Univerzita Komenského v Bratislave

Fakulta matematiky, fyziky a informatiky

Zarovnávanie sekvenovacích dát s vysokým počtom chýb

bakalárska práca

2015 Marcel Schichman

Univerzita Komenského v Bratislave

Fakulta matematiky, fyziky a informatiky

Zarovnávanie sekvenovacích dát s vysokým počtom chýb

bakalárska práca

Študijný program: Informatika
Študijný odbor: 2508 Informatika
Školiace pracovisko: Katedra informatiky
Školiteľ: Mgr. Vladimír Boža

Bratislava, 2015 Marcel Schichman

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Marcel Schichman
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: 9.2.1. informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Zarovnávanie sekvenovacích dát s vysokým počtom chýb
Noisy reads alignment

Cieľ: Najnovšie sekvenovacie technológie produkujú dlhé sekvencie DNA, ktoré majú ale vysoký počet chýb. Veľmi často je potrebné tieto dáta zarovnávať k referenčnému genómu. Cieľom práce je urobiť prehľad ideí a algoritmov, ktoré sa používajú pri zarovnávaní sekvencií s vysokým počtom chýb a následne implementovať a porovnať tieto prístupy.

Vedúci: Mgr. Vladimír Boža
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, PhD.

Dátum zadania: 21.10.2014

Dátum schválenia: 28.10.2014 doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Abstrakt

Cieľom tejto práce bolo vytvoriť nástroj na mapovanie čítaní sekvenovacieho zariadenia PacBio RS II na referenčný genóm. Tieto čítania sa sa vyznačujú dĺžkou, avšak chybovosť dosahuje až 14%. Výslednú aplikáciu dostupnú na adrese `https://github.com/marcelschichman/bakalarka` sme porovnávali s oficiánlym nástrojom na tento účel s názvom BLASR. Rýchlosťou náš nástroj niekoľko krát prekonal BLASR, ale bol menej úspešný zarovnávať čítania na genóm ako jeden celok. Príčinou nezarovnaných úsekov bol rozdiel medzi referenčným genómom a genómom živočícha, z ktorého boli získavané čítania.

Kľúčové slová: mapovanie sekvencii, PacBio, dynamické programovanie, heuristiky

Abstract

The main goal of this thesis was to create a tool for mapping PacBio reads to a reference genome. These reads are significantly longer than read acquired by standard methods, but have high error rate – around 14%. Our tool is available at `https://github.com/marcelschichman/bakalarka`. We performed several tests and compared it to official mapping tool BLASR. Our tool is much faster, but it often splits an alignment into two. This problem is caused by differences between reference genome and genome of the organism used for sequencing.

Keywords: sequence mapping, PacBio, dynamic programming, heuristics

Obsah

Úvod

V tejto práci sa budeme zaoberať zarovnávaním DNA sekvencií. Našim cieľom je vytvoriť nástroj na mapovanie sekvenačných dát na referenčný genóm. Tento nástroj má byť prispôsobený na dáta získané sekvenovacím zariadením PacBio RS II. Metóda získavania DNA v tomto zariadení umožňuje čítať naraz výrazne dlhšie úseky na úkor početnosti chýb. Tá sa pohybuje na úrovni 14%. Dovodom na vývoj nového nástroja je fakt, že existujúci nástroj BLASR pracuje zdanlivo príliš pomaly.

V prvej kapitole vysvetlíme základné pojmy a zadefinujeme súvisiace problémy. V kapitole implementácia popíšeme, ako prebieha mapovanie v našom nástroji a ako sme v ňom využili algoritmus na zarovnávanie [8] Gene Myersa. Výsledky experimentov a porovnanie nášho nástroja s oficiálnym nástrojom na mapovanie spomenutých sekvenačných dát sme spísali do kapitoly experimenty.

Kapitola 1

Zarovnávanie sekvencií

V tejto kapitole všeobecne popíšeme problém zarovnávania, základný prístup na riešenie a prípadné heuristiky na jeho zlepšenie.

DNA sekvencia je reťazec znakov A, C, G, T. Zarovnávanie sekvencií je určenie vzájomnej podobnosti medzi dvoma sekvenciami. Je zložené z dvoch riadkov, pričom riadky zodpovedajú vstupným sekvenciam. V zarovnaní sa do sekvencií vkladajú pomlčky tak, aby malo v stĺpcoch čo najviac rovnakých báz a aby boli riadky rovnakej dĺžky. Na určenie, ktoré zarovnanie je najlepšie, potrebujeme zadefinovať skórovací systém. Skórovací systém priradzuje dvojiciam báz zo zarovnania body podľa toho, či ide o zhodu, substitúdiu, deléciu alebo inzerciu. Niektoré algoritmy rozlišujú aj jednotlivé zhody a substitúcie a priradzujú im rôzne hodnoty.

```
CTTCAGT--AGATTG-CC
-TT-AATCTAGA-TGACC
```

Obr. 1.1: Zarovnanie dvoch DNA sekvencií

1.1 Problém globálneho a lokálneho zarovnania

Hľadanie najlepšieho zarovnania dvoch celých sekvencií sa nazýva problém globálneho zarovnania. Niekedy potrebujeme nájsť najlepšie zarovnanie podreťazcov dvoch sekvencií. Tento problém sa volá problém lokálneho zarovnania. Potreba riešiť tento problém nastáva napríklad pri rekonštrukcii genómu na základe prekrývajúcich sa čítaní zo sekvenovacích zariadení. V tejto práci sa budeme prevažne zaoberať mapovaním, t.j. zarovnávaním celej sekvencie na časť referenčného genómu. Všetky tri problémy spolu úzko súvisia a preto sa na ne dajú uplatňovať podobné princípy.

1.1.1 Needleman–Wunsch

Needlemanov a Wunschov algoritmus slúži na hľadanie najlepšieho globálneho zarovnania medzi dvomi sekvenciami. Využíva sa v ňom princíp dynamického programovania. Na vstupe dostaneme dva reťazce A, B nad abecedou {A, C, T, G} dlhé n, m znakov a výstupom má byť zarovnanie celých reťazcov s najlepším skóre. Pre jednoduchosť zadefinujeme skóre tak, že zhoda má hodnotu +1 a substitúcia, inzercia a delécia majú hodnotu –1. Potrebujeme dvojrozmerné pole M s rozmermi $n + 1$, $m + 1$. Na pozícii $[i, j]$ je skóre najlepšieho zarovnania prefixov A, B s dĺžkami i a j.

