

國立臺灣大學電機資訊學院電子工程學研究所



碩士論文

Graduate Institute of Electronics Engineering
College of Electrical Engineering and Computer Science
National Taiwan University
Master Thesis

使用快速取樣與連結策略之新式長序列基因映射器

A Novel Long Read Aligner
Using Fast Seeding and Linking Strategies

陳乃群

Nae-Chyun Chen

指導教授：盧奕璋 博士

Advisor: Yi-Chang Lu, Ph.D.

中華民國 106 年 7 月

July, 2017

國立臺灣大學碩士學位論文

口試委員會審定書

使用快速取樣與連結策略之新式長序列基因映射器

A Novel Long Read Aligner Using Fast Seeding and
Linking Strategies

本論文係陳乃群君 (R04943093) 在國立臺灣大學電子工程學研究所完成之碩士學位論文，於民國壹零陸年柒月肆日承下列考試委員審查通過及口試及格，特此證明

口試委員：

陸文璋

陳沛陸 (指導教授)

陳子群

陳倩瑜

系主任、所長

吳海宇



致謝

首先要感謝我的指導老師盧奕璋老師，從大三開始連續五年跟著老師學習、做研究，老師給予我極大的自由度，讓我不受拘束得追尋自己有興趣的各個研究主題，老師對知識的熱情、做人做事的嚴謹、還有帶領學生的態度都深深讓我敬佩，當初選擇老師的專題研究是我最大的幸運。其次感謝我的口試委員們陳和麟老師、陳倩瑜老師以及陳沛隆老師。您們都是我的重要引航員，帶領我走入演算法、生物資訊以及基因體醫學等領域的無盡寶山。也感謝聯詠科技的豐厚獎學金，大幅降低我的經濟壓力，得以全心投入研究。

實驗室的夥伴們也讓我的碩士生涯加上多姿的顏色。育誠是我在生物資訊領域的導師，也是我極限趕 paper 的好夥伴，很謝謝你提供我許多重要的研究建議，也希望在畢業之後仍然有繼續合作的機會。謝謝聖睿、騏銘和冠杰，雖然我們研究領域有些不同，但一起戰鬥、討論學術與關心生活是最美好的回憶。筱漩、陽明、鈺翔、纓喻、皓瑋、其昀、東霖、茂然、政彥、岳霖、健安、士軒、承擘、智宇，與你們一起在 BL430 的日子非常開心，感謝你們的陪伴。

另外也謝謝明仁、泓諭、國婷、冠豪、意晴、思鐸、令儀、婷瑋、曜宏、彥謙、人碩、煒甯、昇勳、彥如、威全、伯偉、林翔、子淵、彥安、宗維等好友的支持鼓勵，看著你們常常覺得自己要更努力，希望在未來大家都能開創出自己的的一片天。謝謝我的家人對我的包容，你們是我最堅固的後盾。當然也要謝謝女友昱廷，無論是同在台大或是新竹台北遠距離，我們總是一起分享生活，妳常常能站在我的角度給我務實的建議，期許未來我們都可以成為更好的人。

隨著這本碩士論文的完成，在台大唸書的兩千多個日子即將結束，我也將邁入下一段未知的旅程。很幸運能在台灣最好的大學成長，台大的豐富資源讓我得以在理工、社會科學與生醫領域徜徉，此外，這間學校也用它的一切帶給我一份社會關懷與良善價值。在未來現實與夢想碰撞的長路上，這些養分將伴隨我前進。

2017.07.16 陳乃群

中文摘要



自二十一世紀以來，次世代定序的突破性進展使得基因定序的金錢與時間成本皆大幅降低，開展一系列全新的生物與醫學研究。於 2015 年，利用特殊奈米孔道的高通量定序技術問世，此定序方法能夠快速分析更長的基因序列，有潛力解決許多序列分析的困難，例如大範圍的序列增減、重複區或是單倍型分析。然而由於這類奈米孔道定序的正確率尚不如前一代定序技術，對目前的主流基因映射演算法來說，映射奈米孔道序列相對困難。

在本論文中，我們針對低正確率的長基因序列，提出了一基於布洛—惠勒斯轉換之短序列取樣與連結之基因映射器 (BWSL)。此一方法採用包含取樣階段、連結階段與排列階段的三級架構設計。從目標序列中取樣出大量的短序列以提高映射的成功率，在連結與序列排列階段，我們設計了有效率的一維投影與分區動態演算法以提高程式效率與映射正確率。

在基於模擬奈米孔道序列的實驗中，BWSL 的映射靈敏度優於目前最好的序列映射器。以人類第四對染色體的模擬序列實驗為例，BWSL 的映射靈敏度高達 98.35%，與文獻中表現最好的 GraphMap 相比，結果高出 0.83%。此外，BWSL 比對結果的每鹼基對平均分數與變異分析結果也優於 GraphMap，顯示 BWSL 在奈米孔道序列相關應用上的潛力。

關鍵字：基因序列映射、高通量定序、次世代定序、奈米孔道定序



A Novel Long Read Aligner

Using Fast Seeding and Linking Strategies

Nae-Chyun Chen

Advisor: Yi-Chang Lu

Graduate Institute of Electronics Engineering

National Taiwan University

Taipei, Taiwan

July, 2017





Abstract

In the previous decade, the advent of high-throughput sequencing makes it possible to acquire and analyze sequence data with low cost and high speed. Since 2015, nanopore-based single-molecule sequencing platforms can generate reads longer than thousands of base pairs at high speed. However, when compared to the accuracy of traditional sequencer, the sequencing accuracy of nanopore platforms is relatively lower, which becomes a great challenge for sequence aligners.

In this thesis, we propose Burrows-Wheeler-transform-based aligner with Seeding and Linking (BWSL) to efficiently align long nanopore reads. A three-stage architecture involving seeding, linking and extending is designed for sensitive mapping and accurate alignment. A great number of short seeds is generated to ensure high mapping quality. The seeds are processed with novel algorithms to efficiently be mapped to correct positions and generate accurate alignments.

The sensitivity of BWSL on synthetic MinION datasets outperforms current state-of-the-art mappers. Using human chromosome 4 dataset as an example, the sensitivity reaches as high as 98.35% in BWSL, which is 0.83% better than GraphMap. Also, BWSL has high average alignment scores and great variant calling accuracy.





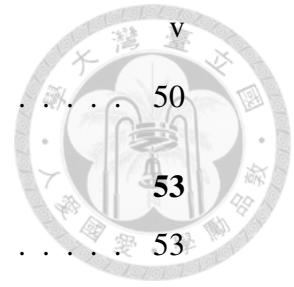
Contents

Abstract	i
List of Figures	vii
List of Tables	xi
1 Introduction	1
1.1 High-Throughput Sequencing	1
1.1.1 Overview of Sequencing Technologies	1
1.1.2 Sequence Data Processing	2
1.1.3 Short Read Sequencing	3
1.1.4 Single-Molecule Sequencing	3
1.2 Motivation	4
1.3 Contribution	5
1.4 Thesis Organization	5
2 Background	7
2.1 Sequence Alignment with Dynamic Programming	7
2.2 String Searching with Indexes	9
2.2.1 Burrows-Wheeler Transform	9
2.2.2 Ferragina-Manzini Index	10
2.3 Related Works	14
2.3.1 BLAST: Basic Local Alignment Search Tool	14



2.3.2	BWA: Burrows-Wheeler Aligner	16
2.3.3	GraphMap	17
3	Algorithms and Software Implementation	19
3.1	Overview	19
3.2	Seeding Stage	20
3.2.1	Seeds Sampling	21
3.2.2	Seeds Alignment	22
3.2.3	Filtration of Hits	23
3.3	Linking Stage	24
3.3.1	Putative Read Positions	25
3.3.2	Queue-Based Flexible Histogram and Partial Sorting	26
3.4	Extending Stage	28
3.4.1	Genome Alignment Using Constant-Memory Trace-Back (GACT)	28
3.4.2	Dynamic Filtration Strategies	31
3.5	Software Implementation	34
3.5.1	The BWA Front-End	34
3.5.2	Memory Management	34
4	Experiments	39
4.1	Overview of the Results of BWSL	39
4.2	Alignment Results with Synthetic Data	41
4.2.1	Synthetic Datasets	41
4.2.2	Evaluation of Sensitivity and Base-Level Accuracy	42
4.2.3	Visualization of Alignment Results	43
4.2.4	Variant Analysis of Alignment Results	44
4.3	Parameters Selection Guidelines	45
4.3.1	Max Number of Hits Allowed	47
4.3.2	Number of Chains	49

4.3.3	The size of the First Tile	50
5	Conclusions and Future Work	
5.1	Conclusions	53
5.2	Future Work	54
	References	55

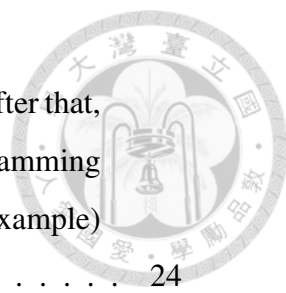







List of Figures

1.1	A sequence aligner maps reads onto correct positions on the reference.	2
2.1	An example of Needleman-Wunsch algorithm.	11
2.2	An example of Smith-Waterman algorithm.	12
2.3	An example of the suffix array and the BWT of sequence ACGCTTG\$. BWT is the last column of the sorted suffix table.	13
2.4	FM-index is a data structure based on Burrows-Wheeler transform. It is suitable for fast string searching. This figure shows an example of the FM-index of sequence ACGCTTG\$.	14
2.5	The working flow of BLAST. (a) The results of matched k -mers. (b) The linked two-hits. (c) The two-hits extended with ungapped extension. The gray area shows an HSP. (d) The HSP is extended with gapped extension.	15
2.6	BWA searches a read against a reference with a prefix trie. In this example, the read is AGC, the reference is ACAGCT\$, and the allowed mismatch number is one.	16
3.1	The workflow for BWSL.	20
3.2	Seeding Stage samples k -consecutive base pairs from a read (i.e. k -bp seeds) and aligns the seeds to the reference. The hits are marked as gray ellipses.	21

- 
- 3.3 An nine-bp seed is first exactly aligned to the reference. After that, the next five nucleotides are sent to the hit filter. If the Hamming distance between the seed and the reference (four in this example) is larger than the given threshold, the seed is removed. 24
- 3.4 Linking Stage translates hits into putative read positions, PRPs, and links the PRPs as chains. The black dotted lines show three chains. The gray dotted regions show the regions covered by the chains. 25
- 3.5 The PRP of a forward-strand hit and reverse-strand hit is calculated separately with Equations 3.6 and 3.7. After the translation, the hits are mapped to the starting positions of reads. 26
- 3.6 An illustration of the sliding queue in Q-FHPS. (a) The element farther than l_{chain} from PRP_i is popped from the queue. There are total six elements in this FHPS bin (including PRP_i itself). (b) PRP_i is then pushed to the queue and the reference point is changed to PRP_{i+1} . There are five elements in the new FHPS bin. 27
- 3.7 Extending Stage is based on GACT with dynamic filtration strategy. The solid line shows the final alignment. The gray boxes are the tiles used in GACT. The first tile is larger than secondary tiles to tolerate short-range imprecision of Linking Stage. 30
- 3.8 The finite state machine of two-level termination. This design helps BWSL tolerate some poorly aligned regions in an alignment for a noisy long read. 33
- 3.9 Operating system organizes the fragmented heap and clean up continuous memory space using memory compaction. 35
- 3.10 The hit positions are stored with a hybrid memory allocation method. 36

- 
- 4.1 An example shows that the alignment recorded by the simulation report is not always the optimal alignment. Here M, D, I stands for match/mismatch, deletion and insertion, respectively. Alignment score is calculated with the default settings of BWSL. (a) Alignment shown in the simulation report. The alignment score is -5 .
 (b) Mathematically optimal alignment. The alignment score is 7 41
- 4.2 Visualize the alignment results of synthetic *E. coli* dataset with IGV from (a) 999 to 42,049 (b) 1,000,999 to 1,042,049 (c) 2,000,999 to 2,042,049 (d) 3,000,999 to 3,042,049 bases. 47

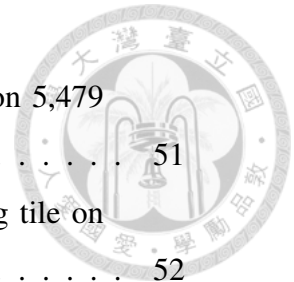




List of Tables

4.1	Simulation parameters for ONT reads.	42
4.2	Five synthetic datasets from 2-kbp to 2-Mbp.	43
4.3	Comparing the alignment results by BWSL with GraphMap on synthetic ONT 2D datasets.	44
4.4	The comparison of characteristics of alignments generated by BWSL and GraphMap with 248,068 synthetic reads generated with <i>H. sapiens</i> chromosome 4.	45
4.5	Comparing the alignment results by BWSL with GraphMap on synthetic ONT 2D datasets for <i>S. cerevisiae</i> chromosome 1 to 5 (the rest are not listed here).	46
4.6	The number of variants called by GraphMap and BWSL on two synthetic datasets with different depths.	48
4.7	The sensitivity and precision of GraphMap and BWSL results on synthetic <i>C. elegans</i> dataset with 86,066 point mutations added.	48
4.8	Comparison of different MNHA coefficients on 5,479 synthetic ONT 2D reads from <i>S. cerevisiae</i>	49
4.9	Comparison of different NC coefficients on 5,479 synthetic ONT 2D reads from <i>S. cerevisiae</i>	50
4.10	Profiling results with different NC coefficients on 5,479 synthetic ONT 2D reads from <i>S. cerevisiae</i>	50
4.11	Comparison of different widths of first Extending tile on 5,479 synthetic ONT 2D reads from <i>S. cerevisiae</i>	51

4.12 Comparison of different heights of first Extending tile on 5,479 synthetic ONT 2D reads from <i>S. cerevisiae</i>	51
4.13 Profiling results with different heights of first Extending tile on 5,479 synthetic ONT 2D reads from <i>S. cerevisiae</i>	52





Chapter 1

Introduction

1.1 High-Throughput Sequencing

1.1.1 Overview of Sequencing Technologies

The Human Genome Project completed in 2003 provided the very first human reference genome. Since then, the sequencing technologies have improved at the extraordinarily fast pace. In comparison with traditional Sanger sequencing [1], new methods enable faster and lower-cost sequencing. The new sequencing technologies are called high-throughput sequencing, or next-generation sequencing (NGS).

