Genomic variation detection using dynamic programming methods

Author: Mengyao Zhao

Boston College

The Graduate School of Arts and Sciences

Department of Biology

GENOMIC VARIATION DETECTION USING DYNAMIC PROGRAMMING METHODS

a dissertation

by

MENGYAO ZHAO

submitted in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Aug 2014

Thesis advisor: Professor Gabor T. Marth MENGYAO ZHAO

GENOMIC VARIATION DETECTION USING DYNAMIC PROGRAMMING METHODS

Abstract

Background

Due to the rapid development and application of next generation sequencing (NGS) techniques, large amounts of NGS data have become available for genome-related biological research, such as population genetics, evolutionary research, and genome wide association studies. A crucial step of these genome-related studies is the detection of genomic variation between different species and individuals. Current approaches for the detection of genomic variation can be classified into alignment-based variation detection and assembly-based variation detection. Due to the limitation of current NGS read length, alignment-based variation detection remains the mainstream approach.

The Smith-Waterman algorithm, which produces the optimal pairwise alignment between two sequences, is frequently used as a key component of fast heuristic read mapping and variation detection tools for next-generation sequencing data. Though various fast Smith-Waterman implementations are developed, they are either designed as monolithic protein database searching tools, which do not return detailed alignment, or they are embedded into other tools. These issues make reusing these efficient Smith-Waterman implementations impractical.

After the alignment step in the traditional variation detection pipeline, the afterward variation detection using pileup data and the Bayesian model is also facing great challenges especially from low-complexity genomic regions. Sequencing errors and misalignment problems still influence variation detection (especially INDEL detection) a lot. The accuracy of genomic variation detection still needs to be improved, especially when we work on low-complexity genomic regions and low-quality sequencing data.

Results

To facilitate easy integration of the fast Single-Instruction-Multiple-Data Smith-Waterman algorithm into third-party software, we wrote a C/C++ library, which extends Farrar's Striped Smith-Waterman (SSW) to return alignment information in addition to the optimal Smith-Waterman score. In this library we developed a new method to generate the full optimal alignment results and a suboptimal score in linear space at little cost of efficiency. This improvement makes the fast Single-Instruction-Multiple-Data Smith-Waterman become really useful in genomic applications. SSW is available both as a C/C++ software library, as well as a stand-alone alignment tool at: https://github.com/mengyao/Complete-Striped-Smith-Waterman-Library. The SSW library has been used in the primary read mapping tool MOSAIK, the split-read mapping program SCISSORS, the MEI detector TANGRAM, and the read-overlap graph generation program RZMBLR. The speeds of the mentioned software are improved significantly by replacing their ordinary Smith-Waterman or banded Smith-Waterman module with the SSW Library.

To improve the accuracy of genomic variation detection, especially in low-complexity genomic regions and on low-quality sequencing data, we developed PHV, a genomic variation detection tool based on the profile hidden Markov model. PHV also demonstrates a novel PHMM application in the genomic research field. The banded PHMM algorithms used in PHV make it a very fast whole-genome variation detection tool based on the HMM method. The comparison of PHV to GATK, Samtools and Freebayes for detecting variation from both simulated data and real data shows PHV has good potential for dealing with sequencing errors and misalignments. PHV also successfully detects a 49 bp long deletion that is totally misaligned by the mapping tool, and neglected by GATK and Samtools.

Thesis advisor: Professor Gabor T. Marth MENGYAO ZHAO

Conclusion

The efforts made in this thesis are very meaningful for methodology development in studies of genomic variation detection. The two novel algorithms stated here will also inspire future work in NGS data analysis.

Contents

Listing of figures

Listing of tables

Acknowledgments

I am sincerely indebted to my mentor Prof. Gabor T. Marth for his patient guidance and continuous support to my research projects. Though I was experienced with biological data processing, I have never done a software development project alone before I entered Prof. Gabor T. Marth's lab. In order to become a professional computational biologist, I aimed to gain independent software development ability during my Ph.D. study. Prof. Gabor T. Marth helped me achieve this goal. He is a professor who trusts students' work and respects our own choices and project designs. The freedom he gave me allowed me enough time and space to try out my own ideas, learn what I am interested in and explore the area that I have never touched before. Under Prof. Gabor T. Marth's supervision, I completed the development of two solid tools, completed my Ph.D. study and gained confidence to expand on my academic development.

I am greatly grateful to my colleagues and fridends in our lab. They gave me smooth coorperations, inspiriting suggestions and warm hearted encouragements. Without them, my Ph.D. study can be miserable and may not progress continuously and finish successfully.

I deeply thank my committee members for their guidance on my research and career development directions. Though we only formally met one time each year, their influence to me is critical, the benefit I got from their insightful suggestions is tremendous.

Especially, I acknowledge the most practical help from our research scientist Wan-Ping Lee. She helped me overcome the difficulties during the progress of my projects, encouraged me when I felt frustrated with my work, and contributed a lot to my publications. I got precious supports from her both academically and spiritually.

At last, I would like to express my eternal thanks to my families. They always show their understanding and love to me. Especially my husband Heng Li and my daughter Jacey Li, who live together with me and gave me the most support whenever I need them. I may not have completed my Ph.D. work without their love. I also would like to take this chance to thank my parents Zhiyun Zhang and Liansheng Zhao. They cared about my Ph.D. study and my life all the time, even if they lived far away from me. I forever appreciate all my family members and my relatives, who warmed my heart and lightened my life during the past years of my memorable Ph.D. life.

1 Introduction

1.1 Next generation sequencing (NGS)

Tracking the history of the development of the natural sciences, the wide and deep application of mathematical methods and quantitative representations and analysis in an academic discipline usually mark the maturation of it. Since the DNA double helix structure was discovered in 1953, people's efforts on this digital representation of life began to drive the most complex and fantastic field of scientific studies, biology, into a new era. We are living in such an exciting time:

1953 - The DNA double helix structure was discovered.[19]

1972 - Recombinant DNA technique was developed. With this technique, the accessible DNA samples for sequencing were not limited to bacteriophage or virus any more. Many interesting DNA fragments became possible to be isolated and sequenced.

1977 - The complete DNA genome of bacteriophage ϕX174 was sequenced. This 5.4 Kilo base pairs (Kbp) genome was the first complete genome to be sequenced.[31]

The same year, the article "DNA sequencing with chain-terminating inhibitors" by Frederick Sanger was published.[105]

1983 - The phage λ genome (48.5 Kbp) was reported.[39]

1984 - The complete genome of Epstein-Barr virus (170 Kbp) was deciphered.[7]

1987 - The first automatic sequencing machine, ABI370, was produced and put on the market by Applied Biosystems.

1995 - The complete genome of the bacterium *Haemophilus influenzae* (about 1 Mbp) was published by scientists at the Institute for Genomic Research. *Haemophilus influenzae* is the first whole-genome-sequenced free-living organism, and its sequencing was the first application of the whole-genome shot-gun sequencing method.[30]

1996 - The method of pyrosequencing was published by Pal Nyren and Mostafa Ronaghi at the Royal Institute of Technology.[100]

1997 - The first genome of *Saccharomyces cerevisiae* (12 Mbp) was reported. This was the first eukaryotic genome to be sequenced.[47,22,21,109,50,16,32,51,9,94,89,92,10,23,11,28]

1998 - The complete genome of *Caenorhabditis elegans* was reported. This was the first metazoan genome to be sequenced.[12] The same year, "Phred" was published by Phil Green and Brent Ewing for DNA data analysis.[26]

2001 - The draft sequence of the *Homo sapiens* genome (about 3,000 Mbp) was published.[56,111]

2005 - The first commercially available next generation sequencing (NGS) machine was developed by Roche 454.