V tabuľke najprv vyplníme prvý riadok a prvý stlpec. $M[0, 0]$ je 0 (je to zarovnanie dvoch prázdnych reťazcov). Pre ostatné políčka platí, že práve jedna z ich súradníc je nula. Ich najlepšie zarovnanie je toľko substitúcií (resp. delécií), aká je hodnota druhej súradnice. Preto $M[0, i]$ a $M[i, 0]$ má hodnotu $-i$.

Teraz ukážeme, ako vypočítať hodnoty $M[i, j]$ za predpokladu, že poznáme hodnoty políčok s menšími súradnicami. Najlepšie zarovnanie na prefixoch dĺžok i, j končí buď dvojicou ($A[i]$, $B[j]$), ($A[i]$, –) alebo (–, $B[j]$). Ak by končilo dvojicou ($A[i]$, $B[j]$), tak zvyšok zarovnania je zhodný so zarovnaním na prefixoch dĺžok i–1, j–1. Podľa rovnosti $A[i]$ a $B[j]$ je $M[i, j]$ buď

		C	T	T	C	A	G	T	A	G	A
	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10
T	-1	-1	0	-1	-2	-3	-4	-5	-6	-7	-8
T	-2	-2	0	1	0	-1	-2	-3	-4	-5	-6
A	-3	-3	-1	0	0	1	0	-1	-2	-3	-4
A	-4	-4	-2	-1	-1	1	0	-1	0	-1	-2
T	-5	-5	-3	-1	-2	0	0	1	0	-1	-2
C	-6	-4	-4	-2	0	-1	-1	0	0	-1	-2
T	-7	-5	-3	-3	-1	-1	-2	0	-1	-1	-2
A	-8	-6	-4	-4	-2	0	-1	-1	1	0	0
G	-9	-7	-5	-5	-3	-1	1	0	0	2	1
A	-10	-8	-6	-6	-4	-2	0	0	1	1	3

Tabuľka 1.1: Ukážka algoritmu na krátkych sekvenciách

$M[i–1, j–1] + 1$ alebo $M[i–1, j–1]–1$. Ak by končilo dvojicou $(A[i], –)$, tak zvyšok zarovnania je zhodný so zarovnaním na prefixoch dĺžok $i–1$, j. Hodnota $M[i, j]$ je preto $M[i–1, j]–1$. Obdobne vieme vypočítať hodnotu $M[i, j]$ v prípade, že by zarovnanie končilo dvojicou $(–, B[j])$. Vieme, že nastane jedna z týchto možností a keďže ide o najlepšie zarovnanie, tak vyberieme najlepšiu z nich. Inak povedané – hodnota $M[i, j]$ je maximum z hodnôt:

- $M[i-1, j] - 1$
- $M[i, j-1] - 1$
- $M[i-1, j-1] + 1$ ak $A[i] = B[j]$ inak $M[i-1, j-1] - 1$

Pole M teraz môžeme celé vyplniť po riadkoch alebo stĺpcoch. Hodnota $M[n, m]$ je skóre najlepšieho zarovnania sekvencií A a B. Každú hodnotu v poli sme vypočítali v konštantnom čase. Preto časová zložitosť algoritmu je $O(nm)$. Spätným chodom vieme zrekonštruovať zarovnanie s príslušným skóre.

Malou úpravou tohto algoritmu vieme počítať aj lokálne zarovnania. Do poľa by sme namiesto záporných hodnôt ukladali nuly. Najlepšie lokálne zarovnanie v takto vytvorenom poli nájdeme tak, že nájdeme najvačšiu hodnotu v celej matici. Z tohto políčka spätne rekonštrujeme zarovnanie ako v pôvodnom algoritme, ale zastavíme sa na prvej nule. Tento algoritmus sa volá Smithov a Watermanov [10].

1.1.2 Heuristiky

Dĺžky sekvencií sú často príliš dlhé na to aby sa dal použiť uvedený algoritmus. Výrazné zrýchlenie sa dá dosiahnuť použitím heuristických metód. Pomocou nich sa dajú obmedziť miesta v poli Needlemanovho a Wunschovho algoritmu.

Zvolíme si parameter W a nájdeme presné zhody sekvencii s dĺžkou W. Hodnotu parametra treba zvoliť tak, aby sme nenašli príliš veľa falošných dvojíc. Tieto dvojice sa nazývajú jadrá zarovnania. Tieto jadrá ešte skúsime rozšíriť, ak sa sekvencie zhodujú aj na susedných bázach.

Dynamickým programovaním potom stačí vypĺňať len maticu medzi koncami dvoch blízkych jadier, ktoré ležia približne na jednej diagonále.

1.2 Mapovanie sekvenačných dát

Sekvenačné dáta sa získavajú procesom sekvenovania roznymi metódami a zariadeniami priamo z DNA buniek. Naraz je možné prečitať len pomerne malú časť genómu organizmu. Na získanie celého genómu je potrebné ho pokryť čítaniami niekoľko krát, aby bolo možné nájsť prekrytia a tým určiť ich vzájomnú polohu. V súčasnosti sa už podarilo prečítať genómy mnohých organizmov vrátane človeka.

Jedným z pomocných problémov v bioinformatike je zistiť, odkiaľ vrámci vopred získaného genómu pochádzajú sekvencie získané zo sekvenovacieho zariadenia. Tento problém sa nazýva mapovanie. Pre jedno čítanie je jeho výstupom:

- oriencátcia; prečítana sekvencia sa môže v genóme nachádzať buď priamo, alebo sa v genóme nachádza jej reverzný komplement
- začiatočná a koncová pozícia na genóme a čítaní (niekedy sa podarí zarovnať len časť čítania)
- zarovnanie časti čítania na zodpovedajúcu časť genómu

Môže sa stať, že sa niektoré čítania alebo ich časti podarí zarovnať na viaceré miesta na genóme.

1.3 Zarovnávanie PacBio sekvencií

V súčasnosti moderné sekvenovacie zariadenie od výrobcu Pacific Biosciences [6] produkuje sekvencie dlhé rádovo 10 tisíc báz. Nevýhodou je pomerne vysoký podiel chýb – priblíže 14%. To znemožnuje použitie štandardných mapovacích nástrojov.

V tejto kapitole popíšeme heuristické metódy nástrojov určených práve na zarovnávanie sekvenovacích dát s vysokým počtom chýb.

1.3.1 BLASR

Nástroj BLASR (Basic Local Alignment with Successive Refinement) slúži na mapovanie sekvencie na referenčný genóm. Mapovanie v nástroji BLASR prebieha v troch fázach. V prvej sa nájdu dvojice presných zhôd minimálnej požadovanej dĺžky. Tieto takzvané jadrá zarovnania sú na základe vzájomnej vzdialenosti roztridené do skupín a ohodnotené. Pre skupiny s dostatočným skóre je riedkym dynamickým programovaním vypočítané približné zarovnanie prechádzajúce cez jadrá v danej skupine. Ak aj toto približné zarovnanie spĺňa podmienky, je celý úsek zarovnaný ešte raz, tentokrát už presným dynamickým programovaním.