According to a report provided by National Institutes of Health (NIH) of United States [2] in 2016, generating a high quality human genome sequence costs about 14 million USD in 2006, and it takes less than 1,500 by late-2015. High-throughput sequencing opens a new era for biological and medical research. Also, new computational methods are proposed to deal with the high volume data generated by new sequencers.

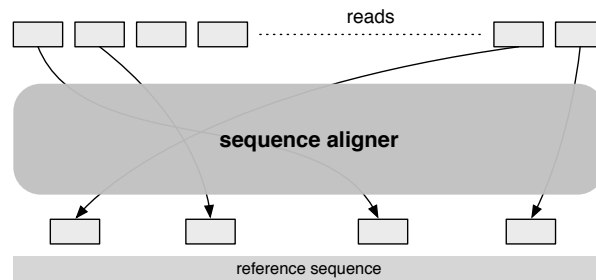


Figure 1.1: A sequence aligner maps reads onto correct positions on the reference.

1.1.2 Sequence Data Processing

Sequences generated by sequencers record piecewise information instead of a whole picture of genome. Therefore, the sequence data need be arranged and organized for further analysis. The arrangement is processed with two methods: *de novo* assembly and reference-based alignment. A *de novo* assembly algorithm merges related reads (short DNA sequences) and generates longer sequences, called contigs. Contigs carry more information than raw reads and are helpful for sequence analysis. A reference-based alignment algorithm maps reads onto a pre-built reference sequence, as shown in Figure 1.1. The differences between DNA sample and the reference are call variants, which are important messages for further analysis. In this thesis, we focus on reference-based alignment, or sequencing alignment.

In comparison with *de novo* assembly, sequence alignment is faster and requires less sequencing depth with the guidance of the reference sequence. Since modern sequencing machines generate high throughput sequencing results, it is not time efficient to align sequences to references with dynamic-programming-based optimal algorithms such as Needleman-Wunsch [3] algorithm and Smith-Waterman [4]. To deal with this problem, a great variety of sequence aligners are proposed. Hashing-based aligners such as BLAST [5] and NextGenMap [6] make use of hash functions as a fast seeding strategy. Graph-based aligners such as deBGA [7] and GraphMap [8] utilize graph structures as an efficient sequence

searching algorithm. Index-based algorithms such as LAST [9], Bowtie2 [10] and BWA-MEM [11] use pre-built indices to accelerate searching speed. Note that aligners are not limited to only one of the classes. For example, GraphMap also pre-builds an index for fast searching.

1.1.3 Short Read Sequencing

Popular DNA sequencing platforms such as HiSeq from Illumina and SOLiD from Life Technologies (now Thermo Fisher) break DNA strands into reads and sequence the DNA subsequences. Reads are further divided into two groups: short-reads (35-bp to 700-bp) and long-reads (longer than 700-bp) [12]. HiSeq and SOLiD systems generate reads with lengths range from 36 base pairs (bps) to 250 bps and are categorized as short-read sequencing platforms. The short-read sequencing platforms generate accurate results (more than 99%) with high throughput and are now dominating the market.

However, both the sequencing-by-ligation approach adopted by SOLiD and sequencing-by-synthesis approach adopted by HiSeq have their intrinsic limitations against sequencing longer reads. With short reads, it is difficult to analyze long-range insertions or deletions (indels), structural variations, repetitive sequences and haplotypes. Even though many computational methods with heuristics and statistical methods have been proposed to solve this issue [13], the confidence of the analysis results is limited. Therefore, new technologies are needed to sequence long reads with high quality and high throughput.

1.1.4 Single-Molecule Sequencing

After 2010, new technologies have emerged to generate longer reads. Pacific Biosciences developed their PacBio sequencing system and Oxford Nanopore Technologies (ONT) built a system named MinION. The two systems are based on single-molecule sequencing method. The adoption of single-molecule template

enables the generation of long reads and simplifies sequencing preparation process. Some researchers call the new systems third-generation sequencing platforms [14] in comparison with second-generation sequencing counterparts such as SOLid and HiSeq.

Among the third-generation sequencing systems, the MinION from ONT, a portable commercially available since 2014, features its real-time streaming, simple preparation and ultra-long read lengths (up to hundreds of kb). It uses a biological pore to sense DNA molecules passing through and applies deep-learning-based algorithms to analyze the responses of currents. ONT claims that these ultra-long reads have great potential in high quality *de novo* assembly, variant analysis, metagenomics, epigenetics and other biological/medical applications [15].

1.2 Motivation

Despite being a promising sequencing technology, currently nanopore sequencing is not able to generate long reads with high accuracy. While short read sequencers have more than 99% sequencing accuracy, the accuracy for one-dimensional (1D) ONT reads is only 83-85%. The accuracy for two-dimensional (2D) ONT reads is around 95% after the R9 release in 2016 [16, 17], which is a great improvement but is still inferior to accuracy of the short read counterparts. The relatively-low sequencing accuracy makes it difficult for sequence mapping or alignment tools to generate high-quality results. Traditional state-of-art sequence aligners such as BWA-MEM [11] and Bowtie2 [10] are designed for high-quality reads. They are not able to generate alignments with sufficient sensitivity if ONT reads are used as inputs.

In 2016, GraphMap [8] is proposed as the very first sequence aligner designed for nanopore data. GraphMap is able to map ONT sequences with more than 95% sensitivity and high speed. However, we find that the alignment results generated

by GraphMap have short Hamming distance with the reference, but the base calling accuracy is low. Therefore, an algorithm to align ONT reads with both high sensitivity and base-level accuracy is still needed.



1.3 Contribution

In this thesis, a novel long read sequence aligner is proposed, named Burrows-Wheeler-transform-based aligner with Seeding and Linking (BWSL). BWSL is a seed-and-extend aligner designed to deal with low-accuracy long reads. With its three-stage architecture, BWSL aligns reads on the reference with great sensitivity and base-level accuracy. First, Seeding Stage samples a great number of short seeds to locate the rough position of a read. Second, Linking Stage uses a novel queue-based algorithm to link the aligned seeds and suggest possible alignment regions. Finally, Extending Stage splits a read into subsequences and generate alignments step by step. Early termination mechanisms are designed to enhance the time efficiency with little loss of alignment accuracy.

In our experiments, BWSL is the sequence aligner with highest sensitivity to the best of our knowledge. Though GraphMap has close sensitivity, the base-level accuracy of BWSL outperforms GraphMap.

1.4 Thesis Organization

In this chapter, we summarize the development of high-throughput sequencing, discuss our motivation and present the main contributions of this thesis. The remainder of the thesis is organized as follows. The background and related works are introduced in Chapter 2. The proposed algorithms and our software implementation details are described in Chapter 3. The experiments are discussed in Chapter 4. Finally, the conclusion is given in Chapter 5.





Chapter 2

Background

2.1 Sequence Alignment with Dynamic Programming

In 1970, Needleman and Wunsch proposed a dynamic programming algorithm to align two amino acid sequences [3]. This algorithm, called Needleman-Wunsch algorithm, is capable of finding the optimal alignment of two sequences end-to-end. In BWSL, Needleman-Wunsch algorithm is adopted with modification.

Needleman-Wunsch algorithm can be divided into two steps: alignment matrix filling and back tracing. In the alignment matrix filling step (Lines 3 to 7 in Algorithm 1), insertion matrix (I), deletion matrix (D) and general score matrix (H) are generated using Equations 2.1 to 2.4. In the following equations, R, Q, s_m, s_s, g_o, g_e stands for reference sequence, query sequence, match score, mismatch penalty, gap open penalty and gap extension penalty, respectively.

$$s(r, q) = \begin{cases} s_m & \text{if } R(r) = Q(q), \\ s_s & \text{otherwise.} \end{cases} \quad (2.1)$$

$$I(r, q) = \max \begin{cases} H(r, q-1) + g_o, \\ I(r, q-1) + g_e. \end{cases} \quad (2.2)$$



$$D(r, q) = \max \begin{cases} H(r-1, q) + g_o, \\ D(r-1, q) + g_e. \end{cases} \quad (2.3)$$

$$H(r, q) = \max \begin{cases} H(r-1, q-1) + s(r, q), \\ I(r, q), \\ D(r, q). \end{cases} \quad (2.4)$$

Along with alignment matrix filling, tracing-back matrix (T) is updated with Equations 2.5 and 2.6. The dot symbol (\cdot) in Equation 2.5 refers to the starting point of an alignment. After filling all the matrices, the alignment is calculated with back tracing. Back tracing starts from the bottom-right of tracing-back matrix (T) and extends along the directions of the arrows in tracing-back matrix (T). When extension reaches the top-left point of tracing-back matrix (T), Needleman-Wunsch algorithm ends and the optimal global alignment is obtained. An example of the alignment of sequences ACGCTTG and AGCTTTG with Needleman-Wunsch algorithm is shown in Figure 2.1. In this example $\{s_m, s_s, g_o, g_m\}$ are set $\{5, -4, -8, -6\}$, which are identical to the default scoring parameters of BWSL. The final score is 14 and the suggested alignment is marked as gray boxes.

$$\mathcal{T}(x) = \begin{cases} \swarrow & \text{if } x = 1, \\ \uparrow & \text{if } x = 2, \\ \leftarrow & \text{if } x = 3, \\ \cdot & \text{otherwise.} \end{cases} \quad (2.5)$$

$$T(r, q) = \mathcal{T} \left(1 + \operatorname{argmax} \begin{cases} H(r-1, q-1) + s(r, q) \\ I(r, q) \\ D(r, q) \end{cases} \right). \quad (2.6)$$

One of the most widely-adopted modifications of Needleman-Wunsch algorithm is Smith-Waterman algorithm [4], proposed in 1981. It differs from Needleman-Wunsch algorithm in that Smith-Waterman algorithm has an additional zero term

in Equation 2.4 to represent the beginning of a new alignment. Also, the alignment score calculated by Smith-Waterman algorithm is the maximum score of the entire general score matrix (H), not the bottom-right score in Needleman-Wunsch algorithm. These modifications make Smith-Waterman algorithm capable of finding highest-scoring subsequences instead of aligning sequences from end to end. This type of alignment strategy, called local alignment, is suitable for finding similar regions in sequences regardless of their similarity. In nucleotide or protein sequences, similar regions in two sequences carry important information for the relationship between the sequences. Therefore, local alignment tools are widely adopted in phylogenetics and other applications focused on local similarities. An example of Smith-Waterman algorithm is shown in Figure 2.2.

BWSL is a global alignment tool so the core aligning algorithm is Needleman-Wunsch algorithm. Besides, we borrow some ideas from Smith-Waterman algorithm and modify our aligning algorithm. Therefore the two dynamic programming methods are both introduced here.

2.2 String Searching with Indexes

2.2.1 Burrows-Wheeler Transform

Burrows-Wheeler transform (BWT) algorithm is proposed in 1994 as a special character rearrangement in a data compression pipeline [18]. It is an effective method to put identical characters together for a better compression rate. BWT is a character string containing all last characters of the lexically sorted suffixes from the original character string. To construct the BWT of a sequence X , a common method is to apply Equation 2.7 to the suffix array (SA) of X , where the suffix array records the indexes of the lexically sorted suffixes of X . Figure 2.3 shows an example of the suffix array and BWT of sequence ACGCTTG\$, where the dollar



Algorithm 1 Needleman-Wunsch Algorithm with Affine Gaps

Input: Reference sequence R and query sequence Q .

Output: Alignment score $score$ and alignment $align$.

```

1:  $align \leftarrow$  empty string
2:  $score \leftarrow 0$ 
3: for  $r \leftarrow 0$  to  $\text{len}(R) - 1$  do
4:   for  $q \leftarrow 0$  to  $\text{len}(Q) - 1$  do
5:     Calculate  $I(r, q), D(r, q), H(r, q), T(r, q)$  with
       Equations 2.2, 2.3, 2.4 and 2.6, respectively.
6:   end for
7: end for
8:  $r \leftarrow \text{len}(R) - 1, q \leftarrow \text{len}(Q) - 1$ 
9:  $score \leftarrow H(r, q)$ 
10: Trace back from  $T(r, q)$  and update  $align$ .
11: return  $\{score, align\}$ 

```

sign (\$) is an end-of-sequence symbol.

$$BWT[i] = \begin{cases} X[SA[i] - 1] & \text{if } SA[i] > 0, \\ \$ & \text{if } SA[i] = 0. \end{cases} \quad (2.7)$$

2.2.2 Ferragina-Manzini Index

In 2000, Ferragina and Manzini applied Burrows-Wheeler transform to string searching [19]. They combined suffix array, BWT, and two auxiliary tables as a compound index, called Ferragina-Manzini index (FM-index). Also, a fast string searching method was proposed. The combination of FM-index and its corresponding string searching method make the novel algorithm efficient to locate substrings in the original string. Though there is extra memory usage, FM-index is compressible with reasonable tradeoff in searching speed. Therefore, it is widely

		A	C	G	C	T	T	G
	-	-	-	-	-	-	-	-
A	-8	-16	-22	-28	-34	-40	-46	-52
G	-14	-3	-11	-17	-23	-29	-35	-41
C	-20	-9	-7	-4	-14	-20	-26	-29
T	-26	-15	-6	-10	-1	-9	-15	-21
T	-32	-21	-12	-10	-7	4	-4	-10
T	-38	-27	-18	-16	-13	-2	9	1
G	-44	-33	-24	-22	-19	-8	6	5

(a) The insertion matrix (I).

		A	C	G	C	T	T	G
	-	-8	-14	-20	-26	-32	-38	-44
A	-	-16	-3	-9	-15	-21	-27	-33
G	-	-22	-11	-7	-6	-12	-18	-24
C	-	-28	-17	-6	-11	-1	-7	-13
T	-	-34	-23	-14	-10	-9	4	-2
T	-	-40	-29	-20	-18	-15	-4	9
T	-	-45	-35	-26	-24	-20	-10	1
G	-	-52	-40	-32	-21	-27	-15	-5

(b) The deletion matrix (D).

		A	C	G	C	T	T	G
	0	-8	-14	-20	-26	-32	-38	-44
A	-8	5	-3	-9	-15	-21	-27	-33
G	-14	-3	1	2	-6	-12	-18	-21
C	-20	-9	2	-3	7	-1	-7	-13
T	-26	-15	-6	-2	-1	12	4	-2
T	-32	-21	-12	-10	-7	4	17	9
T	-38	-27	-18	-16	-13	-2	9	13
G	-44	-33	-24	-13	-19	-8	3	14

(c) The general scoring matrix (H).