...

Since the parallelized version of the pyrosequecing technique was developed by 454 Life Science, high-throughput sequencing techniques have emerged and evolved at a phenomenal rate. After two decades of domination by the automated Sanger sequencing method (the "first generation" technology), the great demand for accurate, fast and inexpensive genomic information catalyzed the rapid development and marketing of NGS platforms. Various sequencing techniques and companies rose and faded through this evolutionary and competitive time. I extract the most influential ones and I am introducing them here with some necessary brief descriptions.

1.1.1 Roche 454 sequencing

The 454 GS20 was the first successfully marketed NGS instrument. The 454 system amplifies DNA with emulsion PCR. In emulsion PCR, each DNA fragment is attached to a streptavidin bead and amplified to 107 copies in an emulsion droplet. Subsequently, the template-containing beads are transferred onto a picotiter plate. On each plate, hundreds of thousands of pyrosequencing reactions carry on simultaneously. During the pyrosequencing reactions, the DNA template is immobilized, and one kind of dNTPs is added at a time. When dNTPs complement to the template DNA, pyrophosphate (PPi) is released. The PPi transforms into ATP and generates detectable light through chemical reactions.[87] The length of the reads produced by the 454 GS20 was 100-150 bp, the throughput of the machine was 20 Mbp per run.[84] (http://www.roche-applied-science.com/)

454 Life Sciences was acquired by Roche Diagnostics after the GS20 was marketed. (http://en.wikipedia.org/wiki/DNA_sequencing) On May 31, 2007, Roche 454 sequenced the complete genome of James Watson. Later on, this genome was published online in the same year by its owner as the second publicly available personal genome.[115] In 2008, Roche launched the 454 GS FLX Titanium, the read length of which reached 700 bp, sequencing accuracy was 99.9% after filtering, and the throughput was 0.7 GB of data per run.[74] In 2009, Roche further incorporated the GS Junior into the 454 sequencing system, and upgraded the throughput to 14 GB of data per run.[42] (http://www.gsjunior.com/instrument-workflow.php) The advantages of Roche 454 are read length and sequencing speed: one run of the whole sequencing process only takes ten hours. However, due to the high cost of reagents and the high sequencing error rate (especially when homopolymer length is over 6 bp), current usage of the 454 system is already very rare, and the production of their machine has stopped.

1.1.2 Illumina sequencing

In 2006, just one year after the 454 GS20, Solexa released the Genome Analyzer (GA). Solexa's sequencing technique was called "sequencing by synthesis". This technique uses solid-phase amplification to duplicate DNA material: initially, the single-stranded DNA template is primed and extended, then the immobilized template is "bridge amplified" with adjacent primers to form 100-200 million molecular clusters.[87] After amplification, all four kinds of ddNTPs that contain different fluorescent dyes are added to these single-stranded DNA templates. After the complement nucleotide binds to the template, unused ddNTPs are washed off, and the incorporated nucleotides are detected. These steps are repeated until the whole complement sequence of the DNA template is sequenced.

In 2007, when Solexa was purchased by Illumina, the throughput of GA was 1 GB of data per run. In October 2009, GA's throughput was increased to 20 GB of data per run, with a sequencing error rate below 2% after filtering. In early 2010, using the same sequencing strategy as GA, Illumina produced the HiSeq 2000 with a throughput of 200 GB of data per run. Now HiSeq 2000 can output 600 GB of data per run in eight days at a cost of $0.02/million bp, which is the lowest cost for NGS sequencing. In 2011, Illumina further launched MiSeq to target the needs of small laboratories. MiSeq shares similar technologies with HiSeq, and can produce 1.5-2 GB data per run with a raw sequencing error rate of 0.8%.[74] Though the throughput of the MiSeq is lower than that of the HiSeq 2000, it has a much shorter turnaround time. That same year, Illumina further produced the HiSeq 2500, which shows higher throughput than the HiSeq 2000 on Illumina's website, but no technical feature from user is found. The sequencing errors of Illumina data tend to happen at the end of the reads. With the advantages of high sequencing accuracy and low cost, Illumina has dominated the sequencing market.

1.1.3 Sequencing by oligo ligation detection (SOLiD)

Just after Solexa released GA in 2006, Agencourt produced the SOLiD platform. SOLiD uses emulsion PCR to amplify DNA samples, similar to the Roche 454. During sequencing, SOLiD uses "two-base-encoded probes" (1, 2-probes): an oligonucleotide sequence carrying two interrogation bases and a dye. There are 16 possible combinations of the two nucleotides. Each dye associates with four of them. After annealing a universal primer to the DNA template, 1, 2-probes are added. The suitable probe ligates to the DNA template and the color of this probe is imaged. After the dimer signals of the whole DNA sequence are recorded, a universal primer will be annealed to another copy of the same DNA template with one base shift to the left. This initiates another round of the dimer ligation process. This cycle is repeated several times to generate the DNA sequence.

The initial SOLiD reads had a length of 35 bp and the throughput of the machine was 3 GB of data per run. In 2006, SOLiD was purchased by Applied Boisystems (AB), and the first SOLiD system was commercialized. In 2008, AB and Invitrogen merged and formed Life Technologies. In late 2010, Life Technologies released the SOLiD 5500xl, which generates 85 bp long reads with an accuracy of 99.99%, and it has a throughput of 30 GB of data per run. Although the error rate of processed SOLiD data is very low, the raw-data error rate is high. Due to the special two-base sequencing method, the SOLiD's afterward data processing is difficult. Besides the limitation of short read length and high requirement of DNA materials, SOLiD is not widely used at the moment. However, the story of Life Technologies is not finished. I will introduce their new sequencing technique later.

1.1.4 Heliscope sequencing of Helicos Biosciences

In 2008, Helicos Biosciences released the Heliscope Single Molecule Sequencing machine. It first realized the direct sequencing of a certain DNA or RNA molecule, and bypassed the bias caused by genome amplification. In this technique, DNA fragments are tailed with poly-A and attached to a poly-T primer on the surface of a float cell. Billions of such molecules are then parallel sequenced by the "sequencing-by-synthesis" method, which is similar to Illumina's sequencing process except that all ddNTPs are labeled with the same dye and only one kind of ddNTPs is added to the DNA template in each round for imaging.[110]

Not only does the single-molecule sequencing technique avoid PCR bias during DNA amplification and it requires much less DNA sample for sequencing, it also provides many meaningful possibilities to biological research, such as population-scale studies of pan-genomes. Though Heliscope sequencing has not gained success in the market due to the short read length, high sequencing error rate and cost, it began an ideal direction for the development of genomic sequencing techniques.