1.3.2 DALIGN

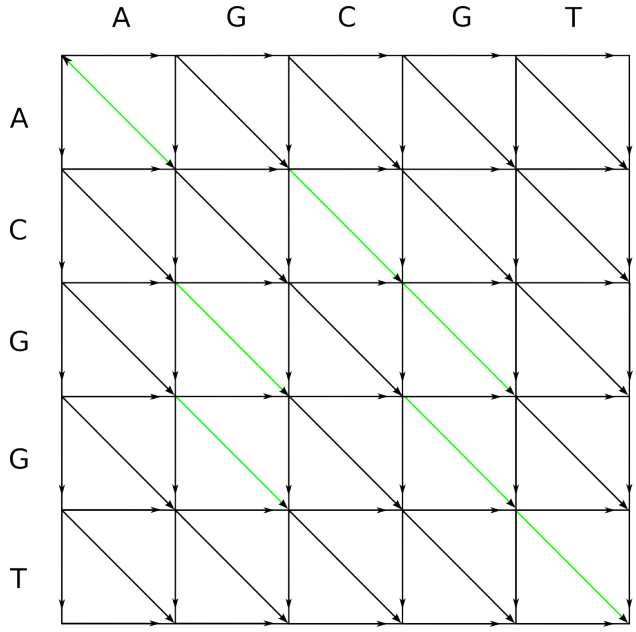
Nástroj DALIGN [2] hľadá lokálne zarovnania medzi dvomi množinami sekvencií. Využíva heuristiky na hľadanie jadier zarovnania a tiež na nájdenie celého lokálneho zarovnania z jadra.

Rapid Seed Detection Máme dve sekvencie A, B dĺžok N. Z nich vytvoríme dva zoznamy súvislých k-tic spolu s ich pozíciou v rámci sekvencie. Tieto zoznamy lexikograficky usporiadame. Z usporiadaných zoznamov vieme postupným prechádzaním oboch zoznamov naraz vytvoriť zoznam jadier zarovnaní. Vytváranie zoznamov k-tic má lineárnu časovú zložitosť od N. Ich následné usporiadanie je v $O(NlogN)$. Hľadanie jadier časovo závisí od ich počtu.

Rapid Local Alignment Discovery Po nájdení jadier ich chceme rozšíriť v oboch smeroch po diagonále na čo najväčšie zarovnania. Najprv zadefinujeme pojem editačný graf. Pre dve sekvencie A, B dĺžok M, N je to graf s vrcholmi (i, j) z množiny $[0, M] \times [0, N]$. Graf má nasledovné hrany:

- $(i-1, j) \longrightarrow (i, j)$ pre $i > 0$ s dĺžkou 1 (delécia)
- $(i, j-1) \longrightarrow (i, j)$ pre $j > 0$ s dĺžkou 1 (inzercia)
- $(i-1, j-1) \longrightarrow (i, j)$ pre $i, j > 0$ s dĺžkou 0 ak $A[i] = B[j]$, inak 1 (zhoda/substitúcia)

Na obrázku 1.2 je znázornený editačný graf pre sekvencie $ACGGT$ a $AGCGT$. Čierne cesty majú dĺžku 1 a zelené 0. Dá sa nahliadnuť, že cesta v grafe jednosnačne definuje jedno zarovnanie a pre každé zarovnanie na sekvenciách A, B existuje cesta v ich editačnom grafe. Zarovnávací algoritmus na tomto grafe postupne hľada množiny najďalej siahajúcich bodov postupne pre vzdialenosti 1, 2, ... začinajúc v danom jadre zarovnania. So stúpajúcou vzdialenosťou počet týchto bodov lineárne narastá. Niektorý z nich patrí



Obr. 1.2: Editačný graf

zarovnaniu ku ktorému výpočet dospeje, ale pre väčšinu vieme s veľkou pravepodobnosťou povedať, že do zarovnania nepatria. V článku sú uvedené dve stratégie orezávania najďalej siahajúcich bodov, ktoré sú vo výslednom zarovnaní s extrémne nízkou pravdepodobnosťou.

Prvá sa opiera of fakt, že zarovnanie by malo mať na každom úseku určenej dĺžky aspoň nejakú rozumnú koreláciu. Pre každý najďalej siahajúci bod si preto postupne dopočítavame počet zhôd za posledných C hrán na ceste z jadra do daného bodu. To sa dá efektívne implementovať tak, že si v bitovom vektore ukládáme, ktoré z posledných C hrán cesty boli zhodou (1) a ktoré substitúciu/deléciou/inzerciou (0). Počet zhôd potom prepočítame na základe poslednej hrany na ceste a hodnoty vo vektore na pozícii C. Po každom kroku sa bitový vektor posunie, na začiatok sa pridá nová hodnota a oreže, aby mal požadovanú dĺžku C.

Druhá stratégia je uchovávať len tie body, ktoré sú v rozsahu L antidiagonál od maximálnej dosiahnutej antidiagonály. Body blízko optimálneho

zarovnania mali na ceste od jadra viac zhôd a preto ležia na výrazne vzdialenejších antidiagonálach ako body mimo od optimálneho zarovnania. Tento fakt spôsobuje, že jedna vlna najďalej siahajúcich bodov má tvar hrotu šípu.

Algoritmus skončí, ak dojde na okraj editačného grafu alebo ak už žiaden najďalej siahajúci bod nespĺňa podmienku lokálnej kvality zarovnania. Orezávacie stratégie robia tento algoritmus heuristickým a len neformálne uvedieme, že má lineárnu časovú zložitosť závislú od dĺžky výsledného zarovnania.

Kapitola 2

Implementácia

Náš cieľ je implementovať nástroj na mapovanie PacBio čítaní na referenčný genóm. Implementujeme ho v jazyku C++. Problém mapovania rozdelíme na hľadanie jadier zarovnaní a ich následné rozšírenie do zarovnaní. V tejto kapitoje popíšeme ako sme postupovali a ktoré zlepšenia pomohli ušetriť čas alebo zvýšiť efektivitu.