		A	C	G	C	T	T	G
	0	←	←	←	←	←	←	←
A	↑	↖	←	←	←	←	←	←
G	↑	↑	↖	↖	←	←	←	←
C	↑	↑	↖	↖	↖	←	←	←
T	↑	↑	↑	↖	↖	↖	←	←
T	↑	↑	↑	↑	↑	↑	↖	↖
T	↑	↑	↑	↑	↑	↑	↑	↖
G	↑	↑	↑	↑	↑	↑	↖	↖

(d) The tracing-back matrix (T).

Figure 2.1: An example of Needleman-Wunsch algorithm.

used in modern genome alignment algorithms.

An example of the FM-index of ACGCTTG\$ is shown in Figure 2.4. Count table C is an array with size of σ , where σ is the size of the alphabet set (Σ). For DNA sequences, Σ contains four types of nucleotides $\{A, C, G, T\}$ and thus $\sigma = 4$. Occurrence table O is a two-dimensional table with size $\sigma \times n$, where n is the length of X . These tables are defined as follows:

$$C(a) = |\{0 \leq i \leq n-1 : X[i] < a\}| \quad (2.8)$$

$$O(a, i) = |\{0 \leq j \leq i : BWT[j] = a\}| \quad (2.9)$$

I		A	C	G	C	T	T	G
	-	-	-	-	-	-	-	-
A	-8	-8	-8	-8	-8	-8	-8	-8
G	-8	-3	-8	-8	-8	-8	-8	-8
C	-8	-8	-7	-3	-8	-8	-8	-3
T	-8	-8	-3	-8	2	-4	-8	-8
T	-8	-8	-8	-8	-4	7	-1	-7
T	-8	-8	-8	-8	-8	-1	12	4
G	-8	-8	-8	-8	-8	-3	6	8

(a) The insertion matrix (I).

D		A	C	G	C	T	T	G
	-	-8	-8	-8	-8	-8	-8	-8
A	-	-8	-3	-8	-8	-8	-8	-8
G	-	-8	-8	-7	-3	-8	-8	-8
C	-	-8	-8	-3	-8	2	-4	-8
T	-	-8	-8	-8	-8	-4	7	-1
T	-	-8	-8	-8	-8	-8	-1	12
T	-	-8	-8	-8	-8	-8	-3	4
G	-	-8	-8	-8	-3	-8	-8	-2

(b) The deletion matrix (D).

H		A	C	G	C	T	T	G
	0	0	0	0	0	0	0	0
A	0	5	0	0	0	0	0	0
G	0	0	1	5	0	0	0	5
C	0	0	5	0	10	2	0	0
T	0	0	0	0	2	15	7	1
T	0	0	0	0	0	7	20	12
T	0	0	0	0	0	5	12	16
G	0	0	0	5	0	0	6	17

(c) The general scoring matrix (H).

T		A	C	G	C	T	T	G

A	.	↖
G	.	.	↖	↖	.	.	.	↖
C	.	.	↖	.	↖	←	.	.
T	↑	↖	←	←
T	↑	↖	←
T	↖	↖	↖
G	.	.	.	↖	.	.	↑	↖

(d) The tracing-back matrix (T).

Figure 2.2: An example of Smith-Waterman algorithm.

With the pre-calculated FM-index, the suffix array interval $\{\underline{R}, \bar{R}\}$ of a query sequence aW can be calculated with Equations 2.10 and 2.11:

$$\underline{R}(aW) = C(a) + O(a, \underline{R}(W) - 1) + 1 \quad (2.10)$$

$$\bar{R}(aW) = C(a) + O(a, \bar{R}(W)) \quad (2.11)$$

As shown in Equations 2.10 and 2.11, FM-index-based searching is operated in reverse order, and thus it is also called backward searching [20]. Starting from the end-of-sequence character ($\$$), the suffix array interval of any query sequence

Sorted suffix table

	0	1	2	3	4	5	6	7
7	\$	A	C	G	C	T	T	G
0	A	C	G	C	T	T	G	\$
1	C	G	C	T	T	G	\$	A
3	C	T	T	G	\$	A	C	G
6	G	\$	A	C	G	C	T	T
2	G	C	T	T	G	\$	A	C
5	T	G	\$	A	C	G	C	T
4	T	T	G	\$	A	C	G	C

SA BWT



Figure 2.3: An example of the suffix array and the BWT of sequence ACGCTTG\$. BWT is the last column of the sorted suffix table.

can be acquired iteratively. Ferragina and Mezini proved that $\underline{R}(aW) \leq \overline{R}(aW)$ if and only if aW is a substring of X .

String searching allowing no mismatch is called exact searching. The time complexity for searching a k -bp query exactly is $O(k)$. After k iterations, $\{\underline{R}, \overline{R}\}$ represents the valid suffix array interval of the alignment. It is notable that the suffix array interval is not the actual position of a query on the reference. An additional transform shown in Equation 2.12 looks up the suffix array for actual alignment positions:

$$pos = SA([\underline{R} : \overline{R}]) \quad (2.12)$$

Despite being an efficient string searching data structure, the construction of FM-index is not straightforward. The naive method to construct a BWT or a suffix array requires memory space $O(n^2)$. Since there are more than 3 billion nucleotide bases in a human genome, $O(n^2)$ memory usage is not practical for most computing platforms. Therefore, a great amount of memory-efficient BWT constructing algorithms are proposed such as BWT-IS [21], lightweight indexing



	0	1	2	3	4	5	6	7
X	A	C	G	C	T	T	G	\$
SA	7	0	1	3	6	2	5	4
BWT	G	\$	A	G	T	C	T	C

Count table		A	C	G	T
		0	1	3	5

Occurrence Table		0	1	2	3	4	5	6	7
A	0	0	1	1	1	1	1	1	1
C	0	0	0	0	0	1	1	2	
G	1	1	1	2	2	2	2	2	
T	0	0	0	0	1	1	2	2	

Figure 2.4: FM-index is a data structure based on Burrows-Wheeler transform. It is suitable for fast string searching. This figure shows an example of the FM-index of sequence ACGCTTG\$.

with external memory [22] and ropeBWT2 [23].

2.3 Related Works

2.3.1 BLAST: Basic Local Alignment Search Tool

Basic Local Alignment Search Tool (BLAST) [5] is a popular local alignment tool based on heuristics against Smith-Waterman algorithm. The workflow of BLAST is illustrated in Figure 2.5. BLAST is one the earliest and most popular sequence aligners following the seed-and-extend paradigm. Since BWSL is also a seed-and-extend based sequencer, we introduce BLAST in this section in spite of the fact that it is not designed for NGS originally.

BLAST first constructs a k -mer table of the subject (reference) sequence. After

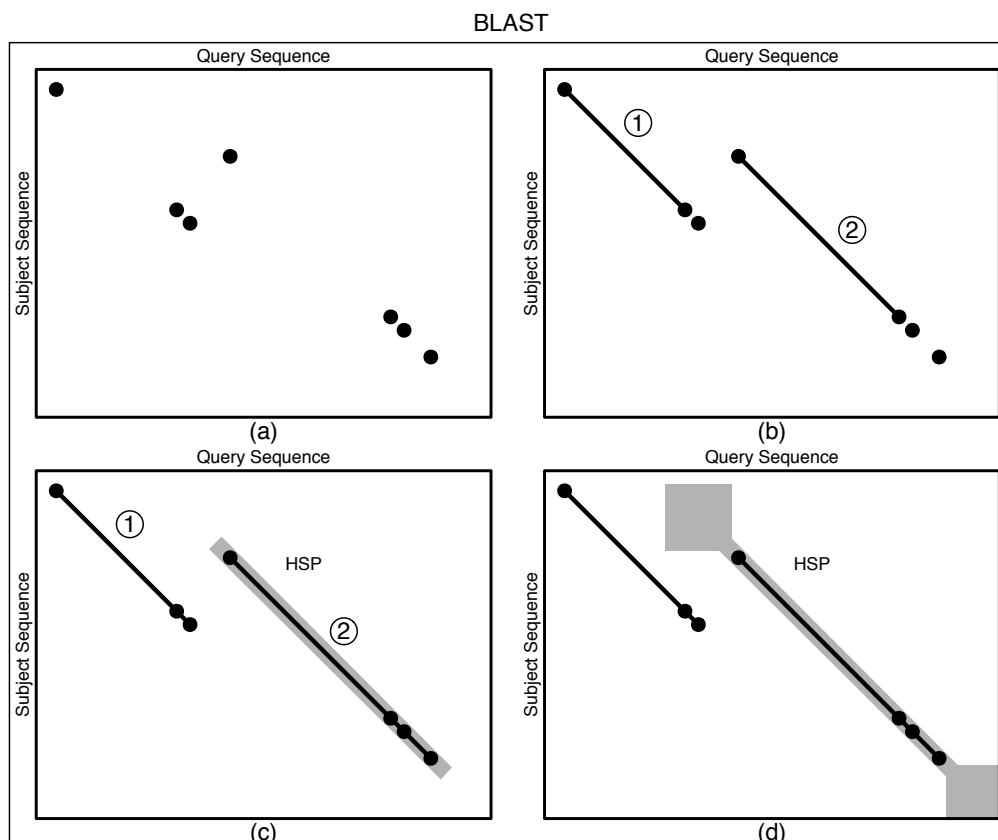


Figure 2.5: The working flow of BLAST. (a) The results of matched k -mers. (b) The linked two-hits. (c) The two-hits extended with ungapped extension. The gray area shows an HSP. (d) The HSP is extended with gapped extension.

that, a query sequence is read and scanned through the k -mer table with a hash function. If one query k -mer matches a subject k -mer, it is called a hit. For two hits locating on the same diagonal and within a limited number of residues, they are linked and extended ungappedly. If one ungapped extension segment has a higher score than a threshold, it is called a high-scoring segment pair (HSP). The HSPs are then extended using gapped Smith-Waterman algorithm and the final alignment score is calculated. Even though lots of new sequence alignment tools have emerged since the advent of next-generation sequencing, BLAST is still adopted by lots of researchers.

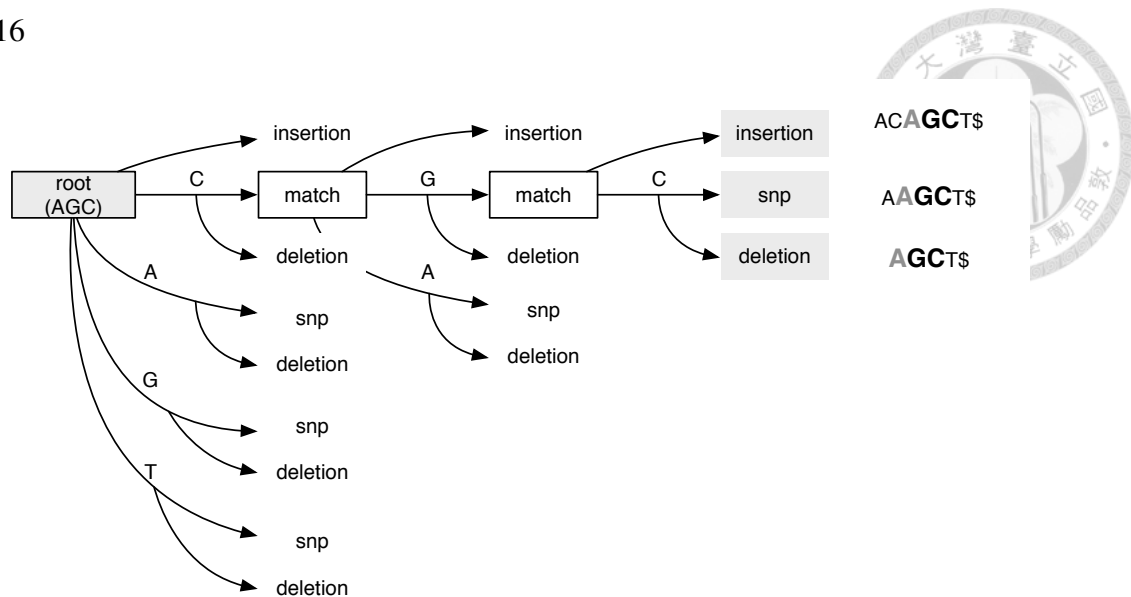


Figure 2.6: BWA searches a read against a reference with a prefix trie. In this example, the read is AGC, the reference is ACAGCT\$, and the allowed mismatch number is one.

2.3.2 BWA: Burrows-Wheeler Aligner

In the era of next-generation sequencing, there is a strong demand of fast alignment of numerous short reads against long references. Earlier methods have their limitations: BLAST is not fast enough for this task, MAQ [24] does not support gapped alignments for single-end reads and other tools are not efficient in terms of time or memory usage. To tackle this problem, Li and Durbin proposed Burrows-Wheeler Aligner (BWA-backtrack) [20] in 2009. It is the most popular next-generation sequencing alignment tool up to date and has been cited for more than 11,300 times since June, 2017.

BWA-backtrack builds an implicit prefix trie based on the FM-index of the reference sequence. To deal with gaps or mismatches (single nucleotide polymorphism, or snp), in each iteration, four possible aligning cases—match, mismatch, insertion and deletion—of a substring of the read are updated in the prefix trie. BWA-backtrack then traverses the trie iteratively to find the suffix array inter-

val. To accelerate the algorithm, there are many techniques applied such as a minimum-mismatch-bounding table $D(\cdot)$ and filtration strategies with heuristics [20].

Besides BWA-backtrack, two newer versions of seed-and-extend-based BWA are proposed: BWA-SW [25] and BWA-MEM [11]. Since BWA-backtrack is designed for Illumina reads up to 100 bps, the two newer tools are extended to align reads from 70 bps to 1 Mbps. The major difference between these two tools is in the method of generating seeds. In BWA-SW, the FM-indexes of queries are also constructed as a prefix directed acyclic word graph (prefix DAWG). Prefix DAWGs help enhance traversing speed in the reference prefix trie. In BWA-MEM, seeds are generated with supermaximal exact matches (SMEMs) [26]. After seeding, BWA-MEM adopts an additional chaining stage to group the hits as a filtration strategy. Both BWA-SW and BWA-MEM pass the seeds (or chains) to a Smith-Waterman extender and then generate final outputs.

