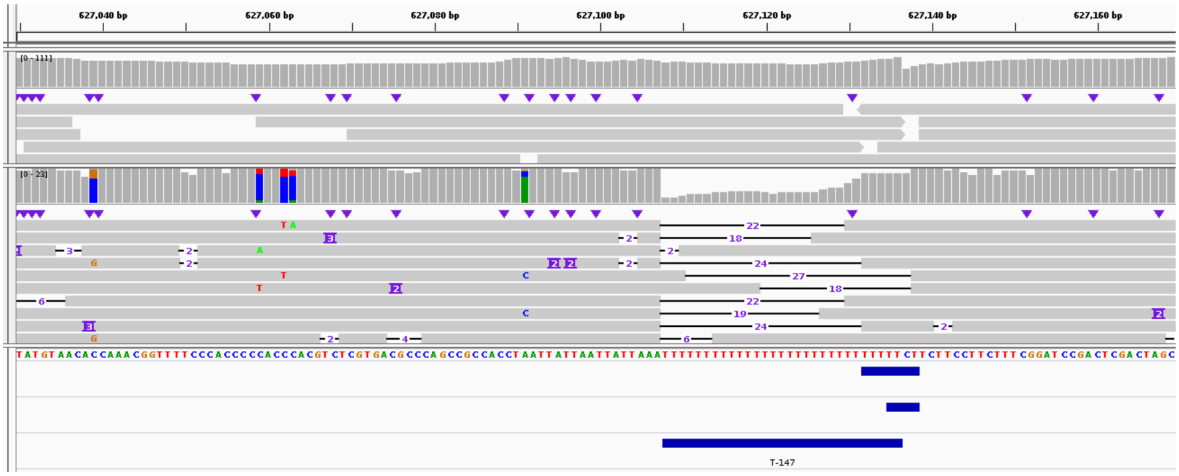
Tabuľka 4.6: Presnosť a úplnosť prvých výsledkov E. coli. Význam stĺpcov je rovnaký ako v tabuľke 4.5.

	Spades1		***Spades2***		***Spades3***		***Velvet***		***Velvet2***	
Presnosť										
Limit1	1/2	50%	1/2	50%	1/2	50%	0/1	0%	2/3	66.6%
Limit3	1/3	33.3%	5/7	71.4%	1/3	33.3%	0/2	0%	2/4	50%
Limit5	1/3	33.3%	5/7	71.4%	1/3	33.3%	0/2	0%	4/6	66.6%
Limit7	1/3	33.3%	5/7	71.4%	1/3	33.3%	0/2	0%	4/7	57.1%
Úplnosť										
Limit1	1/2	50%	1/6	16.6%	1/2	50%	0/2	0%	2/6	33.3%
Limit3	1/2	50%	5/6	83.3%	1/2	50%	0/2	0%	2/6	33.3%
Limit5	1/2	50%	5/6	83.3%	1/2	50%	0/2	0%	4/6	66.6%
Limit7	1/2	50%	5/6	83.3%	1/2	50%	0/2	0%	4/6	66.6%

maximálne 3. Pri výsledkoch kvasinky vidíme vo väčšine prípadov vyššiu presnosť, ktorá so zvyšovaním limitu výrazne klesá. Zároveň pri sekvencii Miniasm môžeme vidieť nízku úplnosť. Rozhodli sme sa preto preskúmať konkrétne chyby a zistiť, čo majú spoločné falošné chyby, ktorých výskyt by sme mohli vyfiltrovať. Sústredili sme sa najmä na skúmanie chýb v zostavení Spades pre kvasinku.

Prvá črta, na ktorú sme narazili, bol problém s homopolymérmi. Homopolymér je sekvencia rovnakých báz opakujúcich sa za sebou. V oblastiach dlhších homopolymérov majú nanopórové čítania problémy. Ich následkom je častý výskyt indelov v dlhých čítaniach v oblastiach homopolymérov. To môže byť dôvodom častého prekryvu oblastí homopolymérov s chybami, keďže sa kvôli indelom tieto čítania nemusia zarovnať ku ideálnym sledom. Príklad takejto oblasti môžeme vidieť na obrázku 4.4. Z dôvodu častého prekryvu chýb s homopolymérmi sme sa rozhodli odstrániť ľubovoľnú chybu, ktorá sa prekrývala s homopolymérom dĺžky aspoň 10. Aj po odstránení týchto chýb sme videli stále veľké množstvo chýb, ktoré boli v bezprostrednej blízkosti homopolyméru. Preto sme sa rozhodli oblasť každého homopolyméru predĺžiť o 5 báz z každej strany a odstrániť všetky chyby, ktoré sa prekrývajú s týmito predĺženými oblasťami. Po týchto úpravách môžeme vidieť výraznú zmenu výsledkov kvasinky v tabuľke 4.7.