1.1.5 Single molecule real time (SMRT) sequencing of Pacific Biosciences (PacBio)

Also in 2008, PacBio announced the SMRT sequencing. In this technique, a DNA polymerase binding to a DNA template is immobilized to the bottom of a nano well. The DNA synthesis carries on with four color-fluorescently labeled nucleotides. When the dye-labeled nucleotide binds to the DNA template, the fluorescence signal is recorded and the by-product quickly diffuses away.[87]

SMRT sequencing currently generates reads with a length of 1500 bp on average, with a raw error rate of 12.86%. In 2013, PacBio produced the RS II sequencing system, which can generate reads with a length of over 20 Kbp with a throughput of 100 Mbp per run in two hours. (http://files.pacb.com/pdf/PacBio_RS_II_Brochure.pdf) In order to solve the problem of the high error rate, PacBio uses circular consensus sequencing (CCS) to generate consensus sequences with an accuracy of > 99.999%. The consensus sequences are produced through the following steps: 1) Sequence while the DNA polymerase circularly generates the same DNA fragment several times. 2) Map the reads to a known reference, or *de novo* assemble a new reference and map reads back to that reference. 3) Generate a consensus sequence based on the read alignment. (http://www.pacificbiosciences.com/pdf/Perspective_UnderstandingAccuracySMRTSequencing.pdf) While achieving high accuracy, the consensus read length is reduced to only about 1-2 Kbp. (One long read needs to contain about ten repeats of the same sequence.) (http://flxlexblog.wordpress.com/2013/02/11/applications-for-pacbio-circular-consensus-sequencing/)

The cost of PacBio is about 55 times higher than that of Illumina HiSeq 2000. Though CCS has a high consensus accuracy, while losing the read length advantage, there are now very few users of CCS. However, for PacBio's regular sequencing with the 20 Kbp RS reads, the afterward data processing, such as read mapping or assembly, can be much easier since the problems stemming from repetitive genomic regions can be largely reduced. In 2013, PacBio released the *de novo* assembled *E. Coli* genome from RS reads, which remains the most complete and accurate *E. Coli* assembly to date. This breakthrough success not only comes from the RS reads but also largely from PacBio's new assembly software.[37] We can say the bioinformatics method helped PacBio to become the currently most popular and promising single-molecule sequencing system.

1.1.6 Ion Torrent semiconductor sequencing

In February 2010, the Ion Torrent sequencer was launched by Life Technologies (the company that produced SOLiD). We need to note that Ion Torrent is not a single-molecule sequencing technique. As an extension of 454 sequencing, Ion Torrent uses the same DNA amplification method, emulsion PCR, to prepare a library. At the same time, it adopted a sequencing principle similar to PacBio's. During Ion Torrent sequencing, the DNA polymerase holding a DNA template is fixed at the bottom of a micro-well, but instead of using dye-labeled oligonucleotides, Ion Torrent uses unmodified dNTPs to do DNA synthesis. Only one kind of dNTPs float across the micro-well at a time. When the polymerase incorporates the appropriate nucleotide, a hydrogen ion is released, which changes the pH of the solution. The ion sensor beneath the micro-well detects this change and records type and amount of the incorporated dNTP.[103]

By 2012, the read length of Ion Torrent PGM was 200 bp with a 1.71% raw error rate. The highest throughput of this machine was 1 Gbp per run in two hours.[96] Similar to 454, the length of a homopolymer is likely to me miss-sensed (especially when the homopolymer is longer than 5 nt) because the ion sensor needs to evaluate the amount of incorporated dNTPs. Though the cost of Ion Torrent is much lower than that of 454, the high sequencing error rate makes it difficult to compete with Illumina.

1.1.7 Nanopore sequencing of Oxford Nanopore

In 2005, Oxford Nanopore Technologies began the development of nanopore sequencing, another single-molecule sequencing technique. This is a technique that has not been commercially available. (https://www.nanoporetech.com/about-us/summary) This technique sequences the DNA in real time while the intact DNA polymers pass a protein nanopore such as an alpha-hemolysin pore. The nanopore is embedded in an electronically resistant membrane, to which a voltage is applied. When DNA passes through the nanopore, the ion current is changed. Different types of nucleotide block the current for different amounts of time, so that the DNA sequence can be sensed.

The DNA sequences generated by this technique are claimed to be very long. Earlier this year, the first Oxford Nanopore read was released. It is an 8,476 bp read of *Pseudomonas aeruginosa* strain 910. Compared to PacBio, Oxford Nanopore also provides long single-molecule sequenced reads, and it does not require any modification of the DNA sample during sequencing, so that its price will possibly be much lower than that of PacBio. The Oxford Nanopore sequencing device is the size as a flash disk, and can be connected to a laptop through a USB port to do post-sequencing data analysis. Currently, the main challenge of this technique is the rapid movement of the DNA strand through the nanopore (1 to 5 μs per base), which makes detection difficult. (http://en.wikipedia.org/wiki/Nanopore_sequencing)

Limited by the length of this thesis, Oxford Nanopore is the only sequencing technique that I am introducing here. However, we should be aware that many other new sequencing techniques are under rapid development at the moment.

1.1.8 Features of the main currently available NGS generated reads

Illumina, PacBio and Ion Torrent are the currently predominant platforms in the sequencing competition, which is still fiercely ongoing. The sequencing reads generated by these platforms are the main data that existing popular software are aimed at. Here I summarize the essential features of Illumina, PacBio and Ion Torrent sequenced reads in Table 1.1. These features are highly influential or even determinative to algorithm design and to the construction of mathematical models of variation detection-related software including mapping tools, assembly tools and variation calling tools.

1.1.9 Applications of NGS in biological research and the future that we should prepare for

Along with the rapid development of NGS sequencing technology, the available genomic data are increasing at an unprecedented, exploding rate. Both the speed of sequencing cost reduc-

Table 1.1: Essential features of Illumina, PacBio and Ion Torrent sequenced reads.[96] These features are important to the design of NGS data analyzing software.

	Illumina HiSeq 2000	Illumina MiSeq	PacBio RS	Ion Torrent PGM
raw error rate	0.26%	0.8%	12.86%	1.71%
read length	up to 150 bp	up to 150 bp	1500 bp in average	200 bp
paired reads	yes	yes	no	yes
insert size between the paired reads	up to 700 bp	up to 700 bp	N/A	up to 250 bp

tion and of GenBank growth are faster than Moore's law. As a result, a lot of biological research is advanced or even created by this big change. For example, whole-genome or whole-exome sequencing greatly helped research in population genetics, evolutionary studies, personal medicine, cancer genomics, metagenomics, and ecological microbiology. The large amounts of data also brought out new datadriven research branches such as genome wide association studies (GWAS). RNA sequencing (RNA-seq) uses NGS techniques to reveal the presence and quantity of RNAs in cells at any given moment, which has replaced microarray to become the main method to evaluate changes in gene expression, alternative gene transcription, and post transcriptional modifications at a higher resolution. Chromatin immunoprecipitation sequencing (ChIP-seq) uses NGS to identify DNA sequences at protein binding sites, which provides a direct way to study gene regulation by protein interactions.[49] Now, both RNA-seq and ChIP-seq are essential tools for understanding genotype-phenotype relationships and the mechanisms of many biological processes and diseases.

In a word, the increased availability of genomic data and the revolution of sequencing techniques are changing biological research in many aspects. As millions of billions of whole-genome sequences of different individuals and different species will become available in the near future, a lot of research in evolution, medicine and ecology will become possible; at the same time, the challenge to find bioinformatics solutions will become more enormous than ever. In another aspect, single-molecule sequencing, and sequencing techniques that produce longer reads at a lower cost are the main directions of the emerging third generation sequencing. With the availability of cheap, long reads, of genomic variation detection and of representation and storage methods, the methods for evaluating gene expression and for downstream data analysis may all change fundamentally. We should prepare for such changes not only mentally but also realistically in our current work.