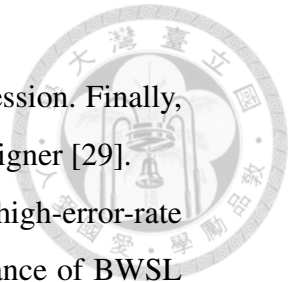
Currently the latest tool, BWA-MEM, outperforms BWA-backtrack and BWA-SW for almost all kinds of reads. It is regarded as one of the state-of-the-art aligners for modern sequencing applications. Besides Illumina sequences, BWA-MEM starts to support ONT 2D sequences in recent versions.

2.3.3 GraphMap

Algorithms such as BWA-MEM [11] and Bowtie2 [10] have great performance on high quality reads from tens-bp to million-bp. However, those algorithms are not suitable for noisy long reads generated by novel single-cell sequencing platforms such as MinION sequencer produced by Oxford Nanopore and PacBio systems by Pacific Biosciences. GraphMap [8] is a fast and sensitive sequencer proposed to deal with such sequences. First, it applies gapped q -grams [27] as a seeding strategy. Second, candidate regions are selected with clustering hits along diagonals. After that, a graph-based algorithm is applied to generate anchors in the candidate regions. The anchors are extended with Longest Common Subsequence in k

Length Substrings (LCSk) algorithm [28] and refined by linear regression. Finally, the alignment is generated using a dynamic-programming-based aligner [29].

GraphMap is one of the best aligners for sequencing potentially high-error-rate reads in terms of its quality and speed. We compare the performance of BWSL with that of GraphMap in this thesis. The details of the evaluations are described in Chapter 4.





Chapter 3

Algorithms and Software

Implementation

3.1 Overview

Third-generation sequencing platforms such as PacBio sequencers and nanopore sequencers are able to generate ultra-long reads to up to hundreds of thousand base pairs. The long read length makes it possible to detect long-range indels, repeats and structural variations. However, the alignment of third-generation long reads is much more difficult compared to traditional short read alignment due to the limited sequencing accuracy. Therefore, we propose BWSL to tackle noisy long reads.

BWSL consists of three main stages: seeding, linking and extending, as shown in Figure 3.1. First, Seeding Stage samples reads to generate seeds and align the seeds to the reference. The aligned seeds are called hits. Second, Linking Stage transforms the hits to putative read positions (PRPs), and links these PRPs as chains. Chains are potential candidates for correct alignment. Third, Extending Stage applies dynamic programming to align the chains suggested by Linking Stage and determines final alignments.

Seeding Stage, Linking Stage and Extending Stage are discussed in Section

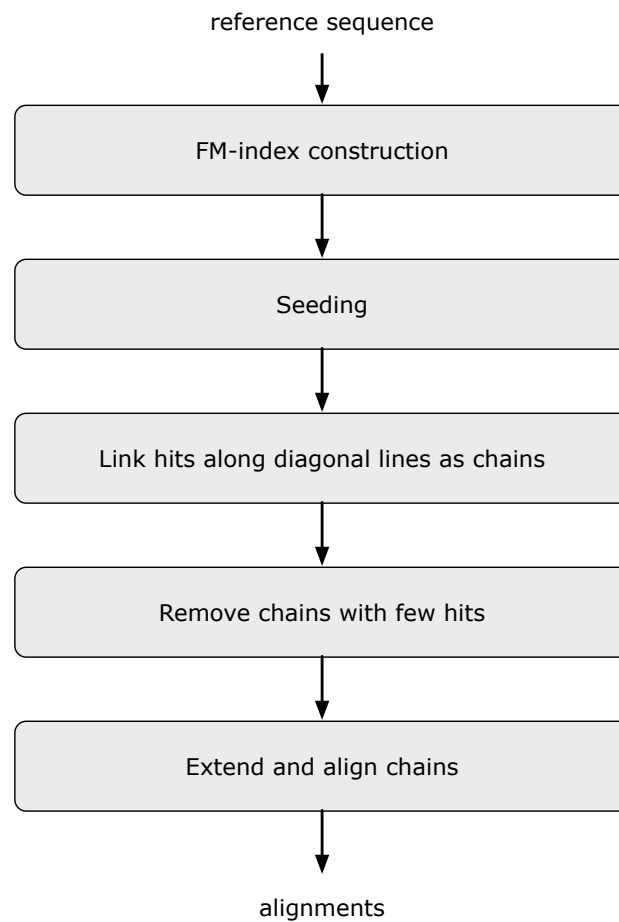


Figure 3.1: The workflow for BWSL.

3.2, 3.3 and 3.4, respectively. Also, in Section 3.5 the software implementation details are discussed.

3.2 Seeding Stage

In Seeding Stage, first seeds are sampled from each read. The sampling method is discussed in Section 3.2.1. Second, the seeds are aligned to the reference with the aid of FM-index. The details of the string searching algorithm is introduced in Section 3.2.2.

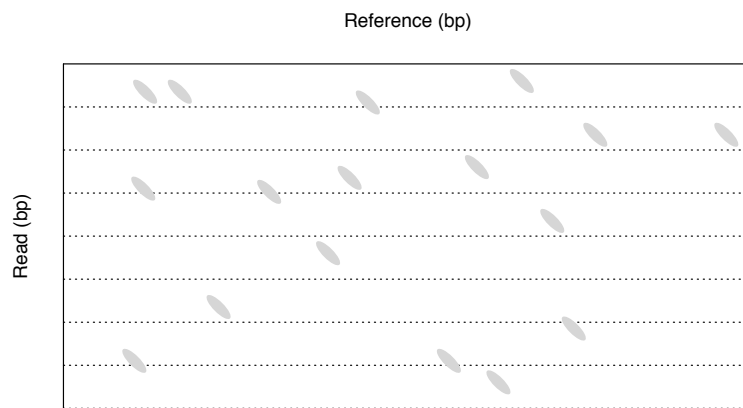


Figure 3.2: Seeding Stage samples k -consecutive base pairs from a read (i.e. k -bp seeds) and aligns the seeds to the reference. The hits are marked as gray ellipses.

3.2.1 Seeds Sampling

BWSL maximizes the probability of finding correct alignment with lots of small seeds. To sample as many seeds as possible without exhausting limited computing resource, the design of the seed processing core needs to be fast and low-cost. Therefore, we construct FM-index [22] for fast string search and then apply exact search to every seed.

Assume that the sequencing accuracy for one base is q and there is no correlation between bases. The expected hit number of seeds (S) with s k -bp seed seeds is as follows:

$$S = sq^k \tag{3.1}$$

In the reference sequence, there might be some subsequences happening to be identical with the seed at wrong positions. Such subsequences are called random hits. Some random hits occur because of biological similarity between sequences, and some are purely due to statistical randomness. Random hits mislead Linking Stage into suggesting incorrect chains, so they are regarded as sampling noises. Seeding Stage tries to raise the number of sampled seeds in the correct region and to avoid generating too many random hits. In a n -bp reference the number

of expected random hits (N) is shown in Equation 3.2. Thus, we define seed significance factor (SSF) by dividing S by N , as shown in Equation 3.3.

$$N = (n - k) \left(\frac{1}{4}\right)^k \quad (3.2)$$

$$SSF = \frac{S}{N} = \frac{s(4q)^k}{n - k} \quad (3.3)$$

The parameters selected in Seeding Stage needs to follow two criteria:

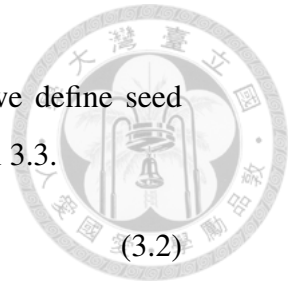
1. Expected hit number S needs to be sufficiently high.
2. Seed significant factor SSF needs to be sufficiently high.

Equations 3.1 to 3.3 show that there is a tradeoff between S and SSF by changing the seed length k . A longer seed length reduces the number of N and accelerates Seeding Stage. However, with a longer length seeds at correct positions are more likely to be dropped. This reduces S and has a negative impact to sequencing sensitivity. By contrast, a shorter seed length results in higher sensitivity but needs more computing time and memory. Empirically we suggest setting S and SSF higher than 50 and 0.5, respectively.

3.2.2 Seeds Alignment

The sampled reads are aligned exactly to the reference sequence with pre-calculated FM-index. Using the backward searching method with FM-index, the time complexity for aligning a k -bp seed is bounded by $O(k)$. The details of this string searching algorithm are described in Section 2.2.

Besides exact search, FM-index is also used in inexact search algorithms which generate alignments allowing mismatches. The algorithms usually use heuristics to filter results likely to be incorrect and accelerate search time. The inexact search algorithms have been used in lots of sequence aligners such as BWA [20] and Bowtie2 [10], which are among of the most cited sequence aligners ever. However, these two programs are designed for high quality reads and their



inexact search algorithms are only optimized when the number of mismatches is low. With increasing number of mismatches, the computing time of inexact search grows very fast. Since the sequencing accuracy for long reads is relatively low, it is inefficient to align whole long reads directly to the reference sequence inexactly. On the other hand, inexactly aligning seeds instead of whole read is another option. This strategy benefits from the efficient inexact search algorithms developed by previous researchers. However, in this case the number of expected random hits (N) grows much faster than expected hit number (S) as shown in Equations 3.1 and 3.2. This results in a large number of noisy hits and much longer processing time in looking up the suffix array table.

As a result, BWSL aligns seeds with exact search by default. It samples a large number of reads to guarantee the expected hit number (S) is sufficient. Since the aligning process is time-efficient, the high seed count does not have big negative impact on the overall performance of BWSL in time. Also, our program supports inexact search for applications requiring higher seeding sensitivity.

3.2.3 Filtration of Hits

A sensitive seeding algorithm maps seeds in a read sequence onto the correct positions in the reference sequence. In addition to sensitivity, precision is also an important factor when evaluating a seeding algorithm. An algorithm with high precision reduces the number of random hits and thus is more efficient in both time and memory usage. In BWSL, a hit filter is designed to improve the precision of Seeding Stage with little loss in sensitivity.

As shown in Figure 3.3, each hit is extended with a short sequence in the tail. If the Hamming distance between the additional tail sequence and the corresponding reference sequence is less than or equal to a given threshold, the hit passes the filter; otherwise it is removed. By default, the additional tail length (l_t) is five and the Hamming distance threshold (th_d) is three. For an ONT 2D sequence following the error profile shown in Section 4.2.1, the base sequencing accuracy

(q) is 0.69. With the above parameters, the correct seed passing rate (*CSPR*) and random seed passing rate (*RSPR*) are shown in Equations 3.4 and 3.5. In this case, more than 95% hits belong to the read are kept and less than 37% random hits are left. The hit filter accelerates the algorithm without much loss of sensitivity.

$$CSPR = 1 - \sum_{i=0}^{l_t - th_d - 1} \binom{l_t}{i} (1-q)^i (q)^{l_t - i} \Big|_{q=0.69, l_t=5, th_d=3} = 0.9653 \quad (3.4)$$

$$RSPR = 1 - \sum_{i=0}^{l_t - th_d - 1} \binom{l_t}{i} \left(1 - \frac{1}{4}\right)^i \left(\frac{1}{4}\right)^{l_t - i} \Big|_{l_t=5, th_d=3} = 0.3672 \quad (3.5)$$

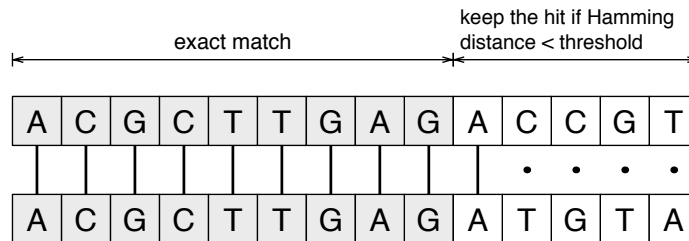
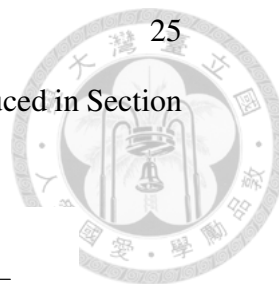


Figure 3.3: An nine-bp seed is first exactly aligned to the reference. After that, the next five nucleotides are sent to the hit filter. If the Hamming distance between the seed and the reference (four in this example) is larger than the given threshold, the seed is removed.

3.3 Linking Stage

The purpose of Linking Stage is to collect the hits generated in Seeding Stage and use the hits to find possible regions for alignment. This stage can be further divided into two steps. First the hits are projected along diagonal lines to estimate the starting position of the read as shown in Section 3.3.1. Second we propose a novel Queue-based Flexible Histogram with Partial Sorting (Q-FHPS) algorithm to link the hits. Linked hits are called chains, which indicate highly possible



regions for alignment. The details of Q-FHPS algorithm are introduced in Section 3.3.2.

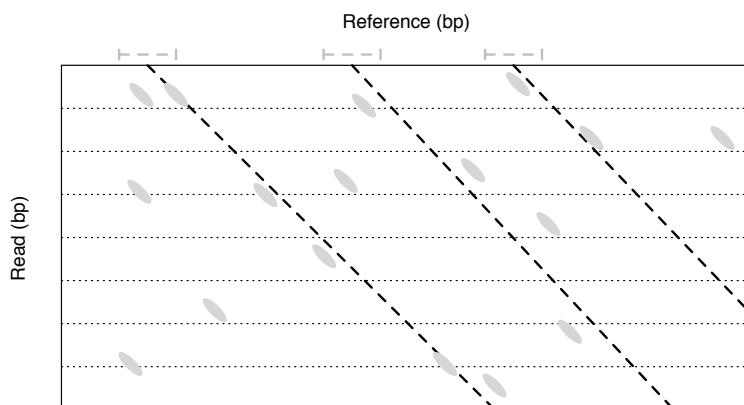


Figure 3.4: Linking Stage translates hits into putative read positions, PRPs, and links the PRPs as chains. The black dotted lines show three chains. The gray dotted regions show the regions covered by the chains.