2.1 Hľadanie jadier zarovnania

Za predpokladu, že sa dané čítanie nachádza v genóme s pravdepodobnosťou chyby p, je zvolený úsek dĺžky L na čítaní zhodný s jeho prislúchajúcim úsekom na genóme s pravdepodobnosťou $(1-p)^L$. Uvažujme chybu 20% a dĺžku jadier 20. Pravdepodobnosť, že sa zvolený úsek zhoduje s jeho obrazom, je v tomto prípade približne $1,15\%$. Stredná hodnota počtu zhôd na a čítaní dĺžky 10000 je preto približne 115. Tieto úseky sa možu prekrývať a vytvárať tak dlhšie jadrá zarovnania. Úvahy spomenuté v tomto odstavci samozrejme platia len za predpokladu, že rozdiely medzi genómom a čítaním sú rozmiestnené náhodne, ale to by pri čítaniach získaných pomocou sekvenovacieho zariadenia PacBio RS 2 podľa jeho výrobcu platiť.

2.1.1 Jadrá pevnej dĺžky

Existuje viacero spôsobov ako hľadať jadrá zvolenej dĺžky. Štandarne sa to rieši vytvorením indexu z genómu, vďaka ktorému vieme rýchlo zistiť, kde sa v ňom nachádza nejaká postupnosť báz. V našej implementácii sme použili hešovaciu tabuľku, do ktorej ukladáme všetky súvislé L-prvkové podmnožiny báz genómu spolu z ich pozíciou. V hešovacej tabuľke je preto pre všetky L-prvkové postupnosti báz nahádzajúce sa v genóme zoznam ich pozícií v genóme a orientácia (či sa na genóme nachádzajú priamo alebo ich reverzný komplement).

DNA sekvencie tvoria 4 druhy báz. Jednu bázu vieme preto vyjadriť dvoma bitmi. Jadrá dĺžky najviac 32 bitov sa dajú zapísať v tvare jedného 64-bitového celého čísla. Výhodou tohto zápisu je jeho rýchle hešovanie a tiež generovanie. Predpokladajme, že máme sekvenciu uloženú v poli čísel $0, 1, 2, 3$ a chceme iterovať cez všetky úseky zvolenej dĺžky L vyjadrené ako 64-bitové číslo. Prepočet hodnoty pre úsek začínajúci na pozícii i na hodnotu pre úsek začínajúci na pozícii $i + 1$ vyžaduje len tri elementárne aritmetické operácie. Najprv sa bitovo posunieme hodnotu o 2 bity doľava, pripočítame hodnotu bázy na pozícii $i + L$ a na výsledok ešte použijeme bitovú masku $2^{2L} - 1$. Nasledujúci kód je ukážkou ako v našej implementácii iterujeme cez všetky úseky sekvencie a ich reverzné komplementy.

```
long long seed = 0;
long long reversedSeed = 0;
long long mask = ((long long)1 << (2 * L)) - 1;
for(int i = 0; i < dataLength) {
    seed = ((seed << 2) + data[i]) & mask;
    reversedSeed = ((reversedSeed << 2) + 3
        - data[dataLength - i - 1]) & mask;
    if (i >= L - 1) {
        // vkladanie do hesovacej tabulky
        // pripadne ine spracovanie
    }
```

}

S vytvoreným indexom vieme týmto istým spôsobom prejsť všetky úseky v čítaní a zistiť, kde a s akou orientáciu sa nachádzajú na genóme.

Spájanie jadier Zhodné úseky sú často dlhšie ako nami zvolený parameter L. Dlhé úseky sa prejavia tak, že niektoré skupiny jadier na seba bezprostredne nadväzujú. Na odľahčenie dalšieho spracovania je vhodné tieto jadrá spájať, pričom si pamätať ich dĺžku. Úlohou je vyrobiť zo zoznamu štruktúr [pozícia na genóme, pozícia na čítaní] zredukovaný zoznam štruktúr [pozícia na genóme, pozícia na čítaní, dĺžka]. Aby nadväzujúce jadrá v pôvodnom zozname boli bezprostredne za sebou, je potrebné ich usporiadať primárne podľa čísla diagonály a sekundárne podla pozície na genóme.

2.1.2 Jadrá s inzerciou

Parameter L výrazne ovplyvňuje počet nájdených jadier. Ak je zvolený príliš malý, nájde sa príliš veľa falošných jadier. Uvažujme genóm s náhodnou distribúciou báz dĺžky N. Stredná hodnota náhodných výskytov zvolenej sekvencie dĺžky L je $(N - L).4^{-L}$. Pri príliš veľkej hodnote L sa stáva, že sa nenájde ani jedno jadro zarovnania. Kedže čítania majú rôzne dĺžky, nie je možné tento parameter zvoliť tak, aby sme pri krátkych čítaniach vždy našli aspoň niekoľko skutočných jadier a pri dlhých nedoplácali na príliš veľké množstvo falošných readov.

V situácii, že nenájdeme ani jedno jadro, sa však dá použiť trik, ktorý s použitím toho istého indexu má veľkú šancú nájsť iné jadrá. Využijeme vlastnosť PacBio čítaní, že chyby čítania sú prevažne inzercie. Namiesto vyhľadávania súvislúch úsekov čítania dĺžky L, vyhľadávame dva súvislé úseky z čítania polovičnej dĺžky, ktoré v čítaní oddeľuje práve jeden znak. Vynechaný znak reprezentuje práve jednu inzerciu. Uvedieme príklad pre parameter $L = 10$. Takto vyzerajú jadrá zarovnania pre daný úsek čítania:

```
...ACGATGCATAGGCAAT...
```

```
ACGAT-CATAG
 CGATG-ATAGG
  GATGC-TAGGC
   ATGCA-AGGCA
    TGCAT-GGCAA
     GCATA-GCAAT
```

Takto vyzerá zarovnanie jedného z nich na genóm:

```
...ACGATGCATAGGCAAT...
      ATGCA-AGGCA
 ...GGATGCA-AGGCACT...
```

Vytváranie 64-bitovej reprezentácie týchto jadier sa dá tiež realizovať len pomocou aritmetických operácií.

```
for (int i = 0, i < read.length() - (length / 2) - 1) {
    // prva polovica
    firstHalf = ((firstHalf << 2) + data[i]) & mask;
    // druha polovica
    secondHalf = ((secondHalf << 2)
        + data[i + (length / 2) + 1]) & mask;

    if (i >= length / 2) {
        // spojenie do jedneho jadra s inzerciou
        seed = secondHalf + (firstHalf << length);
        auto positions = kMerMap.find(seed);
        if (positions != kMerMap.end()) {
            // vyhladavanie v hesovacej tabulke
        }
    }
}
```

Úspešnosť tejto metódy sme overili na simulovaných čítaniach dĺžky 1000 báz na genóme dĺžky 4,6 milióna báz. Na vyvtorenie simulovaných čítaní

sme použili nástroj pbsim [9] a parameter $L = 20$. Aspoň jedno súvislé jadro sa podarilo nájsť v 95,83% čítaní. Aspoň jedno jadro s inzerciou sa podarilo nájsť v 90,1% čítaní. Na čítaniach, na ktorých sa nepodarilo nájsť žiadne súvislé jadrá, sa podarilo nájsť jadrá s inzerciou v 84,91% prípadoch. Použitím oboch metód sme dosiahli úspešnosť hľadania jadier 99,37%.