Vidíme, že u oboch sekvencií sa naša presnosť zvýšila takmer o 10%. Zároveň sa ale úplnosť výrazne znížila. Naša práca chyby neurčuje presne a slúži skôr ako ukazovateľ možných chybových miest, ktoré je následne potrebné bližšie preskúmať. Z toho dôvodu sa zameriavame viac na presnosť, keďže radšej poukážeme na menej správnych chýb, ktoré je bližším preskúmaním možné preveriť, ako poskytnúť viac správnych ale zároveň aj veľa falošných chýb. Tie by zvyšovali celkový čas potrebný na preskúma-



Obr. 4.4: Príklad oblasti s homopolymérom. V najspodnejšom riadku vidíme modrú oblasť homopolyméru, kde sa v sekvencii opakuje písmeno T. V riadku nad ním je nájdená chyba a v treťom riadku zdola referenčná chyba. Obe chyby sa prekrývajú s homopolymérom. V oblasti sivých obdĺžnikov nižšie (dlhé čítania) vidíme čierne čiary, ktoré znázorňujú indely v dlhých čítaniach, kvôli ktorým vznikajú problémy. Horná oblasť sivých obdĺžnikov sú zarovnané krátke čítania.

Tabuľka 4.7: Presnosť a úplnosť výsledkov kvasinky po odfiltrovaní chýb vo vzdialenosti najviac 5bp od homopolyméru. Dáta su rozdelené rovnako ako v tabuľke 4.5.

	Spades		***Miniasm***	
Presnosť				
Limit1	152/203	74.9%	101/138	73.2%
Limit3	429/643	66.7%	122/189	64.6%
Limit5	491/950	51.7%	151/235	64.3%
Limit7	529/1496	35.3%	173/268	64.5%
Úplnosť				
Limit1	184/661	27.8%	124/1506	8.2%
Limit3	449/661	67.9%	154/1506	10.2%
Limit5	509/661	77%	186/1506	12.4%
Limit7	532/661	80.5%	209/1506	13.9%

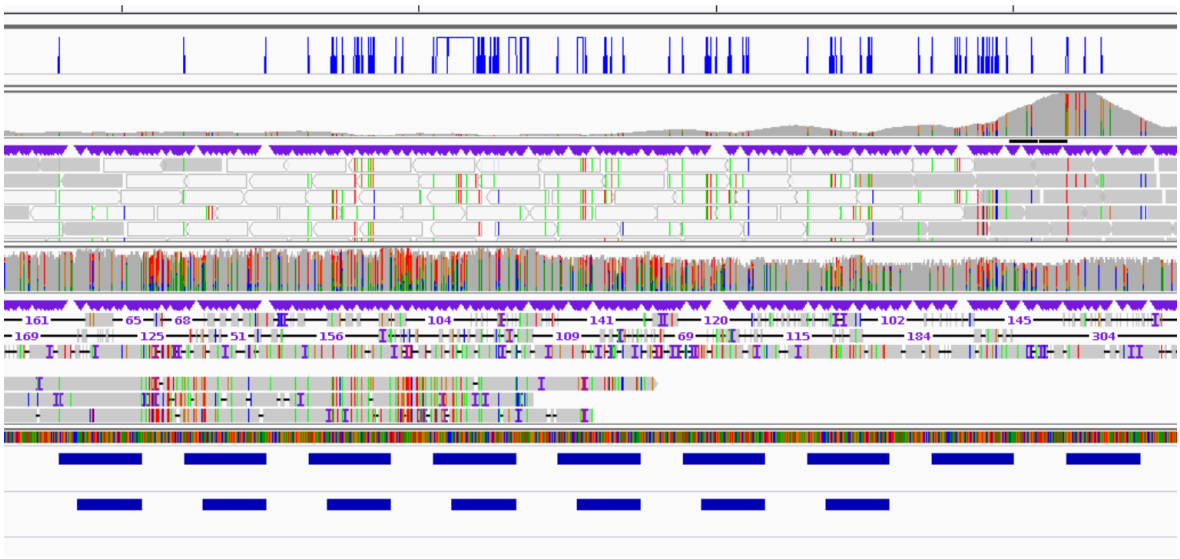
Tabuľka 4.8: Presnosť a úplnosť výsledkov kvasinky po odfiltrovaní dlhých chýb a chýb v blízkosti homopolymérov. Dáta su rozdelené rovnako ako v tabuľke 4.5.