3.3.1 Putative Read Positions

The hits calculated in Seeding Stage record the mapping positions of the seeds on the reference. Seeds sampled at different positions on a read have different offsets. Here we translate the hits to putative read positions, PRPs[7], to represent the putative starting position of each read. For a forward-strand hit, its PRP is calculated by subtracting its read offset to hit position in Equation 3.6. For a reverse-strand hit, its PRP is calculated with Equation 3.7.

$$PRP_{forward} = pos_{ref} - offset_{read} \quad (3.6)$$

$$PRP_{reverse} = pos_{ref} + offset_{read} + length_{seed} - length_{read} \quad (3.7)$$

Figure 3.5 illustrates the differences between Equations 3.6 and 3.7. We can regard PRPs as the topmost (bottommost) positions of diagonals extended from

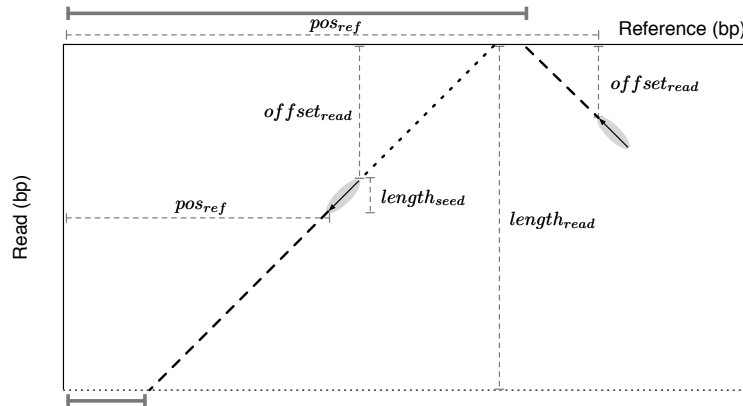


Figure 3.5: The PRP of a forward-strand hit and reverse-strand hit is calculated separately with Equations 3.6 and 3.7. After the translation, the hits are mapped to the starting positions of reads.

forward-strand (reverse-strand) hits. This translation removes the offsets on the read axis and puts all hits in the same one-dimensional coordinate system.

3.3.2 Queue-Based Flexible Histogram and Partial Sorting

To link hits efficiently, we propose Queue-based Flexible Histogram and Partial Sorting algorithm (Q-FHPS, Algorithm 2). Q-FHPS algorithm collects PRPs and builds a flexible histogram in which the bins are not located with a fixed interval. Also, to reduce computing time with as little loss of accuracy as possible, Q-FHPS algorithm applies partial sorting strategy to select and update a fixed amount of bins. The time and memory complexity of Q-FHPS algorithm is $O(p \log p)$ and $O(p \log n)$, where p and n stands for the number of PRPs and the length of reference sequence. The processing time of Q-FHPS algorithm is bounded by sorting p PRPs. The other parts have $O(p)$ time complexity.

To build the flexible histogram, a queue is designed to slide along the reference sequence and scan the sorted PRPs. As illustrated in Figure 3.6, in each iteration the distance between the front element in the queue and the current PRP is calcu-

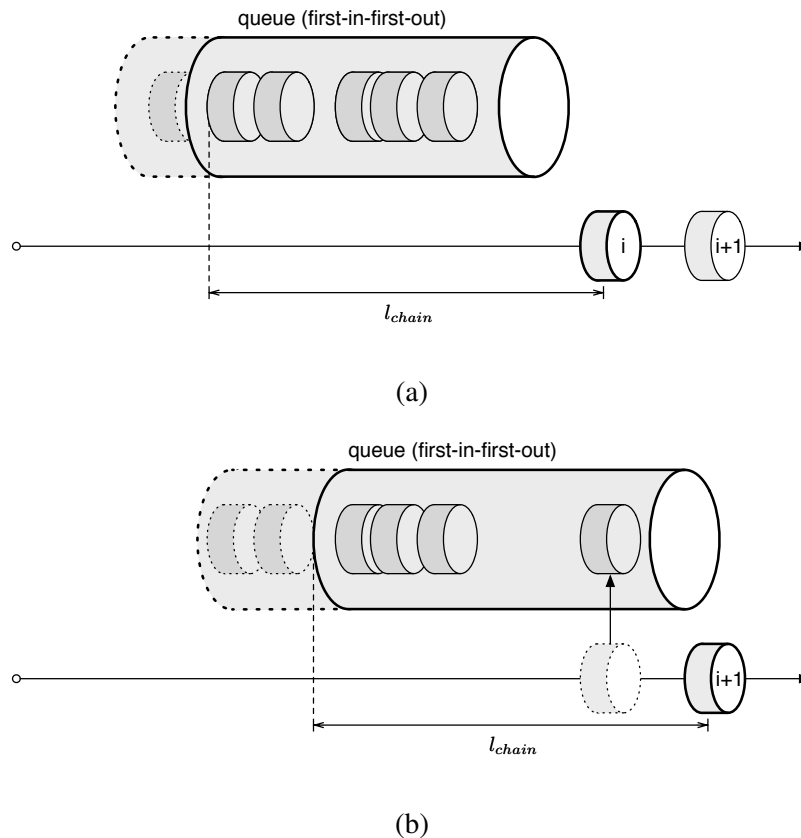


Figure 3.6: An illustration of the sliding queue in Q-FHPS. (a) The element farther than l_{chain} from PRP_i is popped from the queue. There are total six elements in this FHPS bin (including PRP_i itself). (b) PRP_i is then pushed to the queue and the reference point is changed to PRP_{i+1} . There are five elements in the new FHPS bin.

lated. If the distance is larger than the size of a chain l_{chain} , the front element in the queue is popped and this process repeats until the front element is within l_{chain} or the queue is empty. This process is shown from Lines 9 to 11 in Algorithm 2. After that, the size of the queue plus one (the current PRP itself) is the number of elements in the current flexible histogram bin. In this queue-based algorithm, the generated histogram does not have a fixed interval and the bins are dynamically located. Therefore, the histogram is called flexible histogram. Also, since PRP_i denotes the ending position of the search window, in Line 14 the updated

chain position is set to $\frac{PRP_i + Q.front()}{2}$ to approximate the middle position of the chain.

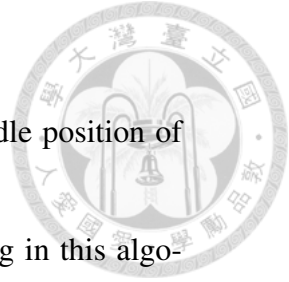
To reduce the number of final outputs, we apply partial sorting in this algorithm. First a given number of chains n_{chains} is initialized. The position and size of a chain are recorded and packed together. The size of a chain refers to the number of PRPs within distance l_{chains} from its central position. The chains only record the top- n_{chains} results, which are ranked by their sizes. If one flexible histogram bin has fewer members than the smallest-sized chain in the inventory, it is not likely to be a valid candidate and therefore discarded. With properly set n_{chains} , partial sorting reduces both time and memory complexity without sacrificing much sensitivity.

3.4 Extending Stage

In Extending Stage, we apply a memory efficient algorithm, Genome Alignment using Constant-memory Trace-back (GACT) [30], with modifications. GACT is effective in reducing the computing complexity of traditional dynamic programming algorithms. The details of GACT is described in Section 3.4.1. Also, we propose novel dynamic filtration strategies to accelerate the aligning process, as shown in Section 3.4.2.

3.4.1 Genome Alignment Using Constant-Memory Trace-Back (GACT)

GACT is a tile-and-align-based global alignment algorithm based on Needleman-Wunsch algorithm. It uses a constant-size tile to align subsequences. This method is efficient in terms of both time and memory usage. Also, GACT is able to align low-sequencing-accuracy long reads with little loss of speed and accuracy. Other heuristic sequence alignment algorithms such as banded Smith-Waterman and X-





Algorithm 2 Queue-Based Flexible Histogram and Partial Sorting

Input: Hit PRPs **PRP**, maximum number of chains n_{chain} and size of a chain l_{chain} .

Output: Suggested chains C .

```

1: Sort PRP.
2: Initialize queue  $Q$ .
3: Initialize  $n_{chain}$  empty chains  $C$ . //Each chain records its central
   position and hit counts.
4:  $Q.push(PR P_0)$ .
5: for  $i \leftarrow 1$  to  $size(\mathbf{PRP}) - 1$  do
6:   if  $Q.empty()$  then
7:      $Q.push(PR P_i)$ .
8:   else
9:     while  $Q.front() < PR P_i - l_{chain}$  do
10:       $Q.pop()$ 
11:    end while
12:     $c \leftarrow Q.size() + 1$  //  $c$  is the number of hits in chain
13:    if  $c > min(C.hitcounts)$  then
14:      Update  $argmin(C.hitcounts)$  with
        (hitcounts  $\leftarrow c$ , position  $\leftarrow \frac{PR P_i + Q.front()}{2}$ ).
15:    end if
16:  end if
17: end for
18: return  $C$ 

```

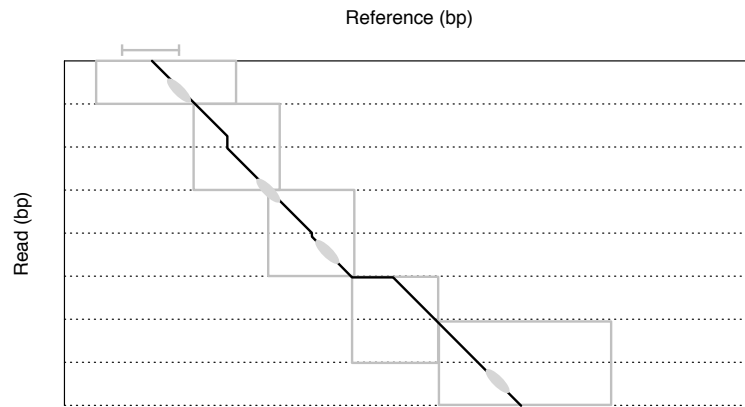


Figure 3.7: Extending Stage is based on GACT with dynamic filtration strategy. The solid line shows the final alignment. The gray boxes are the tiles used in GACT. The first tile is larger than secondary tiles to tolerate short-range imprecision of Linking Stage.

drop are restricted by the number of mismatches, while such number tends to be large for noisy long reads [30]. Also, GACT is a hardware-compatible algorithm. If needed, it is easy to be implemented on hardware systems such as ASICs and FPGAs.

In this thesis, we implement a modified GACT. This algorithm starts from the top-left part of the dynamic programming matrix and extends toward the bottom-right. In each iteration, it calculates a $T \times T$ tile with RB-NWalign algorithm (Algorithm 3). RB-NWalign algorithm is based on affine-gapped Needleman-Wunsch algorithm [3]. A difference between RB-NWalign and original Needleman-Wunsch algorithm is that RB-NWalign finds maximal alignment score on the rightmost column and bottommost row of the alignment matrix instead of simply selecting the bottom-right value. The modified GACT is proposed in Algorithm 4. After in-tile alignment, another tile is stitched to the current one at the max-scoring position as shown in Lines 12 and 13 in Algorithm 4. If the whole query or reference sequence is aligned, GACT terminates. Signal *warning* controls the termination of the algorithm with a dynamic strategy, the details of which

is described in Section 3.4.2.

Since indels may induce imprecision of chain positioning, the width (reference side) of the first tile (w_{ft}) is expanded handle the possible mismatch. The selection of w_{ft} is described in Section 4.3.3. Also, the size of the last tile is doubled in both width and height to avoid the need to use more tiles when there are long insertions or deletions in this tile. These parts are not shown in Algorithm 4 for better readability but illustrated in Figure 3.7.

Algorithm 3 RB-NWalign

Input: Reference sequence R , query sequence Q .

Output: Offset on reference off_R , offset on query off_Q , alignment $align$ and alignment score $score$.

- 1: Build alignment matrix using affine-gapped Needleman-Wunsch(R, Q).
 - 2: $score \leftarrow$ maximal score on right and bottom edges.
 - 3: Update off_R, off_Q and $align$.
 - 4: **return** $\{off_R, off_Q, align, score\}$
-

3.4.2 Dynamic Filtration Strategies

Even with the delicate heuristics of GACT, extending a sequence is still a task with very high time complexity. Moreover, since it is essential to provide enough aligning candidates to ensure high sensitivity, the number of chains suggested by Linking Stage is usually much larger than actual alignments. Therefore, how to reduce the time cost for unnecessary sequences is crucial to the performance of a long read aligner. Here we propose two dynamic filtration strategies in Extending Stage to enhance efficiency of BWSL.

As shown in Line 14 in Algorithm 4, in each iteration the tile alignment score is compared to an alignment terminating threshold. If the score is lower, the tile is considered a weakly aligned tile; otherwise it is a strongly aligned one. Since noisy long reads have low base sequencing accuracy, even in a correct alignment



Algorithm 4 GACT with Dynamic Filtration Strategy

Input: Reference sequence R , query sequence Q , tile size T , suggested chain position pos_R , terminating threshold th and degrade factor d .

Output: Alignment score $score$ and alignment $align$.

```

1:  $align \leftarrow ""$ 
2:  $total\_score \leftarrow 0$ 
3:  $warning \leftarrow \text{False}$ 
4:  $off_R, off_Q \leftarrow 0$ 
5:  $pos_Q \leftarrow 0$ 
6:  $i \leftarrow 0$ 
7: while  $pos_R + T < \text{len}(R)$  or  $pos_Q + T < \text{len}(Q)$  do
8:    $sub_R \leftarrow R[pos_R : pos_R + T]$ 
9:    $sub_Q \leftarrow Q[pos_Q : pos_Q + T]$ 
10:   $off_R, off_Q, tb, score \leftarrow RB\_NWalign(sub_R, sub_Q)$ 
11:   $align.append(tb)$ 
12:   $pos_R \leftarrow pos_R + T - off_R$ 
13:   $pos_Q \leftarrow pos_Q + T - off_Q$ 
14:  if  $score < th \times d^i$  then // Dynamic filtration
15:    if  $warning \leftarrow \text{True}$  then
16:      return  $\{total\_score + score, align\}$ 
17:    else
18:       $warning \leftarrow \text{True}$ 
19:    end if
20:  else
21:     $total\_score \leftarrow total\_score + score$ 
22:     $warning \leftarrow \text{False}$ 
23:  end if
24:   $i \leftarrow i + 1$ 
25: end while
26: return  $\{0, ""\}$ 

```

there might be some weakly aligned tiles. If a high alignment terminating threshold is set, extension is terminated when encountering such a tile in a correct alignment. On the other hand, a low threshold gives rise to heavy time cost with trying too many incorrect alignments. Therefore, a fixed alignment threshold is not suitable for this kind of data. To deal with this problem, we design two filtration strategies to dynamically determine whether extension should be terminated.