1.2 Genomic sequence alignment algorithms

The phenomenal evolution of sequencing technology shifts the bottleneck of NGS studies towards computational data processing, analysis and management steps. One major goal of NGS data processing is to identify genomic variation and to link it to biological questions. Though different approaches can be used to achieve this goal, genomic sequence alignment remains a key step in most currently available practical data-processing pipelines. The definition of "sequence alignment" in Wikipedia is "a method to arrange DNA, RNA or protein sequences to identify the regions of similarity."[88] However, for an algorithm developer it is important to keep in mind that the final purpose of sequence alignment is the detection of variation.

1.2.1 When computer memory was expensive and limited...

When talking about sequence alignment, we cannot avoid the topic of the basic local alignment searching tool (BLAST). The BLAST algorithm and its implementation were published by Altschul *et al.* in 1990.[5] It is one of the most broadly used algorithms in influential programs for sequence search. As a heuristic algorithm, BLAST cannot guarantee the optimal alignment, but it is much faster than the Smith-Waterman (SW) algorithm.[107,34] We can regard BLAST as one of the first popular genomic mapping tools.

The principle of the BLAST algorithm is as follows:

Step 1. Make a k-mer ("seed") list from the query sequence. Take k = 3 as example, if a DNA sequence is ACGTAG, the k-mer list is: ACG, CGT, GTA, TAG. Generally, k is chosen as 11 for genomic sequence alignment.

Step 2. Organize the k-mer list in a hash table.

Step 3. Scan the reference sequences for exact matches to the k-mer list.

Step 4. Extend the exact matches in left and right directions to form longer exact match segments.

Step 5. For the exact match segments that are longer than a threshold, further extend the alignment to the right. This extension allows mismatches and gaps using an alignment algorithm similar to SW.

Step 6. Evaluate the significance of all the gap-allowed alignments and return all alignments whose significance passes a threshold.

As one of the earliest alignment algorithms, the principle of BLAST (hashing the query sequence, searching for exact matches in the reference and extending the matches with approximate string matching) has deeply influenced the development of alignment algorithms until now. Here, we only introduce two kinds of current mainstream alignment algorithms that may be further improved and continually applied to future bioinformatics studies.

1.2.2 Alignment algorithms using hashing methods

Following BLAST, the hashing method was developed in two aspects: improvement of the seeding, and the change from hashing reads to hashing the reference.

In 2002, Ma *et al.* discovered that using seeds with internal mismatches (space seed) allows for increased sensitivity.[82,68,67] The space seed was first used by Eland[18], then it was adopted and improved by several other aligners such as MAQ[66] and SeqMap[48] and soon became the most popular approach for short read alignments. For sequence alignments, approximate matching is always preferred to exact matching due to the existence of genomic variation and sequencing errors. For this reason, a better seeding method should tolerate more variations. To further extend the space seed, a seeding method that allows gaps was proposed by Cao *et al.* in 2005.[13] This kind of seed actually contains multiple short exact matches, which helps a lot in long read alignment. It was used in SHRiMP[101] and RazerS[113,112].

On the other hand, the rapid development of computer science increases the availability of cheap, large memory. This makes hashing a long reference sequence become realistic. Compared to hashing reads, hashing the reference requires less calculation in the indexing step. This is due to the high abundance of read sequences compared to the reference sequence, depending on the sequencing depth. Taking advantage of this, most modern aligners, such as SOAP[70,71], Mosaik[60], Bfast[40], and Novoalign (http://www.novocraft.com/main/index.php) choose to hash the reference in order to achieve an improvement of speed.

1.2.3 Alignment algorithms using suffix tree, enhanced suffix array and FM-index

In another category of alignment software, the seeding step is done by a suffix tree (or functionally equivalent data structure) instead of a hash table. One problem of the hashing method is that if a substring exists several times in the reference, the alignment needs to be performed for each copy. In contrast, on a tree all identical copies of the substring collapse on a single path, and the alignment needs to be applied to that path once only.[65]

A suffix tree is a compressed tree that contains all the suffixes of a string. In the suffix tree, the suffixes of a certain string are keys; the beginning positions of the suffixes in the original string are values. Figure 1.1 shows the suffix tree of the string "BANANA". Suffix trees enable many efficient string operations, including fast, exact string matching. For example, to match the string "NAN" to "BANANA", it just needs to start from the root of this tree to seek the branches that contain "NAN". In Figure 1.1, the grey path denotes the branches that contain the string "NAN". The number '2' in the grey rectangle tells us the string "NAN" matches to the string "BANANA" from its position 2 (0-based coordinate). With the suffix tree, to do the math for this exact string, string comparison needs to be done only two times at the two oval notes. Aligners using a suffix tree include MUMer[54] and OASIS[86]. In such software, after the

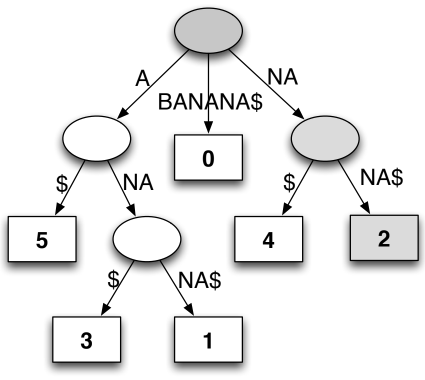


Figure 1.1: Suffix tree of the string "BANANA". '$' marks the ending of the string. The strings on the branches are the suffixes of "BANANA". The numbers in the rectangle notes are the starting position of the corresponding suffix in the string "BANANA". The coordinates of "BANANA" are 0-based.

exact string-matching step, the entire query sequence is aligned to the candidate tree path by a dynamic programming method.

The construction of a suffix tree can be completed in linear time according to the length of its corresponding string. A classical algorithm for suffix tree construction is Ukkonen's algorithm.[35] To state the basic idea of this algorithm, I need to distinguish two definitions:

1. Explicit suffix tree (suffix tree): All suffixes of the corresponding string are represented by the nodes in the explicit tree. The suffixes are labeled on the branches, and are ended with an additional ending mark '$'.
2. Implicit suffix tree: Remove the ending mark '$' from the explicit suffix tree. Remove the internal nodes and their daughter branches from the explicit suffix tree, and mark the corresponding suffixes with pointers.

Figure 1.2 shows the "implicit" suffix tree of the suffix tree shown in Figure 1.1.

With the above definitions, we can roughly describe Ukkonen's suffix tree construction algorithm as:

For a string S [1 ... n],

Initialization:

Build an "implicit" suffix tree T from the leftmost residue of the string S[1]. Use a pointer to mark the end of the suffix.

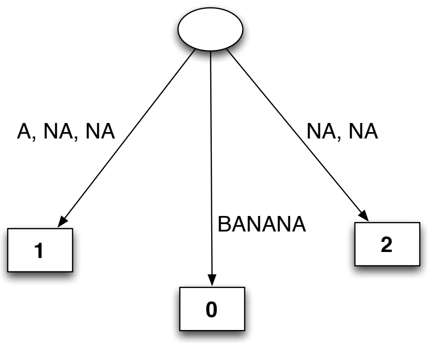


Figure 1.2: The "implicit" suffix tree of the suffix tree shown in Figure 1.1. In this tree, the suffixes represented by the internal leaves in the suffix tree are marked with '. In the computational implementation of the "implicit" suffix tree, such suffixes are marked by pointers for fast accessing.