	Spades		***Miniasm***	
Presnosť				
Limit1	153/198	77.3%	101/136	74.3%
Limit3	439/615	71.4%	124/183	67.8%
Limit5	498/883	56.4%	153/227	67.4%
Limit7	544/1418	38.4%	174/263	66.2%
Úplnosť				
Limit1	185/661	28%	124/1506	8.2%
Limit3	459/661	69.4%	157/1506	10.4%
Limit5	515/661	77.9%	190/1506	12.6%
Limit7	546/661	82.5%	211/1506	14%

nie všetkých chýb. Nežiadúcim účinkom tejto filtrácie bolo odstránenie viacero väčších chýb, na ktoré sme sa zameriavali. Väčšina úsekov, ktoré pri zarovnaní zostavenia Spades pre kvasinku ku referencii neboli zarovnané od začiatku kontigu po koniec, ale boli v niektorých miestach predelené (nami hľadané chyby), obsahovali v prípade našich dát homopolymér. Tieto chyby sme odstránili pri filtrácii a teda nemohli skúmať. Tento jav nemusí nastávať pri všetkých genómoch, keďže sa množstvo homopolymérov v jednotlivých genómoch líši.

Po ďalšom skúmaní zostávajúcich chýb sme videli niektoré falošne pozitívne chyby, ktoré boli voči ostatným chybám výrazne dlhšie. Pri príliš dlhých chybách je možné, že presahujú dĺžku dlhých čítaní a z toho dôvodu nie sú pokryté sledmi z týchto čítaní. Ide teda o podsledy, ktoré nie sú nutne chybami, ale nedokážeme ich pomocou dlhých čítaní naším spôsobom preveriť. Preto sme z nájdených chýb najprv odfiltrovali úseky dlhšie ako 3500bp (táto dĺžka je o niečo kratšia ako medián použitých dlhých čítaní) a následne sme zvyšné chyby spojili do neprekrývajúcich sa častí. Nakoniec sme vyfiltrovali chyby prekrývajúce sa s homopolymérmi. Takto sme získali výsledky v tabuľke 4.8.

V tabuľke vidíme celkový pokles počtu nájdených chýb, pričom počet referenčných chýb sa nezmenil. Niektoré chyby boli nahradené kratšou chybou. Boli to chyby, ktoré sa prekrývali s dlhšou chybou a v pôvodnom postupe sa s nimi spojili. Keď sme dlhé chyby hneď na začiatku odfiltrovali, tieto kratšie chyby tam zostali namiesto dlhých. Tento jav spôsobil spolu so znížením celkového počtu nájdených chýb zvýšenie počtu správne určených chýb. Prvý dôvod je nahradenie dlhých referenčných chýb kratšími, ktoré následne neboli odfiltrované kvôli prekryvu s homopolymérmi a zároveň pokrývali nejakú nájdenú chybu. Tá pôvodne nebola správna, keďže dlhý úsek, ktorý ju



Obr. 4.5: Príklad správne nájdených chýb v zostavení Spades pre kvasinku. V spodnom riadku vidíme modré oblasti, ktoré znázorňujú nájdené chyby. Nad nimi máme referenčné chyby. Vidíme, že nie všetky referenčné chyby sa nám podarilo nájsť. Nad referenčnými chybami vidíme oblasť sivých obdĺžnikov (dlhých čítaní), v ktorých farebný prúžok znamená rozdielnu bázu od sekvencie. Nad dlhými čítaniami máme krátke čítania, ktoré taktiež obsahujú farebné prúžky, pri rozdiele báz od sekvencie. V najvrchnejšom riadku sú výsledky softvéru Pilon, pričom nepotvrdené bázy (potenciálne chyby) sú označené modrou.

pokrýval sme odfiltrovali kvôli homopolyméru. Druhý dôvod je podobný, avšak nastal pri nájdených chybách. Z väčšej chyby vznikla menšia, ktorá sa neodfiltrovala kvôli homopolyméru a pritom bola správna. Týmto spôsobom sme získali doteraz najlepšie výsledky.