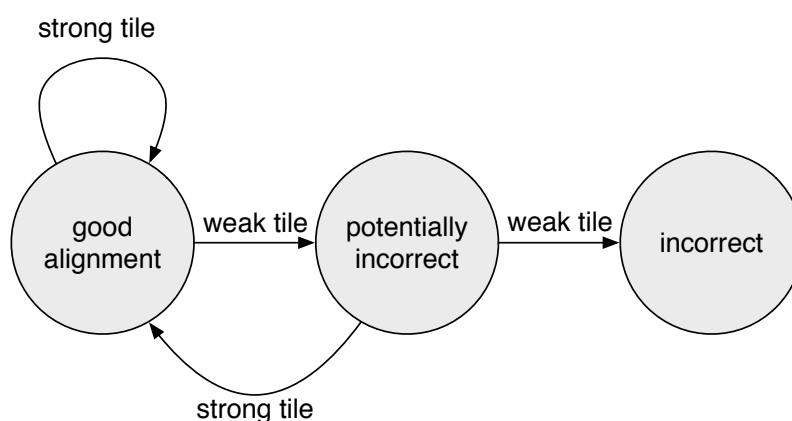
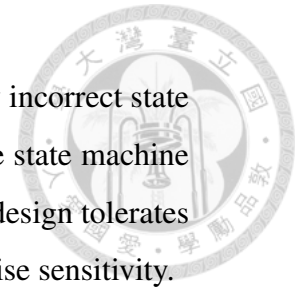


Figure 3.8: The finite state machine of two-level termination. This design helps BWSL tolerate some poorly aligned regions in an alignment for a noisy long read.

The first strategy is to discount the termination threshold. Since a query-reference combination is more likely to belong to a correct alignment if it is in a late GACT phase, we prefer completing the alignment to terminating it early. Even though this turns out not to be the best alignment, the cost is affordable. The second strategy is called two-level termination. The design is inspired from the 2-level adapting branch predictor [31], which is a widely used architecture in Central Processing Units (CPU). The design of two-level termination is illustrated in the finite state machine in Figure 3.8 and shown from Lines 15 to 22 in Algorithm 4. The finite state machine starts from the potentially incorrect state. When a strong tile occurs, it enters the good alignment state. When encountering a weakly aligned tile, the finite state machine enters the potentially incorrect state

rather than terminates the alignment promptly. If in the potentially incorrect state another weak tile is encountered, GACT ends; otherwise the finite state machine goes back to the good alignment state. The two-level termination design tolerates some poorly aligned regions in a long read alignment and helps raise sensitivity.



3.5 Software Implementation

3.5.1 The BWA Front-End

The foundations of BWSL are upon FM-index-based seed searching. To the best of our knowledge, BWA [20] is the best open source tool to construct FM-indices and search sequences with them. Therefore, BWA is adopted as the front-end of the proposed aligner, in charge of the construction of FM-index and seeds alignment. Since BWA is written in C and the rest of the program is implemented with C++, the C++ part is packed as a shared library and included by the C-programmed part. The interface is properly handled by our compiling script.

In BWA, by default the suffix array file (.sa) only records every 32 indices to save disk space. The rest index values can be calculated with traversing BWT. Since in BWSL there are possibly a great amount of alignments for each seed, traversing BWT occupies a large portion of the total execution time. To deal with this problem, the suffix array file in BWSL records all the index values. This effectively increases computing speed but adds disk space and memory usage.

3.5.2 Memory Management

For genome alignment application, the amount of reads to be processed is huge. Also, for the alignment of large-scale genome such as human chromosome the memory usage is high. Under such a condition, the classical memory management method, allocate-then-free, costs a great deal of time on memory compaction.

Memory compaction is an operating-system-level function to manage mem-

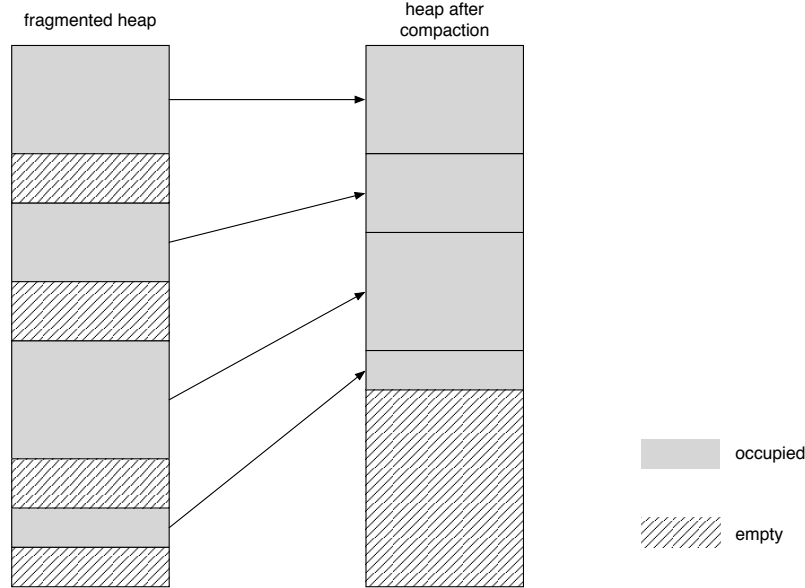


Figure 3.9: Operating system organizes the fragmented heap and clean up continuous memory space using memory compaction.

ory. As illustrated in Figure 3.9, the occupied memory blocks in heap are usually arranged with some space between blocks. Memory compaction is executed when a task is trying to allocate a continuous memory space but there is no such space in heap. The fragmented occupied blocks are put together and thus free blocks are left together. A compaction process is composed of freeing and migrating memory. If a program continuously calls memory compaction, the sum of memory freeing and migrating time may cost more than half of the total time.

BWSL resolves this issue by memory pre-allocation. For variables need be allocated dynamically, a given space is initialized at the starting of the program. When a task is finished, the memory space is not deleted in the heap. Instead, a operation named reset is called as shown in the following codes:

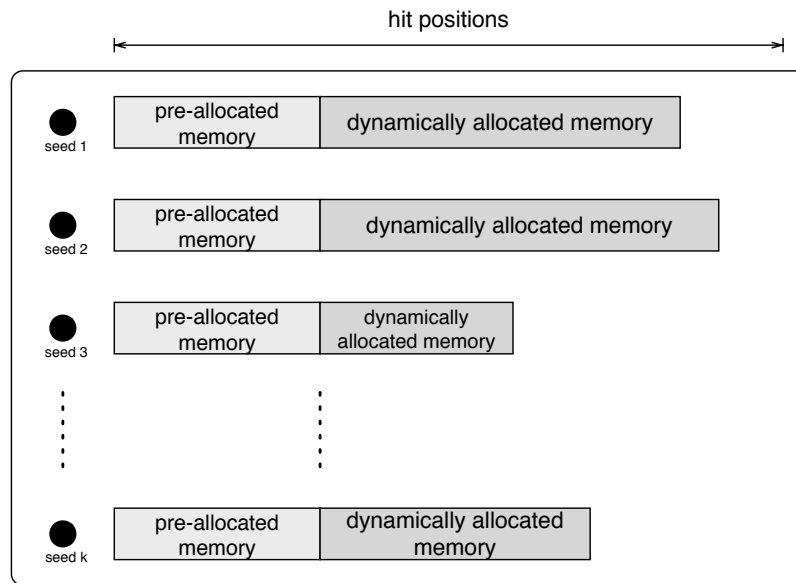
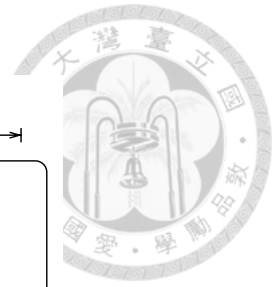


Figure 3.10: The hit positions are stored with a hybrid memory allocation method.

```
typedef struct{
    int *array;
    size_t used;
    size_t size;
} array;
```

```
void freeArray(array *a){
    free(a->array);
    a->array = NULL;
    a->used = a->size = 0; }
void resetArray(array *a){
    a->used = 0; }
```

Memory pre-allocation avoids memory compaction and reduces computing time. However, for variables with a wide range of size, pre-allocating has a huge memory overhead. For example, in BWSL each read is split into short seeds and the seeds are then aligned to the reference. The number of alignments of each seed (hits) varies greatly. It is possible that the occurrence of hits for a seed located in a repetitive region outnumbers that of a normal seed by more than a hundred times. This issue is solved by using a hybrid method, as shown in Figure 3.10. In this hybrid method, a fixed-size memory is pre-allocated. This space is set to one fifth of the maximum hit number by default. Also, dynamically allocated memory space is used to store hits when the number of hits exceeds the size of the pre-allocated memory. The hybrid method saves memory compaction time and

avoids using too much memory space.







Chapter 4

Experiments

4.1 Overview of the Results of BWSL

BWSL is designed as an effective sequencer to align reads to correct positions and generate near optimal alignments. In this chapter, the alignment results of BWSL is analyzed and compared to that of GraphMap [8]. GraphMap is a well-designed sequence aligner for nanopore reads, the details of which is described in Section 2.3.3. It is the aligner with the highest sensitivity by 2016 to the best of our knowledge. To evaluate the alignment results generated by different tools, clear definitions for mapping accuracy (sensitivity) and base-level accuracy are necessary. However, there are no universal definitions for both measurements in genetic applications. Here we discuss common measurements of sensitivity and base-level accuracy and introduce the measurements adopted by BWSL.

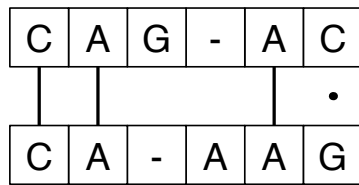
First, to precisely measure sensitivity, well-defined benchmarking tools such as Rabema [32] provide unified approaches to determine mapping quality. However, the tools often need an extra full-sensitivity mapper to generate golden results, which is not efficient in time. Also, the scoring mechanisms for the extra mappers are not flexible enough. Therefore, easier definitions are currently adopted by most sequence aligners. GraphMap defines the mapping of one read correct if its position is less than or equal to ± 50 bases from the golden posi-

tion recorded by the simulator. Similar definitions are also adopted by popular sequence aligners such as BWA-MEM [11], which uses a range of ± 20 bases. Here, we adopt the definition used by GraphMap because both tools focus on long nanopore reads.

Second, the measurement of base-level accuracy is more complicated. In the original manuscript of GraphMap, a true-positive base is defined as “*a base was considered correctly aligned if it was placed in exactly the same position as it was simulated from*” [8]. However, the simulation report does not always provide the mathematically optimal alignment of sequences. Figure 4.1a shows an example taken from a real simulation report. The alignment between “G-” and “-A” is impossible for an aligner to figure out, and is not the mathematical optimal alignment. Instead, Figure 4.1b shows the optimal alignment of the two sequences. Therefore, the definition used by GraphMap does not represent the real base-level accuracy of bases mapping. Instead, average alignment score is used as the primary base-level accuracy indicator. In this chapter, the term “score” refers to average alignment score unless stated otherwise. Average alignment score is defined as:

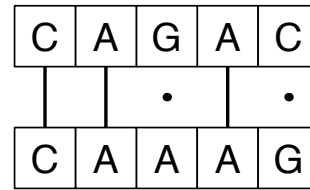
$$\text{Average alignment score} = \frac{\text{Total alignment score}}{\text{Total alignment length}} \quad (4.1)$$

We believe that the evaluation with average alignment score is more suitable to compare scoring-table-based aligners because the tracing-back process (see Section 2.1) depends on alignment score. In spite of the fact that the optimal score need be calculated with a complete dynamic programming algorithm, a higher average score reflects an alignment closer to optimum.



alignment: MMDIMM, score: -5

(a)



alignment: MMMMM, score: 7

(b)

Figure 4.1: An example shows that the alignment recorded by the simulation report is not always the optimal alignment. Here M, D, I stands for match/mismatch, deletion and insertion, respectively. Alignment score is calculated with the default settings of BWSL. (a) Alignment shown in the simulation report. The alignment score is -5 . (b) Mathematically optimal alignment. The alignment score is 7.

4.2 Alignment Results with Synthetic Data

4.2.1 Synthetic Datasets

A recent evaluation for MinION R9 shows that the sequencing accuracy of 1D reads is improved to higher than 80% [16]. However, the volume of MinION R9 sequencing data is insufficient in comparison to the previous MinION systems. Therefore, we adopt the error profile [8] learnt from R7.3 data with LAST [9], as shown in Table 4.1. Since the quality of R7.3 data (1D: 59%, 2D: 69%) is inferior to current release, we only analyze our results with higher quality 2D parameters in Table 4.1. The synthetic reads are simulated using PBSIM [33].

The reference sequences used in this section is shown in Table 4.2. All the sequences are from frequently-used model organisms. Most of the reference sequences are identical to those adopted by GraphMap to avoid biased comparisons toward BWSL. The simulated depth is set to 20 for shorter reference sequences; it is set to 10 for *C. elegans* and human chromosome 4 references to save experiment time.

Table 4.1: Simulation parameters for ONT reads.

Simulator: PBSIM	1D reads	2D reads
Accuracy mean	0.59	0.69
Accuracy std	0.05	0.09
Accuracy min	0.40	0.40
Length mean	4400	5600
Length std	3900	3500
Length min	50	100
Length max	100000	100000
Error types ratio (mismatch:insertion:deletion)	51:11:38	55:17:28



4.2.2 Evaluation of Sensitivity and Base-Level Accuracy

We evaluate the performance of BWSL and GraphMap with four indices: speed, peak memory usage, sensitivity and average alignment score. Here speed is measured with average runtime for each read, and peak memory usage is measured with resident set size (RSS). RSS refers to the portion of memory actually held in main memory for the process. Table 4.3 shows the comparison on the datasets shown in Table 4.2. Generally, BWSL has better alignment accuracy and GraphMap is more computational efficient. For all five datasets BWSL is slower. It uses less memory for small references but requires more for large ones. BWSL is a more sensitive aligner than GraphMap by 0.24% to 1.19%. If measured with mapping error rate (defined as $1 - \text{sensitivity}$), the results of BWSL is only 37.70% to 91.24% to that of GraphMap. Also, BWSL has higher average alignment scores by 3.1% to 3.9% with the same scoring matrix as GraphMap.