Recursion: For i = 2, 3, ... n:

> Update the tree T from the "implicit" suffix tree of string S[1 ... i - 1] to that of the string S[1 ... i] with the following rules:
>
> Rule 1: If path S[j ... i-1] (j = 1,... , i - 2) ends at a leaf edge string, concatenate S[i] to the original edge string. Update the pointer to mark S[i].
>
> Rule 2: If path S[j ... i-1] (j = 1,... , i - 2) ends before the last residual of the leaf edge string, and does not continue by S[i], add a new leaf labeled with S[i] at the end of the path. Add a pointer to mark S[i].
>
> Rule 3: If suffix S[j ... i] is already in the tree, mark the end of the suffix with a pointer. (Or update the position of the existing suffix pointer.)

Termination:

> Convert the final "implicit" suffix tree to "explicit" suffix tree by marking each suffix with a node and adding '$' to the end of the suffixes.

For the string "BANANA", the suffix tree construction process is shown in Figure 1.3.

The storage of a tree structure is usually complicated and memory-consuming. A suffix tree takes O(L^2) space, where L is the length of the reference sequence. The large memory requirement limits the usage of this powerful string-matching method to very small genomes. To solve this problem, the enhanced suffix array was derived by Abouelhoda *et al.*[3] The enhanced suffix array emulates the suffix tree to realize the string-matching function. Though it does not sup-

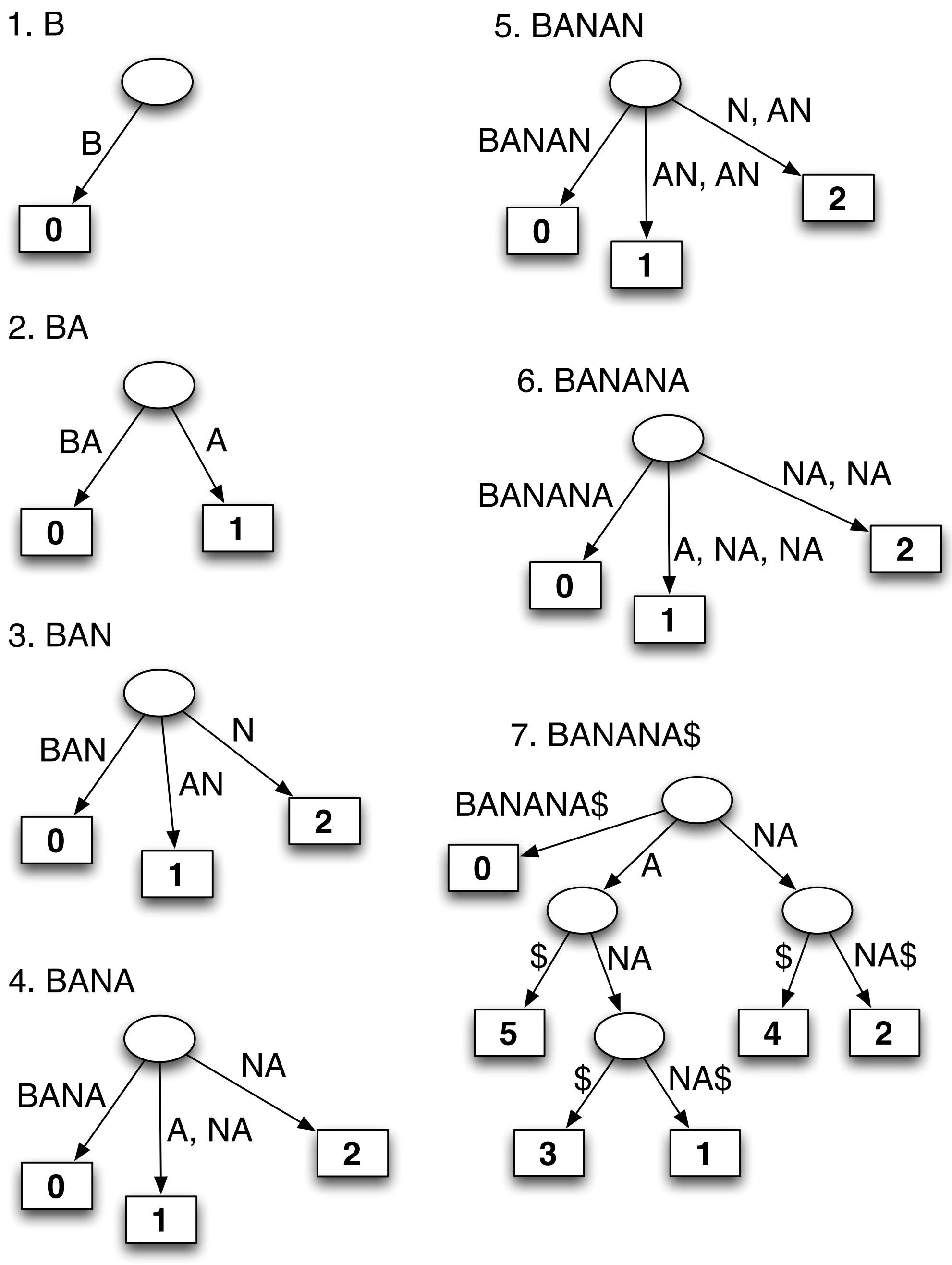


Figure 1.3: The seven steps to construct the suffix tree of the string “BANANA”. Step 1 correlates to the “initialization”; step 2 to 6 correlate to the “recursion”, and step 7 correlates to the “termination” of Ukkonen’s algorithm.

port all the string operations supported by the suffix tree, this data structure reduces memory usage. Vmatch and Segemehl[38] are both based on the enhanced suffix array.

Later on, Ferragnia and Manzini[29] further reduced memory usage of the enhanced suffix array through the FM-index. The FM-index contains the Burrows-Wheeler transformation of the reference, a sampled suffix array and some auxiliary arrays. It keeps the string-matching function of the suffix tree, but uses much less memory (typically 0.5-2 bytes per 1 bp of genomic sequence). Due to this, the FM-index is more widely used compared to the suffix tree or the enhanced suffix array. Aligners based on the FM-index include BWT-SW[55], BWA[63], Bowtie[59], and GEM[83]. BWA-SW actually represents both the reference and read sequences with the FM-index, and matches these two FM-indexes directly.

It is important to note that, although many efficient aligners achieve rapid read alignment with the FM-index, it does not mean the FM-index bears a theoretical running-time advantage over a hash table. The efficiency of a hash table largely depends on the design of its hashing method. As a rule in software development, the performance of a good piece of software usually comes from its good implementation rather than from a good theory.

1.2.4 The essential element: approximate string matching algorithm

The alignment methods mentioned above are only a brief introduction of the mainstream strategies. Due to the way sequencing methods have developed (e.g. longer reads with more sequencing errors), the requirements from various biological problems (e.g. the reference genome may be incomplete or inaccurate, cross-species comparison may be required) and the complexity of genomic variation itself (e.g. the existence of long INDELs), the seed and extension alignment methods keep getting challenged. Not only the seeding but also the alignment-refining step is under pressure to improve.

Although in some cases the seed extension can be skipped with the help of long, gapped seeds, for many aligners this step is an unavoidable bottleneck for time and memory usage. Accurate seed extension requires dynamic programming, typically the SW algorithm. Various acceleration strategies of this dynamic programming algorithm were developed to reduce the time and space complexities. Among them, the most widely used solutions are the single-instruction multiple-data (SIMD) SW[27,98,116] and the bit-vector algorithm[91]. Such vectored alignment methods were adopted by many aligners, such as Novoalign, SHRiMP and RazerS3[113,112]. In this thesis I will present my SSW Library, which is an enhanced version of the SIMD SW, and it is one of the most accurate and efficient alignment-refining modules to date. Besides being used in the seed-extension step of aligners (SSW's application in Mosaik), SSW has already been widely applied in sequence assembly and variation detection tools. SSW's contribution to the rapid development of new sequence processing tools is far from trivial.