Pri sekvenciách E. coli bol výskyt homopolymérov výrazne nižší. Preto sa odfiltrovaním chýb prekrývajúcich sa s homopolymérmi výsledky takmer vôbec nezmenili. Jednu takúto chybu (nájdenú a jej prislúchajúcu referenčnú) sa nám podarilo odfiltrovať v sekvencii Spades2 a jednu v Spades3. Pri filtrácii chýb dlhších ako 3500bp sa nám v každej sekvencii podarilo odfiltrovať jednu falošnú chybu, čo je pri daných počtoch chýb výrazné zlepšenie.

Po skúmaní zostávajúcich chýb sme nenašli ďalšie kritériá, podľa ktorých by bolo možné chyby jednoduchým spôsobom vyfiltrovať a tým zvýšiť presnosť algoritmu. Na obrázku 4.5 vidíme ukážkový priklad zostávajúcich správne nájdených chýb. Nachádza sa tam sedem skutočne pozitívnych chýb a dve falošne negatívne chyby. Vidíme, že tieto nami nájdené chyby sú v oblastiach, v ktorých dlhé aj krátke čítania poukazujú na možné problémy v sekvencii a zároveň ich aj softvér Pilon označuje za potenciálne chybné.

Tabuľka 4.9: Porovnanie výsledkov so softvérom Pilon. V tabuľke vidíme percento báz, ktoré boli označené za potenciálne chybné softvérom Pilon. Pri celej sekvencii je to pomer označených chybných báz v celej sekvencii ku celkovému počtu báz zostavenia sekvencie. Pri ďalšíc hriadkoch je to pomer počtu báz v týchto chybách označených softvérom Pilon ku súčtu dĺžok chýb určených pri danom limite.

	Spades	***Miniasm***
Celá sekvencia	0.39%	1%
Referenčné chyby	3.7%	38%
Limit1	5%	19.6%
Limit3	2.4%	8.4%
Limit5	2.5%	7.5%
Limit7	2.4%	8.1%

Pri finálne určených chybách sme pri kvasinke dodatočne sledovali, či sú výsledky v istej miere podobné výsledkom softvéru Pilon. Zo softvéru Pilon sme si vybrali určenie nepotvrdených báz (potenciálne chybné bázy). V našej práci sa zameriavame skôr na väčšie úseky, a preto nie je možné priame porovnávanie chýb určených naším softvérom a softvérom Pilon. Preto sme sledovali percento nepotvrdených báz v celom genóme a následne percento nepotvrdených báz v nami určených chybách. Výsledky tejto analýzy sa nachádzajú v tabuľke 4.9. Môžeme vidieť, že oproti percentu nepotvrdených báz v referenciách sú percentá u našich chýb výrazne vyššie, najmä pri limite 0 (referenčné chyby) a limite 1, pri ktorom sme dosahovali najvyššiu presnosť algoritmu. V prípade náhodne vybraných oblastí z genómu očakávame, že percento nepotvrdených báz bude podobné percentu nepotvrdených báz celého genómu. Výrazne zvýšenie percenta nepotvrdených báz v našich chybách preto hovorí o tom, že naše oblasti súvisia s chybami, ktoré určil softvér Pilon. To do istej miery podporuje relevantnosť našich výsledkov.

Záver

Cieľom tejto práce bolo priniesť nový pohľad na problematiku hľadania chýb v zostavených genómoch. Základom našej metódy bolo využitie reťazcových grafov a dlhých čítaní. Pomocou podobností zarovnaní sekvencie a dlhých čítaní sme v sekvencii hľadali chyby.

Najprv sme využili existujúci softvér, pomocou ktorého sme vytvorili kompaktovaný reťazcový graf z krátkych čítaní. K tomuto grafu sme následne zarovnali skúmané sekvencie a dlhé čítania. Takto sme získali sledy vrcholov so znamienkami, ktoré reprezentovali zarovnanú sekvenciu a dlhé čítania. Zo sledov zarovnania sekvencie sme následne vytvárali podsledy istých dĺžok. Pre každý podsled sme následne skúmali, či sa nachádzajú aj v sledoch dlhých čítaní v dostatočnom počte. V prípade, že vzor nebol dostatočne veľakrát zastúpený v dlhých čítaniach, označili sme ho za chybu.