2.1.3 Iné možnosti

Hešovacia tabuľka nie je jediná možnosť ako vyhľadávať zarovnania. V tejto časti stručne popíšeme, ako sa dajú hľadať pomocou sufixového poľa [7] a tiež dátovú štruktúru, ktorá by v našom prístupe mohla nahradiť hešovaciu tabuľku.

Sufixové pole

Sufixové pole je dátová štruktúra použiteľná na vyhľadávanie všetkých výskytov daného podreťazca v reťazci.

Máme reťazec $S = S[0]S[1]S[2]\dots S[n-1]$. Označme $F[i]$ pre $i \in \{0,\dots,n-1\}$ sufixy reťazca S začínajúce znakom $S[i]$. Sufixové pole A je pole celých čísel, ktoré na pozícii j obsahuje i také, že existuje j sufixov reťazca lexikografixky menších ako $F[i]$. Binárnym vyhľadávaním sa dajú získať pozície všetkých výskitov hľadaného podreťazca dĺžky m v čase $O(m \lg n)$, avšak využitím pokročilých techník [4] sa dá dosiahnuť časová zložitosť $O(m)$.

Ako príklad uvedieme sufixové pole na reťazci *mississipi*. V tabuľkách 2.1 sú uvedené všetky jeho sufixy usporiadané podľa začiatočnej pozície a lexikograficky. Tabuľka 2.2 zobrazujúca výsledné sufixové pole obsahuje postupne začiatočné pozície lexikograficky usporiadaných sufixov.

Hešovacia tabuľka s minimalizátorom

Dátovú štruktúru [11] budeme používať na vyhľadávanie jadier zarovnania pevnej dĺžky L. Zadefinujme minimalizátor ako lexikograficky najmenší

Sufix	i
mississippi	0
ississippi	1
ssissippi	2
sissippi	3
issippi	4
ssippi	5
sippi	6
ippi	7
ppi	8
pi	9
i	10
	11

(a)

Sufix	i
	11
i	10
ippi	7
issippi	4
ississippi	1
mississippi	0
pi	9
ppi	8
sippi	6
sissippi	3
ssippi	5
ssissippi	2

(b)

Tabuľka 2.1: Zoznam sufixov zoradený podľa začiatočnej pozície (a) a lexikograficky (b)

j	0	1	2	3	4	5	6	7	8	9	10	11
A[j]	11	10	7	4	1	0	9	8	6	3	5	2

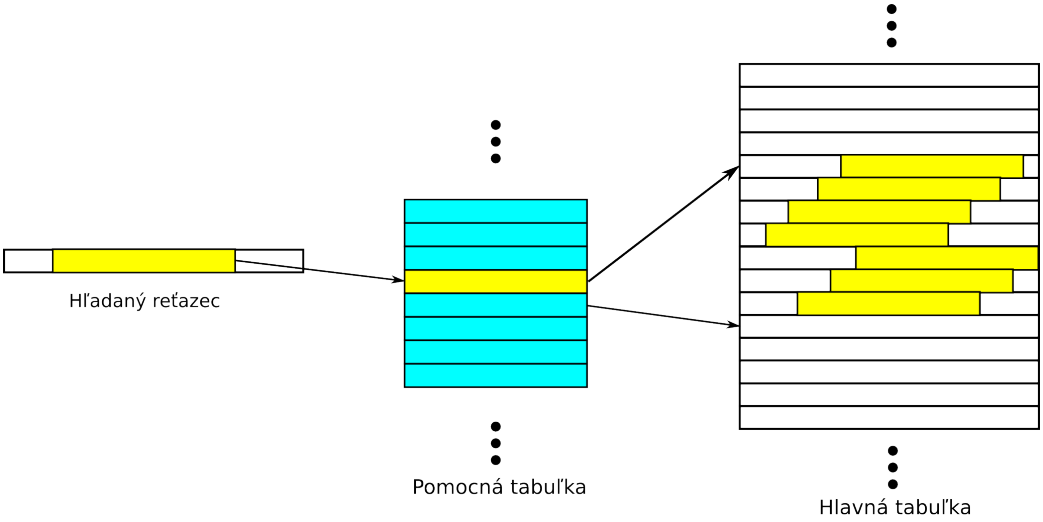
Tabuľka 2.2: Sufixové pole na reťazci *mississipi*

súvislý podreťazec pevnej dĺžky M z daného reťazca. Index tejto dátovej štruktúry pozostáva z dvoch tabuliek. V hlavnej tabuľke sú uložené všetky podreťazce genómu dĺžky L spolu s ich pozíciami. Túto tabuľku usporiadame primárne podľa minimalizátorov a sekundárne podľa celých podreťazcov. V pomocnej tabuľke je pre každý minimalizátor uložená pozícia v hlavnej tabuľke, od ktorej všetky podreťazce obsahujú lexikograficky väčší alebo rovný minimalizátor. Takto vytvorená pomocná tabuľka nám umožní v konštantnom čase nájsť tú časť hlavnej tabuľky, v ktorej sa nachádzajú podreťazce s hľadadým minimalizátorom.

Vyhľadávanie v tejto štruktúre urýchľuje fakt, že dva po sebe vyhľadávané reťazce sa skoro celé prekrývajú, lebo postupne prechádzame celé čítanie. Takto sa prekrývajúce reťazce majú väčšinou zhodný minimalizátor. Proces vyhľadávania preto začína binárnym vyhľadávaním podreťazca na tom istom úseku hlavnej tabuľky, na ktorom sme hľadali predchádzajúci podreťazec. Ak sa ho nepodarilo nájsť, vypočítame nový minimalizátor. Ak sa nezmenil, vieme, že podreťazec sa v genóme nenachádza. Ak sa zmenil, binárne vyhľadávame podreťazec na úseku prislúchajúcemu novému minimalizátoru. Viacnásobné binárne vyhľadávanie na jednom úseku pamäte je veľmi výhodné vzhľadom na cache.