1.3 Genomic variation detection methods

The detection of genomic variation (including single nucleotide polymorphism, multiple nucleotide polymorphism, insertion and deletion) using NGS data is one of the most important tasks in the field of genomics. Detection of genomic variation is essential for understanding many biological phenomena such as evolution, the differences between species and individuals, and the mechanisms of many diseases. Genomic variation information also helps a lot with the clinical treatment of diseases. Variation detection methods differ depending on the type of variation that people work on (including short SNP, INDEL detection, and large copy number variation detection). Here we focus on the detection of short to medium length (1 - 50 bp) variation. Detection of this type of variation is mostly realized by two kinds of approaches: one is based on read alignment to the reference genome, the other is based on whole genome *de novo* assembly. Most currently popular variation detectors fall into the first category, and they apply the Bayesian model to each column of the alignment results (pileup data) to call genotypes of the sequenced genome.

1.3.1 Variation detection based on the Bayesian model and pileup data

The pileup data is generated from the alignment results of the read mapping tools. Each column of the alignment result is a unit of the pileup data, which includes the reference nucleotide and the read nucleotides. Note that the pileup data of one locus sometimes can include multiple columns of the alignment result, depending on the algorithm of the variation detection tool and the kind of variation (SNP, MNP or INDEL) the tool is looking for.

In theory, if the sequenced genome is different from the reference genome at a given locus, the reads at that position will contain different nucleotides than the reference. In reality, differences between the read and reference sequences can also be caused by sequencing errors or misalignment. These are the reasons for false positive variation calls. Controversially, sequencing errors and misalignment can also mask a true variation. To distinguish true variation from errors, the Bayesian method is used as a statistical tool to recover the underlying genotype from these disturbance factors.

Simply stated, the Bayesian method uses two inputs: 1) the known probability of different variations happening at a certain locus in the individual genome (the prior), and 2) the observed pileup data at this locus; from these, it generates one output: the posterior probability of the genotype of the sequenced genome at this locus. By this transformation, while observing data D, the probability of the underlying genotype T_i can be expressed as

$$P(T_i|D) = \frac{P(T_i)P(D|T_i)}{\sum_{x=1}^{S} P(T_x)P(D|T_x)}$$

where S is the count of all possible genotypes. Taking SNP calling as an example, for a diploid genome, $T_i = H_1H_2 \in AA, CC, GG, TT, AC, AG, AT, CG, CT, GT$, S = 10. Each prior genotype $P(T_i)$ can be estimated based on the reference genome and existing large variation discovery studies. For example, the SNP rate between two haploid human chromosomes was estimated to be 0.001[104], and the error rate of the human reference genome was estimated to be 1×10^{-5}.[44] Based on these, the homozygous SNP rate can be estimated to be 0.0005, and the heterozygous rate is 0.001.[69] If we consider that transitions happen four times more frequently than transversions[121], the probabilities of different genotypes given a reference nucleotide can be calculated (Table 1 of the SOAPsnp paper[69]). Assuming the base read depth is n at a certain locus, the likelihood $P(D|T_i)$ for each T_i can be calculated as

$$P(D|T_i) = \prod_{k=1}^{n} \frac{P(d_k|H_1) + P(d_k|H_2)}{2}$$

where H1 and H2 are the assumed genotype T_i on the two chromosomes; d_k is each allele in the reads at that locus. $P(d_k|H_1)$ and $P(d_k|H_2)$ can be directly gotten from the sequencing error rate. After the posteriors of all possible T_i are calculated, the T_i with the largest $P(T_i|D)$ will be chosen as the called genotype and the quality of this genotype can be set based on the corresponding $P(T_i|D)$.

Moreover, the allele type, base quality, mapping quality of the read, base coordinate of the read, the consecutive repeat times of a nucleotide, and whether the allele is in a duplicated read, etc. may all be considered as the observed data and can be incorporated in the genotype calling to achieve better software performance. Popular variation detection tools based on this method include the Genome Analysis Toolkit (GATK)[85], Freebayes[33] and Samtools[64].

1.3.2 INDEL realignment

Due to sequencing errors, genomic variation and the complexity of genomic sequences, alignment artifacts are unavoidable problems that influence the downstream variation detection a lot. Especially in repeat and low-entropy regions, the generation of a detailed alignment becomes more difficult, and usually various alignments can be considered to be equally optimal. For reads from the same genomic region, different alignments to the reference can be seen due to random sequencing errors and alternative best alignments. These uncertainties usually mislead the variation detector. Because of the time complexity of the alignment algorithms and the reference genome length, most current alignment methods can only do one pairwise alignment between the reference and one read at a time. In NGS sequencing, one genomic region is usually sequenced multiple times. The information from different reads (containing different sequencing errors) can complement each other to help us get a better alignment of them. Using the information from all the reads together is usually essential to achieving a reasonable alignment, and this realignment task using multiple reads together should be done by variation detection tools. Actually, all currently successful variation detection tools rely on local realignment to achieve their performance.

Using the correlation between reads of the same genomic region is important for getting the correct alignment, especially when INDELs exist in the corresponding region. One realignment algorithm, SRMA[41], constructs a variant graph based on the original alignment of each read to the reference sequence, then realigns each read back to the variation graph. When building the variation graph, if the read is originally exactly matched to the reference, the graph is a single line; if another read has any variation, a path is added according to the alignment. One example of variant graph construction is shown in Figure 1.4. When the variant graph is being built based on all the reads, a dynamic programming procedure similar to SW is applied to the variant graph between each read. During this realignment process the alternative alignment path on the graph (derived by other reads) can be seen and considered for each read alignment. Therefore, the realigned read may align differently and better compared to its original alignment. In fact, GATK and Freebayes both use simplified versions of SRMA to realize their local read realignment.

However, SRMA and similar methods have a serious drawback: an INDEL can only be correctly realigned if at least one of the original reads was aligned correctly with that INDEL. Because the variant graph is generated from the original read alignment, if none of the reads are aligned with the INDEL, the following realignment procedure cannot see the INDEL path from the graph and will most likely not generate any alignment with an INDEL. A better solution of the realignment problem was actually developed in 1997. This algorithm is called "ReAligner" by Anson and Myers[6], and it is able to generate a correct INDEL realignment independently of the initial alignment. To generate the realignment of sequences $S_1, S_2, \ldots, S_k$, the