Table 4.4 shows the aligning characteristics of BWSL and GraphMap. Since GraphMap tends to align sequences with a great number of matches, its identity is higher than that of BWSL. On the other hand, GraphMap has to generate lots

Table 4.2: Five synthetic datasets from 2-kbp to 2-Mbp.

Organism	Strain	Whole genome size (bps)	Number of chromosomes	Simulated depth
<i>N. meningitidis</i>	Z2491	2,184,406	1	20
<i>E. coli</i>	K-12, MG1655	4,641,652	1	20
<i>S. cerevisiae</i>	S288C	12,157,105	17	20
<i>C. elegans</i>	WBcel235	100,286,401	7	10
<i>H. sapiens</i> , Chr4	GRCh38.p7	190,214,555	1	10

of gaps to align the regions with low numbers of matches. This results in its high ratio of gaps and longer alignment lengths. However, the alignment score calculated by GraphMap is lower than that of BWSL, showing that the results of BWSL are closer to optimum. Table 4.5 shows the alignment results of five synthetic datasets generated with chromosome 1 to 5 from *S. cerevisiae* (with whole *S. cerevisiae* genome as reference). The alignment results of the sub-datasets do not show significant correlation with the size of synthetic templates.

4.2.3 Visualization of Alignment Results

To further analyze the results, the SAM [34] files generated by BWSL and GraphMap are visualized with Integrative Genomics Viewer (IGV) [35, 36]. Figure 4.2 illustrates the visualization results for the *E. coli* dataset shown in Table 4.2. The range of the sample regions is 42-kb, which is the maximum visualizable range allowed under our IGV settings. Five regions are selected with an interval of 1,000,000 bases. The visualization results show that generally both tools map reads onto correct positions, but BWSL has much higher sequencing quality. The sequencing depth is 20X for this dataset.

Table 4.3: Comparing the alignment results by BWSL with GraphMap on synthetic ONT 2D datasets.

	Sensitivity	Score	Time (ms)	Peak RSS (MB)
<i>N. meningitidis</i> , 2.2 Mbp, 7,890 reads				
GraphMap	0.9876	2.3278	22.19	431
BWSL	0.9933	2.4060	28.72	166
<i>E. coli</i> , 4.6 Mbp, 16,565 reads				
GraphMap	0.9889	2.3193	23.11	538
BWSL	0.9952	2.3916	38.88	300
<i>S. cerevisiae</i> , 12 Mbp (17 chromosomes), 43,494 reads				
GraphMap	0.9809	2.3342	23.11	981
BWSL	0.9928	2.4084	72.53	712
<i>C. elegans</i> , 100 Mbp (7 chromosomes), 179,086 reads				
GraphMap	0.9726	2.3434	84.37	3,065
BWSL	0.9750	2.4346	365.23	4,371
<i>H. sapiens</i> , Chr4, 190 Mbp, 248,068 reads				
GraphMap	0.9755	2.3302	161.5	4,677
BWSL	0.9838	2.4175	1,492.23	6,780

4.2.4 Variant Analysis of Alignment Results

The sequencing results are further translated to pileup format with SAMtools [34]. After that, BCFtools [37] is used to call variants from the data. The command we use is as follows:

```
samtools mpileup -ugf ref.fa in.sam |
bcftools call -vm0 z -o out.vcf.gz
```

The alignment results of *E. coli* and *C. elegans* datasets are analyzed, as shown in Table 4.6. Since the reads are simulated from reference, a lower number of variants called refers to better alignment quality. In the *E. coli* (20X depth)

Table 4.4: The comparison of characteristics of alignments generated by BWSL and GraphMap with 248,068 synthetic reads generated with *H. sapiens* chromosome 4.

	Length	Identity	Gaps	Hamming distance	Total score
GraphMap	5,940.72	0.7554	0.1214	1,453.13	13,843.06
BWSL	5,883.96	0.7505	0.0893	1,467.77	14,224.47

dataset, both tools generate high-quality results and BWSL slightly outperforms GraphMap (number of variants: 2 vs. 11). To further compare the alignment quality of the two tools, a lower-depth dataset (*C. elegans*, 10X) is tested. In this experiment, the number of variants called with the BWSL alignments is less than one hundredth to that of GraphMap, showing the great alignment quality of the proposed method.

Also, the aligners' abilities to detect variants are tested and compared with *C. elegans* dataset, as shown in Table 4.7. Point mutations (86,066) are added to the reference and synthetic reads with mutations are then generated. The number of point mutations follows the average proportion of nucleotide differences between humans, one in 1,000 to 1,500 [38]. Here a true positive is defined as calling a mutation with perfectly identical position, reference nucleotide type and alternative nucleotide type. BWSL greatly outperforms GraphMap in terms of precision (92.44% to 21.01%), but both aligners call a small number of variants with less than 1% sensitivity. This results show that there is still great room for improvement for the variant calling abilities of current nanopore aligners.

4.3 Parameters Selection Guidelines

In this section, important parameters used in BWSL are analyzed and discussed and the parameters selection guidelines are provided. We generate 5,479 reads

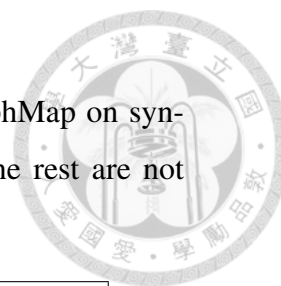


Table 4.5: Comparing the alignment results by BWSL with GraphMap on synthetic ONT 2D datasets for *S. cerevisiae* chromosome 1 to 5 (the rest are not listed here).

	Sensitivity	Score	Time (ms)	Peak RSS (MB)
<i>S. cerevisiae</i> , Chr1, 230 Kbp				
GraphMap	0.9794	2.3253	27.96	610
BWSL	0.9915	2.41	82.27	894
<i>S. cerevisiae</i> , Chr2, 813 Kbp				
GraphMap	0.9850	2.3407	28.75	521
BWSL	0.9867	2.4212	73.49	970
<i>S. cerevisiae</i> , Chr3, 317 Kbp				
GraphMap	0.9710	2.322	28.41	495
BWSL	0.9833	2.401	100.65	910
<i>S. cerevisiae</i> , Chr4, 1.5 Mbp				
GraphMap	0.9788	2.3287	28.72	551
BWSL	0.9867	2.4168	86.39	1,012
<i>S. cerevisiae</i> , Chr5, 577 kbp				
GraphMap	0.9899	2.3313	27.82	506
BWSL	0.9880	2.4179	74.77	952

from chromosome 4 of *S. cerevisiae* and align them to whole *S. cerevisiae* genome (12,157,105-bp). Computing time and peak memory usage is recorded to represent computing efficiency. Sensitivity and average alignment score are shown to evaluate the quality of the results. To analyze the computing time of different functions and stages in BWSL, Linux kernel profiling tool perf is used as a profiler. For simplicity, profiled functions not belonging to any specific stage (e.g. malloc) or occupying less than 1% of the total runtime are not listed in the following comparisons.

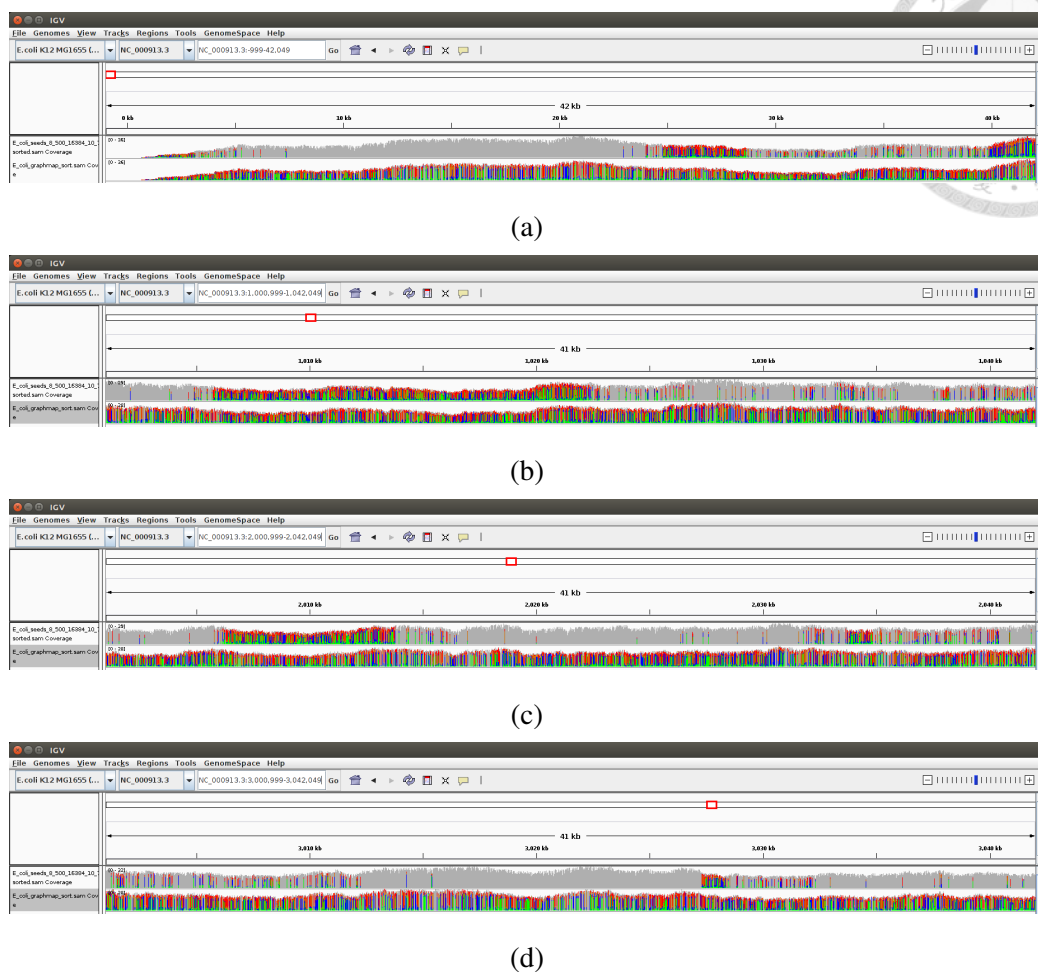


Figure 4.2: Visualize the alignment results of synthetic *E. coli* dataset with IGV from (a) 999 to 42,049 (b) 1,000,999 to 1,042,049 (c) 2,000,999 to 2,042,049 (d) 3,000,999 to 3,042,049 bases.

4.3.1 Max Number of Hits Allowed

The number of hits is associated with the number of expected random hits (N) as shown in Equation 3.2. The expected number of hits is easily to be estimated with a simple probabilistic model. However, some subsequences have a great number of alignments in the genome if they locate in repetitive regions. Such hits are not helpful for finding the correct mapping and have a high cost of both time and memory usage. Therefore, for a seed with number of alignments higher than a threshold called Max Number of Hits Allowed (MNHA), it is discarded.



Table 4.6: The number of variants called by GraphMap and BWSL on two synthetic datasets with different depths.

Dataset	Variants called: BWSL	Variants called: GraphMap
<i>E. coli</i> , 20X	2	11
<i>C. elegans</i> , 10X	21	2,313

Table 4.7: The sensitivity and precision of GraphMap and BWSL results on synthetic *C. elegans* dataset with 86,066 point mutations added.

Dataset:	Sensitivity	Precision
<i>C. elegans</i> , 10X		
BWSL	0.43%	92.44%
GraphMap	0.99%	21.01%

MNHA is defined in proportion to the number of expected random hits, as shown in Equation 4.2, where n, k, c_{MNHA} stand for the length of reference, seed length and MNHA coefficient, respectively. Parameters k, c_{MNHA} are 8 and 10 by default.

$$MNHA = c_{MNHA}(n - k)\left(\frac{1}{4}\right)^k = c_{MNHA}N \quad (4.2)$$

MNHA coefficient is selected empirically. Experiments in Table 4.8 shows the results under different MNHA coefficients. There is no significant difference in average alignment score with varying MNHA coefficients. The computing time and memory usage increase as there are more hits allowed. As for mapping sensitivity, it grows with MNHA when the coefficient is low. However, for the tested dataset the increase stops when c_{MNHA} grows from 10 to 15. This is likely because repetitive hits disrupt the linking efficiency. Our final choice for HMHA is 10 to maximize sensitivity.

Table 4.8: Comparison of different MNHA coefficients on 5,479 synthetic ONT 2D reads from *S. cerevisiae*.

	Num. of hits	Sensitivity	Score	Time (ms)	Peak RSS (MB)
$c_{MNHA} = 1$	4,205.66	0.4563	2.6693	17.13	401
$c_{MNHA} = 5$	65,492.60	0.9839	2.4190	75.71	676
$c_{MNHA} = 10$	97,447.12	0.9867	2.4168	86.39	1,012
$c_{MNHA} = 15$	108,256.73	0.9865	2.4165	90.43	1,294

4.3.2 Number of Chains

The number of chains suggested by Linking Stage is an important factor of the aligning efficiency of BWSL. A sufficient number of chains is important for high mapping sensitivity. When the reference sequence is large, the number of random hits is increasing and the linking results tend to be more noisy, so the number of chains should also increase to absorb the uncertainty. We design Equation 4.3 to handle the uncertainty with taking a square root of the length of reference, where c_{NC} is the number of chains coefficient. The square root function is an empirical selection to slowly raise the number of chains with the increase of reference size.

$$NC = c_{NC} \frac{\sqrt{n}}{1,000} \quad (4.3)$$

Different c_{NC} is tested as shown in Table 4.9. Average alignment score and peak memory usage are indifferent with the change of c_{NC} , and there is a tradeoff between sensitivity and average time. By default, c_{NC} is set to 5. This coefficient can be freely adjusted by users depending on applications. Table 4.10 shows the profiling results with different c_{NC} values. With a higher c_{NC} , Extending Stage occupies a larger proportion of the total computing time, from 41.77% ($c_{NC} = 3$) to 56.80% ($c_{NC} = 8$). The change of c_{NC} has little effect on the runtime of Seeding Stage, Linking Stage and others.

Table 4.9: Comparison of different NC coefficients on 5,479 synthetic ONT 2D reads from *S. cerevisiae*.