2.2 Vytváranie zarovnaní

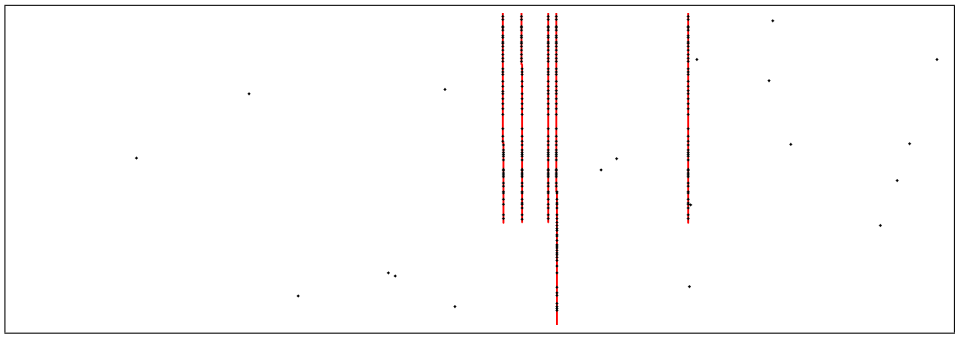
Vytváranie zarovnaní medzi jadrami pomocou dynamického programovania je príliš pomalý proces. V našom nástroji sme preto zvolili prístup použitý v nástroji DALIGN popísaný v predchádzajúcej kapitole. Implementáciu samotného vytvárania lokálneho zarovnania v jazyku C je dostupná prostredníctvom služby GitHub. Aby sme ju mohli pohodlne používať v našom nástroji písanom v jazyku C++, sme pre ňu vytvorili vlastné objektovo orientované rozhranie. Vstupom pre zarovnávač sú dve sekvencie a súradnice, cez ktoré ma zarovnanie prechádzať. Na výstupe dostaneme najlepšie zarovnanie v oboch smeroch, ktoré končí buď tým, že zarovnávač došiel na koniec nie-



Obr. 2.1: Vizualizácia vyhľadávania pomocou minimalizátora

ktorej zo sekvencii, alebo tým, že lokálne už nespĺňa požiadavky na kvalitu.

Pomocou popísaného zarovnávača skúšame vytvárať zarovnania z každého nájdeného jadra zarovnania. Pre falošné jadrá výpočet skončí veľmi rýchlo, lebo okrem nájdenej presnej zhody sa v okolí nachádzajú akoby náhodné dáta. Na miestach, kde sa dá zarovnať celé čítanie, alebo nejaká jeho významná časť, je jadier viacero. Aby sme nestrácali čas viacnásobným vytváraním toho istého kusu zarovnania, je treba pamätať si, ktoré miesta na editačnom grafe už sú pokryté v nejakom zarovnaní. Na vyriešenie tejto úlohy sa tiež dá použiť hešovacia tabuľka. Pre každé zarovnanie získame pomocou nášho rozhrania zoznam súradníc na genóme a čítaní, cez ktoré zarovnanie prešlo. Pridávať do tabuľky každý bod by bolo zbytočne časovo náročné. Preto pred ich pridaním do hešovacej tabuľky celočíselne vydelíme súradnice zvolenou konštantou a samotné pridávanie vykonáme, len ak sa nový pridávaný bod nerovná tomu predchádzajúcemu. Súradnice sú 32-bitové čísla a tie vieme uložiť do jedného 64-bitového, čo urýchli proces hešovania. S takto vytvorenou štruktúrou už vieme pre každé jadro povedať, či sa už nachádza v



Obr. 2.2: Vizualizácia jadier a zarovnaní na editačnom grafe

nejakom vypočítanom zarovnaní, alebo je potrebné ho vypočítať.

Obrázok 2.2 znázorňuje rozloženie jadier zarovnání a z nich vytvorené lokálne zarovnania v editačnom grafe. Zvislá os je pozícia na čítaní a vodorovná je pozícia na genóme. Jadrá sú reprezentované čiernymi bodmi a zarovnania sú červené čiary. Genóm je mnohonásobne dlhší ako čítanie, preto nie je zachovaný pomer strán a zarovnania sú v obrázku skoro kolmé.

2.2.1 Spájanie zarovnávaní

Reálne dáta majú v rôznych miestach rôznu chybovosť. Úseky s vyššou chybovosťou niekedy nespĺňajú podmienku pre lokálnu kvalitu zarovnania. Preto sa stáva, že čítanie sa nepodarí zarovnať celé, ale len po častiach. Tieto časti na seba zjavne nadväzujú a preto je potrebné ich spojiť do jednej. Na tento prípad používame osobitnú inštanciu zarovnávača s vyššou tolerancoiu chýb.

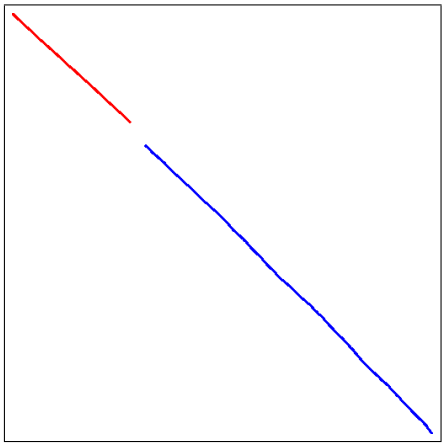
Aby sme nestrácali čas krátkymi zarovnaniami, ktoré vzniknú okolo každého jadra, zadefinovali sme podmienku, ktoré zarovnania sú významné. Keďže získanú množinu významnych zarovnaní budeme potrebovať sekvenčne prechádzať, vkladať do nej prvky a mazať z nej existujúce prvky, je vhodné ju uložiť do spájaného zoznamu. Spájanie zarovnávaní prebieha tak, že pre každé zarovnanie A skúsime nájsť iné zarovnanie B, ktoré naň nadväzuje v kladnom smere. Podmienka nadväznosti testuje, či zarovnania ležia na blízkych diagonálach a či začiatky a konce majú správnu vzájomnú pozíciu v zmysle

antidiagonál. Ak najdeme zarovnanie B, A nahradíme novým zarovnaním vytvoreným zarovnávačom s vyššou toleranciou chýb. Nové zarovnanie má šancu prejsť aj mierne horší úsek, a tým spojiť zarovnania A a B do jedného. Následne je potrebné znovu prejsť zoznam zarovnaní a vymazať tie, ktoré rozšírené A pokrylo.