Takto nájdené chyby sme následne bližšie skúmali a ďalej filtrovali. Prvým druhom filtrácie bolo odstránenie chýb, ktoré sa prekrývali s homopolymérmi. Ďalším druhom filtrácie bolo odstránenie chýb dlhších ako 3500bp, ktoré sú príliš dlhé na to, aby boli otestované pomocou našej metódy. Na záver sme naše výsledky porovnali s výsledkami softvéru Pilon.

Metóda, ktorú sme navrhli, by sa dala ďalej vylepšovať a rozširovať pre získavanie lepších výsledkov. Prvým možným miestom na zlepšenie by bolo bližšie preskúmanie možností softvéru GraphAligner na spresnenie zarovnaní ku grafu, ktoré výrazne ovplyvňujú výsledky našej práce. Druhou možnosťou by bola zložitejšia filtrácia homopolymérov. Keďže pri momentálnej metóde je možné odstránenie správne určených chýb, bolo by lepšie dané časti s homopolymérmi odfiltrovať ešte pred určovaním chýb. Ďalšou možnosťou by bolo bližšie skúmanie finálnych chýb a nájdenie ďalších možností ich filtrácie.

Literatúra

[1] BED format. https://genome.ucsc.edu/FAQ/FAQformat.html#format1. Accessed: 2020-04-20.

[2] GFA format. http://gfa-spec.github.io/GFA-spec/GFA1.html. Accessed: 2020-04-20.

[3] JSON format. https://www.json.org/json-en.html. Accessed: 2020-05-16.

[4] Saprochaete fungicola genome sequencing. https://www.ebi.ac.uk/ena/data/view/PRJEB30635. Accessed: 2020-05-28.

[5] Joel Armstrong, Ian T Fiddes, Mark Diekhans, and Benedict Paten. Whole-genome alignment and comparative annotation. *Annual review of animal biosciences*, 7:41–64, 2019.

[6] Anton Bankevich, Sergey Nurk, Dmitry Antipov, Alexey A Gurevich, Mikhail Dvorkin, Alexander S Kulikov, Valery M Lesin, Sergey I Nikolenko, Son Pham, Andrey D Prjibelski, et al. SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing. *Journal of computational biology*, 19(5):455–477, 2012.

[7] Daniel Branton, David W Deamer, Andre Marziali, Hagan Bayley, Steven A Benner, Thomas Butler, Massimiliano Di Ventra, Slaven Garaj, Andrew Hibbs, Xiaohua Huang, et al. The potential and challenges of nanopore sequencing. In *Nanoscience and technology: A collection of reviews from Nature Journals*, pages 261–268. World Scientific, 2010.

[8] Broňa Brejová, Hana Lichancová, Viktória Hodorová, Martina Neboháčová, L'ubomír Tomáška, Tomáš Vinař, and Jozef Nosek. Genome Sequence of an Arthroconidial Yeast, Saprochaete fungicola CBS 625.85. *Microbiol Resour Announc*, 8(15):e00092–19, 2019.

[9] Rayan Chikhi, Antoine Limasset, and Paul Medvedev. Compacting de Bruijn graphs from sequencing data quickly and in low memory. *Bioinformatics*, 32(12):i201–i208, 06 2016.

[10] Peter JA Cock, Christopher J Fields, Naohisa Goto, Michael L Heuer, and Peter M Rice. The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic acids research*, 38(6):1767–1771, 2010.

[11] Phillip EC Compeau, Pavel A Pevzner, and Glenn Tesler. How to apply de Bruijn graphs to genome assembly. *Nature biotechnology*, 29(11):987, 2011.

[12] Fatima Cvrčková. *Úvod do praktické bioinformatiky*. Academia, 2006.

[13] John W Davey, Seth J Davis, Jeremy C Mottram, and Peter D Ashton. Tapestry: validate and edit small eukaryotic genome assemblies with long reads. *BioRxiv*, 2020.

[14] Sarah E DeWeerdt. What's a Genome. `http://www.genomenewsnetwork.org/resources/whats_a_genome/Chp1_1_1.shtml`. Accessed: 2020-04-20.