	Sensitivity	Score	Time (ms)	Peak RSS (MB)
$c_{NC} = 3$	0.9856	2.4181	73.76	1,010
$c_{NC} = 5$	0.9867	2.4168	86.39	1,012
$c_{NC} = 8$	0.9885	2.4151	102.87	1,011

Table 4.10: Profiling results with different NC coefficients on 5,479 synthetic ONT 2D reads from *S. cerevisiae*.

	Seeding Stage	Linking Stage	Extending Stage	Others and < 1%
$c_{NC} = 3$	22.97% (16.93 ms)	17.22% (12.70 ms)	41.77% (30.81 ms)	18.04% (13.31 ms)
$c_{NC} = 5$	20.16% (17.42 ms)	14.53% (12.55 ms)	48.82% (42.18 ms)	16.49% (14.25 ms)
$c_{NC} = 8$	16.66% (17.14 ms)	12.26% (12.61 ms)	56.80% (58.43 ms)	14.28% (14.69 ms)

4.3.3 The size of the First Tile

BWSL applies an incremental alignment strategy to enhance efficiency in Extending Stage. In this strategy, the design of the first alignment tile is critical because it is a filter to control whether a chain is passed to the rest extending tiles. As discussed in Section 3.4, the first tile is larger in width in comparison to other tiles. By default the width of the first tile is 1,000 bases and that of the rest is 200 bases. This is to deal with the uncertainty of chain positions suggested by Linking Stage. With the same height (200 bases) as other tiles, the first tile occupies a large proportion (more than 40%) of total time, as shown in Table 4.13. Therefore, to raise extending efficiency, it is important to reduce the time cost of the first tile.

First, reducing the width of the first tile (w_{ft}) is a possible solution to enhance

Table 4.11: Comparison of different widths of first Extending tile on 5,479 synthetic ONT 2D reads from *S. cerevisiae*.

	Sensitivity	Score	Time (ms)	Peak RSS (MB)
$w_{ft} = 600$	0.9766	2.4140	74.75	1,003
$w_{ft} = 800$	0.9845	2.4159	83.25	1,006
$w_{ft} = 1,000$	0.9867	2.4168	86.39	1,021
$w_{ft} = 1,200$	0.9869	2.4169	94.49	1,012

Table 4.12: Comparison of different heights of first Extending tile on 5,479 synthetic ONT 2D reads from *S. cerevisiae*.

	Sensitivity	Score	Time (ms)	Peak RSS (MB)
$h_{ft} = 50$	0.9801	2.4209	85.61	1,006
$h_{ft} = 100$	0.9867	2.4168	86.39	1,012
$h_{ft} = 200$	0.9870	2.4168	105.62	1,021

efficiency since it decreases the size of the matrices used in our dynamic programming algorithm. Different w_{ft} values are tested, as shown in Table 4.11. In this experiment, computing time and sequencing sensitivity increase with higher widths, while sequencing sensitivity encounters small marginal effect from $w_{ft} = 1,000$ to $w_{ft} = 1,200$. This is because indels in some reads move the correct alignments away from diagonals. The precise starting positions for such reads are difficult to calculate with our linking algorithm. For nanopore reads, the number of indels is high due to the limited accuracy, raising the uncertainty in Linking Stage. Therefore, a large enough w_{ft} is necessary to capture the uncertainty. Otherwise sensitivity drops even though the rough position of the correct chain is found. The above experiment shows that the uncertainty of chain positions can be absorbed with setting w_{ft} at 1,000 bases. To avoid sacrificing sensitivity, it is not suggested to cut w_{ft} to speed up the algorithm.

Second, different heights of the first tile (h_{ft}) are tested. The filtration thresh-

Table 4.13: Profiling results with different heights of first Extending tile on 5,479 synthetic ONT 2D reads from *S. cerevisiae*.

	Extending Stage: first tile	Extending Stage: secondary tiles	Others and < 1%
$h_{ft} = 50$	14.11% (12.08 ms)	33.19% (28.41 ms)	52.69% (45.11 ms)
$h_{ft} = 100$	27.24% (23.53 ms)	21.58% (18.64 ms)	51.18% (44.21 ms)
$h_{ft} = 200$	44.71% (47.22 ms)	14.04% (14.83 ms)	41.25% (43.57 ms)

olds are set with Equation 4.4:

$$th_{ft} = 0.8h_{ft} \quad (4.4)$$

The experimental and profiling results are shown in Tables 4.12 and 4.13. It is shown that setting h_{ft} to 100 bases has a sensitivity close to that of 200 bases and has a speedup of 18%. If reducing h_{ft} further to 50 bases, there is a significant drop of sensitivity. Therefore, by default h_{ft} is set to 100 bases for the high sensitivity and speed.

The reason accounting for low sensitivity at 50-base-height is that the alignment results generated with a very shallow tile are possibly far from optimal. With error propagation, the extension of a correct chain might be terminated early and sensitivity drops. Also, the computing time under 50-base-height is close to that of 100-base-height. The reason is that the filtration efficiency of the first tile degrades with decreasing height. Though the filtration threshold is set with a universal formula (Equation 4.4), a shorter alignment length makes the filter more vulnerable to the randomness in reads. With lower filtration efficiency of the first tile, more secondary tiles are used for extending. It is shown that under 50-base-height the time cost of secondary tiles is relatively higher in the profiling results.



Chapter 5

Conclusions and Future Work

5.1 Conclusions

In this thesis, Burrows-Wheeler-transform-based aligner with Seeding and Linking (BWSL) is proposed to map noisy long reads accurately and generate high-quality alignments.

BWSL uses a three-stage architecture—seeding, linking and extending—to effectively sample seeds from a read, estimate read position and generate high-quality alignment. Novel algorithms such as Q-FHPS and dynamic extending filters are designed for efficient computing.

Compared to state-of-the-art aligners on synthetic datasets, BWSL outperforms all others in terms of sensitivity. The mapping error rates of BWSL are only by 37.70% to 99.24% to those of the next best mapper, GraphMap. Also, the alignment quality of BWSL outperforms GraphMap with the number of false positives only one hundredth (21 vs. 2,313) in 10X synthetic *C. elegans* dataset. In conclusion, BWSL is a powerful tool to efficiently process long reads with relatively low accuracy and enables further applications based on such reads in the future.

5.2 Future Work

First, despite being an aligner with high aligning quality, BWSL has relatively lower computational efficiency. Profiling results show that Extending Stage occupies a large proportion of BWSL's runtime. Also, BWSL samples a great number of seeds for high sensitivity. However, this strategy generates a lot of random hits, which use large memory. If more accurate seeding and chain selection algorithms are designed, the computational efficiency is likely to be improved.

Second, BWSL adopts reads globally to the reference. Because of the limitation of tile size, it is difficult for our algorithm to align reads with long-range variations. Applying new dynamic programming strategies or using local alignment algorithms are possible solutions to this issue.

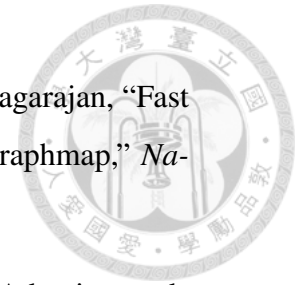
Last, some experiments are not tested in this thesis because of the lack of time and computing resources. For example, entire human genome is not tested with BWSL because of the limited memory of our machines. This can be solved with using a more powerful computing platform such as Amazon Web Services (AWS) or improving our program with methods mentioned above.





References

- [1] F. Sanger, S. Nicklen, and A. R. Coulson, "Dna sequencing with chain-terminating inhibitors," *Proceedings of the national academy of sciences*, vol. 74, no. 12, pp. 5463–5467, 1977.
- [2] (2016) The cost of sequencing a human genome. National Institute of Health. [Online]. Available: <https://www.genome.gov/sequencingcosts/>
- [3] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of molecular biology*, vol. 48, no. 3, pp. 443–453, 1970.
- [4] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *Journal of molecular biology*, vol. 147, no. 1, pp. 195–197, 1981.
- [5] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of molecular biology*, vol. 215, no. 3, pp. 403–410, 1990.
- [6] F. J. Sedlazeck, P. Rescheneder, and A. Von Haeseler, "Nextgenmap: fast and accurate read mapping in highly polymorphic genomes," *Bioinformatics*, p. btt468, 2013.
- [7] B. Liu, H. Guo, M. Brudno, and Y. Wang, "debga: read alignment with debruijn graph-based seed and extension," *Bioinformatics*, p. btw371, 2016.




- [8] I. Sović, M. Šikić, A. Wilm, S. N. Fenlon, S. Chen, and N. Nagarajan, “Fast and sensitive mapping of nanopore sequencing reads with graphmap,” *Nature communications*, vol. 7, 2016.
- [9] S. M. Kiełbasa, R. Wan, K. Sato, P. Horton, and M. C. Frith, “Adaptive seeds tame genomic sequence comparison,” *Genome research*, vol. 21, no. 3, pp. 487–493, 2011.
- [10] B. Langmead and S. L. Salzberg, “Fast gapped-read alignment with bowtie 2,” *Nature methods*, vol. 9, no. 4, pp. 357–359, 2012.
- [11] H. Li, “Aligning sequence reads, clone sequences and assembly contigs with bwa-mem,” *arXiv preprint arXiv:1303.3997*, 2013.
- [12] S. Goodwin, J. D. McPherson, and W. R. McCombie, “Coming of age: ten years of next-generation sequencing technologies,” *Nature Reviews Genetics*, vol. 17, no. 6, pp. 333–351, 2016.
- [13] T. J. Treangen and S. L. Salzberg, “Repetitive dna and next-generation sequencing: computational challenges and solutions,” *Nature Reviews Genetics*, vol. 13, no. 1, pp. 36–46, 2012.
- [14] D. J. Munroe and T. J. Harris, “Third-generation sequencing fireworks at marco island: advances in sequencing platforms promise to make this technology more accessible,” *Nature biotechnology*, vol. 28, no. 5, pp. 426–429, 2010.
- [15] O. N. Technologies. (2016) About minion. [Online]. Available: <https://nanoporetech.com/products/minion>
- [16] N. Loman. (2016) Nanopore r9 rapid run data release. [Online]. Available: http://lab.loman.net/2016/07/30/nanopore-r9-data-release/#disqus_thread
- [17] O. N. Technologies. (2016) no thanks, ive already got one. [Online]. Available: <https://www.youtube.com/watch?v=nizGyutn6v4>



- [18] M. Burrows and D. J. Wheeler, “A block-sorting lossless data compression algorithm,” 1994.
- [19] P. Ferragina and G. Manzini, “Opportunistic data structures with applications,” in *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*. IEEE, 2000, pp. 390–398.
- [20] H. Li and R. Durbin, “Fast and accurate short read alignment with burrows-wheeler transform,” *Bioinformatics*, vol. 25, no. 14, pp. 1754–1760, 2009.
- [21] D. Okanohara and K. Sadakane, “A linear-time burrows-wheeler transform using induced sorting,” in *International Symposium on String Processing and Information Retrieval*. Springer, 2009, pp. 90–101.
- [22] P. Ferragina, T. Gagie, and G. Manzini, “Lightweight data indexing and compression in external memory,” *Algorithmica*, vol. 63, no. 3, pp. 707–730, 2012.
- [23] H. Li, “Fast construction of fm-index for long sequence reads,” *Bioinformatics*, p. btu541, 2014.
- [24] H. Li, J. Ruan, and R. Durbin, “Mapping short dna sequencing reads and calling variants using mapping quality scores,” *Genome research*, vol. 18, no. 11, pp. 1851–1858, 2008.
- [25] H. Li and R. Durbin, “Fast and accurate long-read alignment with burrows-wheeler transform,” *Bioinformatics*, vol. 26, no. 5, pp. 589–595, 2010.
- [26] H. Li, “Exploring single-sample snp and indel calling with whole-genome de novo assembly,” *Bioinformatics*, vol. 28, no. 14, pp. 1838–1844, 2012.
- [27] S. Burkhardt and J. Kärkkäinen, “One-gapped q-gram filters for levenstein distance,” in *Annual Symposium on Combinatorial Pattern Matching*. Springer, 2002, pp. 225–234.



- [28] G. Benson, A. Levy, and B. R. Shalom, “Longest common subsequence in k length substrings,” in *International Conference on Similarity Search and Applications*. Springer, 2013, pp. 257–265.
- [29] G. Myers, “A fast bit-vector algorithm for approximate string matching based on dynamic programming,” *Journal of the ACM (JACM)*, vol. 46, no. 3, pp. 395–415, 1999.
- [30] Y. Turakhia, K. J. Zheng, G. Bejerano, and W. J. Dally, “Darwin: A hardware-acceleration framework for genomic sequence alignment,” *bioRxiv*, p. 092171, 2017.
- [31] T.-Y. Yeh and Y. N. Patt, “Two-level adaptive training branch prediction,” in *Proceedings of the 24th annual international symposium on Microarchitecture*. ACM, 1991, pp. 51–61.
- [32] M. Holtgrewe, A.-K. Emde, D. Weese, and K. Reinert, “A novel and well-defined benchmarking method for second generation read mapping,” *BMC bioinformatics*, vol. 12, no. 1, p. 210, 2011.
- [33] Y. Ono, K. Asai, and M. Hamada, “Pbsim: Pacbio reads simulator toward accurate genome assembly,” *Bioinformatics*, vol. 29, no. 1, pp. 119–121, 2013.
- [34] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, R. Durbin *et al.*, “The sequence alignment/map format and samtools,” *Bioinformatics*, vol. 25, no. 16, pp. 2078–2079, 2009.
- [35] H. Thorvaldsdóttir, J. T. Robinson, and J. P. Mesirov, “Integrative genomics viewer (igv): high-performance genomics data visualization and exploration,” *Briefings in bioinformatics*, vol. 14, no. 2, pp. 178–192, 2013.

- 
- [36] J. T. Robinson, H. Thorvaldsdóttir, W. Winckler, M. Guttman, E. S. Lander, G. Getz, and J. P. Mesirov, “Integrative genomics viewer,” *Nature biotechnology*, vol. 29, no. 1, pp. 24–26, 2011.
- [37] H. Li, “A statistical framework for snp calling, mutation discovery, association mapping and population genetical parameter estimation from sequencing data,” *Bioinformatics*, vol. 27, no. 21, pp. 2987–2993, 2011.
- [38] L. B. Jorde and S. P. Wooding, “Genetic variation, classification and ‘race’,” *Nature genetics*, vol. 36, pp. S28–S33, 2004.