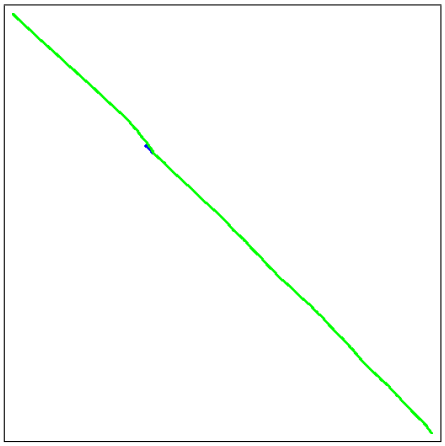
Zjednodušená implementácia je uvedená v nasledujúcom bloku kódu:

```
for (auto it = alList.begin();
        it != alList.end(); ++it) {
    bool doRealign = false;

    // hladanie napojitelneho zarovnania
    for (auto it2 = alList.begin();
            it2 != alList.end(); ++it2) {
        if (it == it2) continue;
        if (AreConnectable(*it, *it2)) {
            doRealign = true;
            numConnectableBefore++;
            break;
        }
    }
    // ak sa naslo, pouzijeme zarovnavac
    // s vyssou toleranciou
    if (doRealign) {
        Alignment newAlignment;
        dwHighTolerance.ComputeAlignment(genome, r
            ead, it->GetStartPos(), newAlignment);
        newAlignment.ComputeTrace();
        *it = newAlignment;
    }

    // mazanie prekrytych zarovnani
```

```
    list<Alignment>::iterator erasable;
    for (auto it2 = alList.begin();
            it2 != alList.end(); ) {
        erasable = it2++;
        if (it == erasable) continue;
        if (IsCoveredBy(*erasable, *it)) {
            alList.erase(erasable);
        }
    }
}
```



Obr. 2.3: Nadväzujúce zarovnania



Obr. 2.4: Výsledok po zarovnaní s vyššou toleranciou chýb

Obrázky 2.3 a 2.4 sú vykreslené pomocou nástroja na export zarovnaní, ktorý je súčasťou našej implementácie. Čiara jednej farby reprezentuje jedno zarovnanie v editačnom grafe. Zvyslá os reprezentuje pozíciu na čítaní, vodorovná pozíciu na genóme. Na obrázku 2.3 sú zobrazené dve zarovnania, ktoré náš algoritmus označil za nadväzujúce. Spojnica medzi koncom červeného a začiatkom modrého zarovnania má väčší sklon ako samotné zarovnania. To je spôsobené vyšším počtom chýb (prevažne inzercií) na danom úseku. Použitím zarovnávača s vyššou toleranciou chýb sa v tomto prípade podarilo

zarovnať oba úseky spolu. Výsledné zarovnanie zobrazené na obrázku 2.4 takmer presne pokrýva obe zarovnania.

Kapitola 3

Experimenty

V tejto kapilote porovnáme náš nástroj na mapovanie s oficiálnym nástrojom na mapovanie PacBio čítaní – BLASR. Budeme sledovať čas spracovávania a efektivitu týchto nástrojov na genóme baktérie Escherichia Coli K12 MG1665 [5] a množine čitaní dĺžky aspoň 1000 báz získaných pomocou sekvenovacieho zariadenia PacBio RS II. Tie isté parametre porovnáme aj na simulovaných čítaniach vytvorených pomocou nástroja pbsim. Okrem testovania finálnej verzie tiež overíme vplyv vylepšení uvedených v predchádzajúcej kapitole.

3.1 Porovnanie na PacBio čítaniach

V tejto časti porovnáme čas behu a úspešnosť hľadania zarovnaní nášho nástroja s nástrojom BLASR na reálnych dátach. Rozdiely medzi genómom a čítaniami sú tu spôsobené nie len chybovosťou vyplývajúcou z metódy čítania, ale aj skutočnými rozdielmi medzi referenčným genómom a genómom organizmu, z ktorého boli čítania získané. V tabuľke 3.1 sú uvedené základné informácie o mapovaných čítaniach.

Úspešnosť zarovnávania porovnávame na základe najdlhšieho zarovnania pre dané čítanie. Ak je rozdiel dĺžok najdlhších zarovnaní menší ako 200, považujeme nástroje rovnako úspešné na danom čítaní. Tento rozptyl sme použili preto, lebo nástroje sa správajú rôzne na koncoch zarovnania a preto

Počet	29291
Maximálna dĺžka	14494
Minimálna dĺžka	1000
Medián dĺžky	2606

Tabuľka 3.1: Základné informácie o zarovnávaných PacBio čítaniach

	BLASR	Náš algoritmus
Čas [s]	606,39	80,61
Rovnaká úspešnosť	20825	
Vyššia úspešnosť	5786	1279

Tabuľka 3.2: Porovnanie úspešnosti zarovnávania na pbsim čítaniach

by dochádzalo by k k systematickému zvýhodňovaniu jedného z nich aj v skoro totožných zarovnaniach. Výsledky porovnávania sú uvedené v tabuľke 3.2. Riadok Vyššia úspešnosť udáva počet zarovnaní, na ktorych bol uvedený nástroj úspešnejší ako ten druhý. Náš zarovnávací nastroj je niekoľko krát rýchlejší ako BLASR, avšak BLASR na 19,75% čítaní našiel dlhšie zarovnania. Náš nástroj si počínal lepšie len na 4.37% čítaní. To, že bol BLASR na niektorých čítaniach úspešnejší, môže byť spôsobené aj tým, že náš nástroj nejaké zarovnanie prehlásil za dve, pretože úsek medzi nimi mal veľmi malú koreláciu. V takejto situácii treba zvážiť, ktorý výsledok je "správnejší".

3.2 Porovnanie na simulovaných čítaniach

V tejto časti porovnáme čas behu a úspešnosť hľadania zarovnaní dvoch nástrojov na čítaniach vytvorených pomocou nástroja pbsim. Tieto čítania majú s referenčným genómom vyššiu koreláciu, lebo boli simulovane čítané z toho istého referenčného genómu. V tabuľke 3.3 sú uvedené základné vlastnosti simulovaných čítaní.

Počet	28578
Maximálna dĺžka	24678
Minimálna dĺžka	1000
Medián dĺžky	2605

Tabuľka 3.3: Základné informácie o zarovnávaných PacBio čítaniach

	BLASR	Náš algoritmus
Čas [s]	660,58	67,87
Rovnaká úspešnosť	28185	
Vyššia úspešnosť	4	367

Tabuľka 3.4: Porovnanie úspešnosti zarovnávania na pbsim čítaniach

Nástroje najprv porovnáme rovnakým spôsobom ako na skutočných dátach. Výsledky sú uvedené v tabuľke 3.4. Čas behu nášho nástroja je stále niekoľko násobne kratší, ale na týchto čítaniach je naša úspešnosť podobná a možno aj lepšia ako v prípade BLASRu.