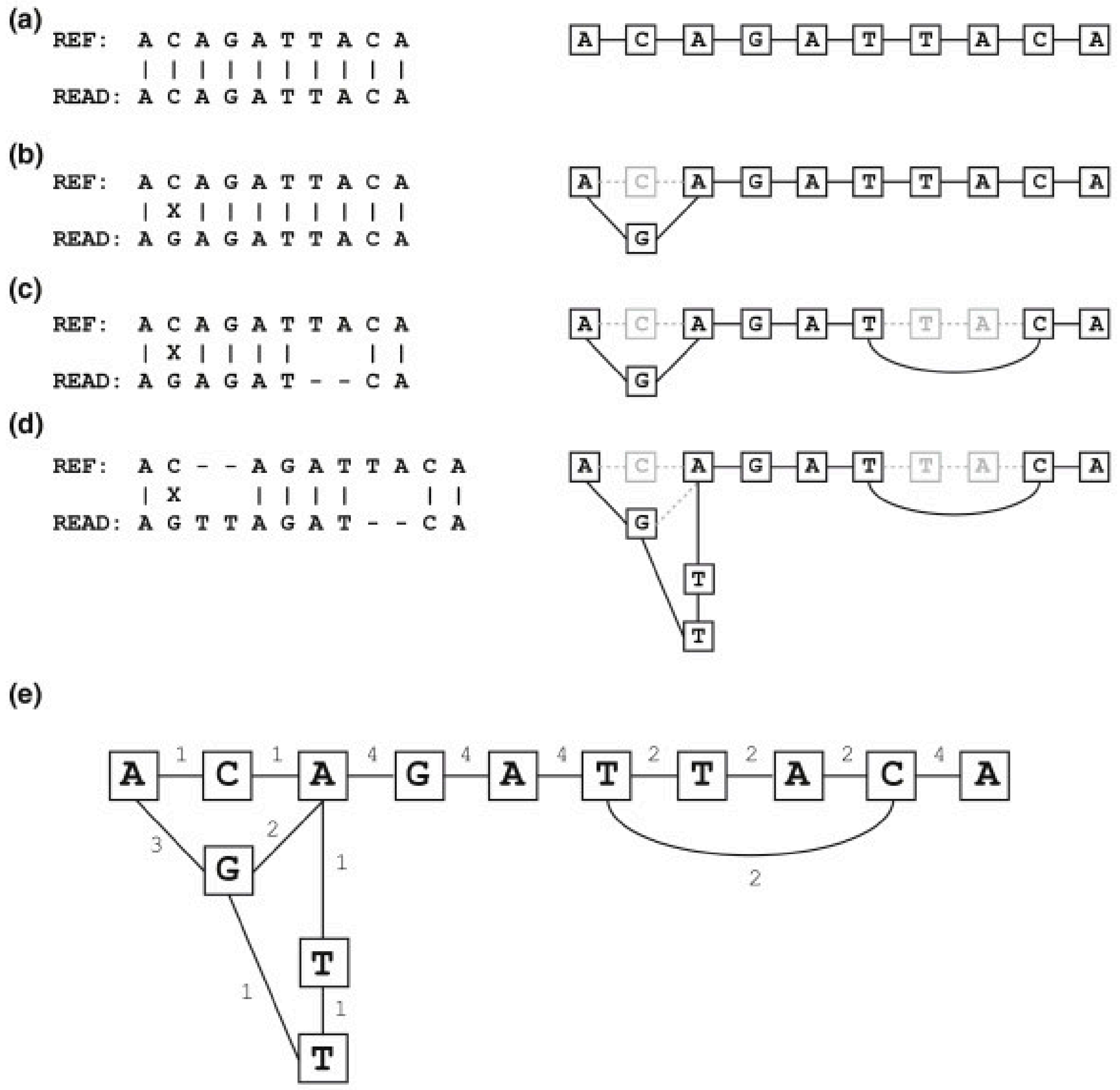


Figure 1.4: Variant graph construction. (Figure is from Homer and Nelson's paper[41]) (a) - (d) Four read alignments (left) are used consecutively to construct a variant graph (right). (e) shows the resulting variant graph. The numbers labeled on the edges are the counts of the alignments that contain this path.

core algorithm of this method is described using this pseudo code as follows:

```
repeat
        for (i = 1, i <= k, ++i) {
                Remove the ith read from the current multi-sequence alignment of $S_1 - S_k$.
                Align the ith read to the profile composed of the multi-sequence alignment of
                the remaining reads by banded dynamic programming.
        }
        Calculate the overall penalty of the current multi-sequence alignment of $S_1 - S_k$ column
        by column.
until The overall penalty of the multi-sequence alignment remains the same as the last iteration.
note: The banded dynamic programming here is a banded SW without the distinction between gap-open and gap-extension, which is a very fast algorithm.
```

Besides its ability of discovering INDELs that are not shown in the original alignment, ReAligner has another advantage: it realigns all the reads without any information from the reference sequence. This is a more correct approach compared to SRMA. Reads are from an individual genome that is independent of the reference. By realigning the reads only to each other, the bias from the reference genome can be avoided during variation detection. However, it is a pity that no variation detector is currently using ReAligner.

In this thesis we will present another method, the profile hidden Markov model (PHMM), which not only has all the advantages of ReAligner but also increases ReAligner's power by combining the probabilities from all alternative alignments of the reads.

1.3.3 Variation detection based on whole genome *de novo* assembly

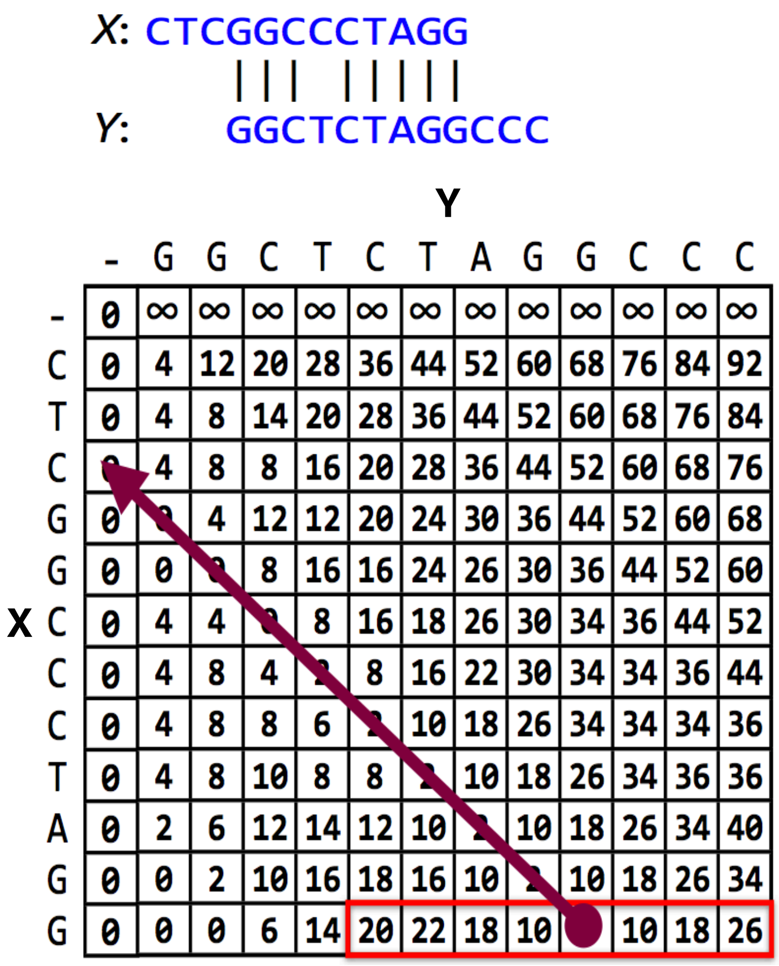
Another approach to variation detection, which is fundamentally different from the methods based on genome alignment, is variation detection based on whole genome *de novo* assembly.