[15] Sara Goodwin, John D McPherson, and W Richard McCombie. Coming of age: ten years of next-generation sequencing technologies. *Nature Reviews Genetics*, 17(6):333, 2016.

[16] Cyril Goutte and Eric Gaussier. A probabilistic interpretation of precision, recall and F-score, with implication for evaluation. In *European Conference on Information Retrieval*, pages 345–359. Springer, 2005.

[17] Martin Hunt, Taisei Kikuchi, Mandy Sanders, Chris Newbold, Matthew Berriman, and Thomas D Otto. REAPR: a universal tool for genome assembly evaluation. *Genome biology*, 14(5):R47, 2013.

[18] Inc. Illumina. Illumina technology. `https://www.illumina.com/science/technology.html`. Accessed: 2020-04-20.

[19] Miten Jain, Sergey Koren, Karen H Miga, Josh Quick, Arthur C Rand, Thomas A Sasani, John R Tyson, Andrew D Beggs, Alexander T Dilthey, Ian T Fiddes, et al. Nanopore sequencing and assembly of a human genome with ultra-long reads. *Nature biotechnology*, 36(4):338, 2018.

[20] Thomas Laver, J Harrison, PA O'neill, Karen Moore, Audrey Farbos, Konrad Paszkiewicz, and David J Studholme. Assessing the performance of the oxford nanopore technologies minion. *Biomolecular detection and quantification*, 3:1–8, 2015.

[21] Heng Li. Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *Bioinformatics*, 32(14):2103–2110, 2016.

[22] Elaine R Mardis. Next-generation DNA sequencing methods. *Annu. Rev. Genomics Hum. Genet.*, 9:387–402, 2008.

[23] Pierre Marijon, Rayan Chikhi, and Jean-Stéphane Varré. Graph analysis of fragmented long-read bacterial genome assemblies. *Bioinformatics*, 35(21):4239–4246, 2019.

[24] U.S. National Library of Medicine National Center for Biotechnology Information. FASTA format. `https://blast.ncbi.nlm.nih.gov/Blast.cgi?CMD=Web&PAGE_TYPE=BlastDocs&DOC_TYPE=BlastHelp`. Accessed: 2020-04-20.

[25] Adam Novak. vg.proto format. `https://github.com/vgteam/libvgio/blob/master/deps/vg.proto`. Accessed: 2020-04-20.

[26] N Patrick Higgins. Chromosome Structure. *e LS*, 2001.

[27] Aaron R Quinlan. BEDTools: the Swiss-army tool for genome feature analysis. *Current protocols in bioinformatics*, 47(1):11–12, 2014.

[28] Mikko Rautiainen. Graphaligner. `https://github.com/maickrau/GraphAligner`. Accessed: 2020-04-20.

[29] Mikko Rautiainen, Veli Mäkinen, and Tobias Marschall. Bit-parallel sequence-to-graph alignment. *Bioinformatics*, 35(19):3599–3607, 2019.

[30] Jared T Simpson and Mihai Pop. The theory and practice of genome sequence assembly. *Annual review of genomics and human genetics*, 16:153–172, 2015.

[31] Li Song, Liliana Florea, and Ben Langmead. Lighter: fast and memory-efficient sequencing error correction without counting. *Genome biology*, 15(11):509, 2014.

[32] Wing-Kin Sung. *Algorithms for next-generation sequencing*. Chapman and Hall/CRC, 2017.

[33] Robert Vaser, Ivan Sović, Niranjan Nagarajan, and Mile Šikić. Fast and accurate de novo genome assembly from long uncorrected reads. *Genome research*, 27(5):737–746, 2017.

[34] Bruce J Walker, Thomas Abeel, Terrance Shea, Margaret Priest, Amr Abouelliel, Sharadha Sakthikumar, Christina A Cuomo, Qiandong Zeng, Jennifer Wortman, Sarah K Young, et al. Pilon: an integrated tool for comprehensive microbial variant detection and genome assembly improvement.

[35] Daniel R Zerbino and Ewan Birney. Velvet: algorithms for de novo short read assembly using de Bruijn graphs. *Genome research*, 18(5):821–829, 2008.

Prílohy

Ukážkove dáta a hlavné zdrojové kódy skriptov, využívaných v našej práci sú priložené na SD karte.