Nástroj pbsim okrem čítaní generuje aj ich pozície na genóme, takže úspešnosť môžeme porovnať aj presnejšie ako len na základe najdlhšieho zarovnania. Preto sme vytvorili jednoduchý program, ktorý najprv zistí, kde sa čítania majú namapovať a následne pre každé čítanie určí, ako najviac túto správnu pozíciu niektoré zarovnanie pokrýva. V tabuľke 3.5 sú uvedené počty čitaní, ktoré boli zarovnané na spravný úsek na genóme aspoň na uvedený počet percent.

So znižujúcou sa podmienkou na pokrytie sa úspešnosť nášho algoritmu nemení takmer vôbec. Je to spôsobené tým, že rozširovanie zarovnaní algoritmom G. Mayersa je na týchto simulovaných čítaniach veľmi úspešné. Ak sa podarí nájsť aspoň jedno správne jadro zarovnania, je skoro isté, že sa podarí namapovať celé čítanie.

Pokrýva aspoň [%]	BLASR	Náš algoritmus
100	3035	28561
95	27035	28561
90	28174	28561
80	28536	28563
70	28560	28563
50	28569	28564
30	28569	28564
10	28570	28564

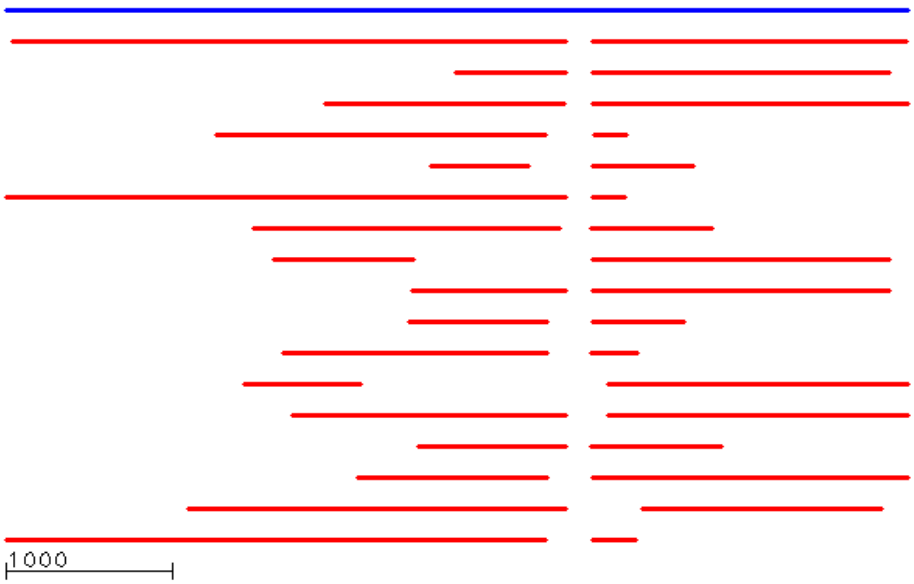
Tabuľka 3.5: Porovnanie úspešnosti zarovnávania na pbsim čítaniach podľa referencie

3.3 Úseky s nízkou koreláciou

V tejto časti sa zameriame na úseky s nižšou koreláciou medzi referenčným genómom a PacBio čítaniami. Ak sú naozaj spôsobené rozdielom medzi referenčným genómom a genómom organizmu, z ktorého boli získané čítania, mali by sa prejaviť nich všetkých zarovnaniach, ktoré cez ne prechádzajú. Na overenie tejto hypotézy sme si nechali vykresliť množiny dvojíc nadväzujúcich zarovnaní, ktoré sú rozdelené na prekrývajúcich sa úsekoch a obsahujú aspoň N prvkov. Jeden z nich je na obrázku 3.1.

3.4 Zhrnutie

Náš nástroj je, ako sme očakávali, rýchlejší v porovnaní s BLASRom. Jeho nevýhodou je však rozdeľovanie zarovnaní, ktoré obsahujú úsek s horšou koreláciou. Tento problém čiastočne vyriešilo implementovanie spájania zarovnaní. Ešte lepšie výsledky by sa možno dali dosiahnuť zásahmi do samotného zarovnávacieho algoritmu, ale to už v tejto práci nepokryjeme.



Obr. 3.1: Úsek s nízkou koreláciou

Záver

Podarilo sa nám naprogramovať nástroj na mapovanie čítaní sekvenovacieho zariadenia Pacific Biosciences RS II. Je výrazne rýchlejší ako oficiálny nástroj BLASR, avšak na reálnych dátach nedosahuje rovnakú úspešnosť zarovnávania celých čítaní. Je dostupný cez službu GitHub na adrese:

`https://github.com/marcelschichman/bakalarka`.

Literatúra

[1] Konstantin Berlin, Sergey Koren, Chen-Shan Chin, James Drake, Jane M Landolin, and Adam M Phillippy. Assembling large genomes with single-molecule sequencing and locality sensitive hashing. *bioRxiv*, page 008003, 2014.

[2] Mark J Chaisson and Glenn Tesler. Mapping single molecule sequencing reads using basic local alignment with successive refinement (blasr): application and theory. *BMC bioinformatics*, 13(1):238, 2012.

[3] Richard Durbin. *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge university press, 1998.

[4] Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 390–398. IEEE, 2000.

[5] Illumina. E.coli mg1655 illumina sequencing dataset. `ftp://webdata:webdata@ussd-ftp.illumina.com/Data/SequencingRuns/MG1655/MiSeq_Ecoli_MG1655_110721_PF.bam`, 2015. Accessed: 2015-03-03.

[6] Sergey Koren, Michael C Schatz, Brian P Walenz, Jeffrey Martin, Jason T Howard, Ganeshkumar Ganapathy, Zhong Wang, David A Rasko, W Richard McCombie, Erich D Jarvis, et al. Hybrid error correction and de novo assembly of single-molecule sequencing reads. *Nature biotechnology*, 30(7):693–700, 2012.

[7] Udi Manber and Gene Myers. Suffix arrays: a new method for on-line string searches. *siam Journal on Computing*, 22(5):935–948, 1993.

[8] Gene Myers. Efficient local alignment discovery amongst noisy long reads. In *Algorithms in Bioinformatics*, pages 52–67. Springer, 2014.

[9] Yukiteru Ono, Kiyoshi Asai, and Michiaki Hamada. Pbsim: Pacbio reads simulator—toward accurate genome assembly. *Bioinformatics*, 29(1):119–121, 2013.

[10] Temple F Smith and Michael S Waterman. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, 1981.

[11] Derrick E Wood and Steven L Salzberg. Kraken: ultrafast metagenomic sequence classification using exact alignments. *Genome Biol*, 15(3):R46, 2014.