Compared to the variation detection based on read mapping, assembly-based methods totally avoid the misalignment problem and the bias from the reference genome. With this approach, all reads from an individual genome are *de novo* assembled into long contigs, and the variations between the sequenced genome and the reference are then directly generated by comparing the assembled contigs to the reference. The whole genome assembly algorithms fall into into one of two categories: assembly using an overlapped graph and assembly using a De Bruijin graph (DBG). Both methods were first proposed at the computer science conference DIMACS in 1994 for solving the string-assembly problem: Gene Myers proposed the overlap graph, while Michael Waterman (one of the authors of the SW algorithm) proposed the DBG. Both algorithms were published in 1995[43,90].

1.3.3.1 The overlapped graph assembly

The overlapped graph assembly was later on refined to the overlap-layout-consensus (OLC) assembly. OLC assembly completes the assembly in roughly three steps: 1) Build an overlap graph. 2) Bundle simple stretches of the graph into contigs. 3) Build a consensus sequence of each contig from the reads.

The overlaps between every pair of reads need to be found for building an overlap graph. To do this, the aforementioned suffix tree is a good data structure. For this purpose, a suffix tree needs to be built with suffixes of all the read strings. After the tree is built, for each read Ri, we search the prefix of its string from the root of the suffix tree. The matched branch (other than R_i itself) represents the suffix of another read. All exact overlaps between any two reads can be found this way. Assuming we have n reads of length l, the total length N = nl, and a is the number of read pairs that overlap. The time complexity of building the suffix tree is O(N), and to report all the overlaps is O(a). The overall time for finding all the overlaps is O(N + a), meaning in the worst condition a = n2. In reality, it is better to allow mismatches and gaps when finding the overlaps between reads. This means that dynamic programming for string matching needs to be applied after filtering out the unoverlapped read pairs using the suffix tree. This dynamic



	-	G	G	C	T	C	T	A	G	G	C	C	C
-	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
C	0	4	12	20	28	36	44	52	60	68	76	84	92
T	0	4	8	14	20	28	36	44	52	60	68	76	84
C	0	4	8	8	16	20	28	36	44	52	60	68	76
G	0	0	4	12	12	20	24	30	36	44	52	60	68
G	0	0	0	8	16	16	24	26	30	36	44	52	60
C	0	4	4	0	8	16	18	26	30	34	36	44	52
C	0	4	8	4	2	8	16	22	30	34	34	36	44
C	0	4	8	8	6	2	10	18	26	34	34	34	36
T	0	4	8	10	8	8	2	10	18	26	34	36	36
A	0	2	6	12	14	12	10	2	10	18	26	34	40
G	0	0	2	10	16	18	16	10	2	10	18	26	34
G	0	0	0	6	14	20	22	18	10	[illegible]	10	18	26

Figure 1.5: An example of the prefix-to-suffix alignment of two strings. (This figure is revised from Langmead's slides.[Langmead]) This figure shows a simplified scoring matrix of this dynamic programming method. For easy description and understanding, gap-open and gap-extension penalties are not distinguished in this matrix. Real computational implementations do distinguish the gap-open and gap-extension. When locating the largest score in the last line of this matrix, the leftmost cells are skipped. This means we only consider the overlap with meaningful length. In this figure, the red arrow marks the backtrace path.

programming is trimmed specially for aligning the suffix of a string to the prefix of another string: a global alignment but initializing the beginning column with 0, the beginning row with ∞, and backtracing from the last line of the suffix-aligned string to the first line of the prefix-aligned string in the matrix. An example of this dynamic programming method is shown in Figure 1.5. Of course, there are alternative methods to do the string overlapping such as Celera Assembler's overlapper (http://wgs-assembler.sourceforge.net/wiki/index.php?title=Main_Page) and Simpson and Durbin's assembler based on FM-index[106], but the basic ideas are similar.

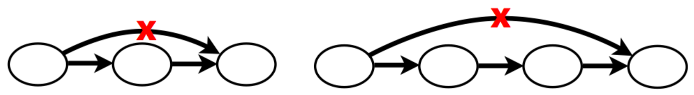


Figure 1.6: Two examples of redundant edges. The edges marked by the red crosses are the redundant edges. The information represented by these edges (a long overlap between the beginning and end edges) is already contained in the connections of the central nodes (short overlaps between reads).

The overlap graph is built based on the read overlap: the reads are the notes, if a suffix of a read R1 is overlapped with the prefix of another read R2 with the overlap longer than a threshold, an edge is added from R1 to R2, and so on. The edges in the overlap graph can be weighted when there are several similar overlaps from similar read pairs. After the initial graph is built, edges carrying redundant information (such as the ones shown in Figure 1.6) or low-weight branches ending abruptly (may be caused by sequencing errors) are removed to reduce the complexity of the graph. The non-branching stretches in the remaining graph are the contigs. Some other parts of the graph may still contain branches or loops caused by repeat sequences. To date, such sub-graphs still cannot be resolved. This problem is especially tremendous for graphs constructed from short reads. The last step of OLC is constructing the consensus sequence of the contigs: lining up the overlapped reads, and for each locus, taking the major nucleotide as the consensus genotype. In this step, sequencing errors and ploidy can cause uncertainties.

Other assemblers based on OLC include SGA and PHAST (http://gcat.davidson.edu/phast/). From the above description, we can see that building an overlap graph is time consuming.

1.3.3.2 *De Bruijin* graph assembly

DBG assembly also contains the three steps: 1) graph construction, 2) contig construction from non-branching paths, and 3) consensus sequence generation for the contigs. In this approach, step 2) and 3) are very similar to DBG assembly, so in this section I focus on the description of step 1), the DBG construction. During DBG construction, all the reads are broken into k-mers. Take k = 3 as example, if a DNA sequence is ACGTAG, the k-mers are: ACG, CGT, GTA, TAG. Then a DBG is built based on the exact (k - 1)-mer overlap between each pair of the k-mers. In a DBG, the unique k-mers are vertices; if the (k - 1)-mer suffix of a k-mer is the same as the (k - 1)-mer prefix of another k-mer, one edge is added from the suffix-overlapped node to the prefix-overlapped node, and so on. DBG construction can be realized by a suffix tree or a hash table or other functionally similar data structures. As the overlap length between each pair of k-mers is fixed, a DBG naturally does not contain redundant edges as shown in Figure 1.6. DBG's construction process is also easier compared to OLC; it only does exact string matching. Due to this advantage, in the past few years, many assemblers utilized the DBG method, including Cortex[45], SOAPdenovo[72], and Velvet[119].

On the other hand, constructing the graph with short k-mers wastes the information carried by long reads. As a result, DBG can only resolve the repeats that are contained in the k-mers, and since k is usually much smaller than the read length, DBG is less capable than OLC for sequence assembly. Because the count of k-mers is large, the actual computational time consumed by constructing and interpreting the DBG is similar to that of OLC. As the sequencing read length keeps increasing, OLC's advantage has become more striking recently.

1.3.4 Problems and new challenges

In summary, the detection of variation is a crucial task at the foundation of a lot of biological research and clinical treatments. However, there is still a long way for us to explore for achieving accurate variation detection, especially for INDELs, MNPs, and other structural variations, for variation in low-entropy genomic regions, and for complicated variation such as in cancer genomes. The traditional variation detection methods based on pileup data and the Bayesian model cannot avoid the difficulties caused by alignment uncertainties and the bias from the reference genome. Current assembly methods have difficulty with assembling repeat regions and with resolving complexities caused by multi-ploidy. The large time and space complexities are also a big problem for practical variation detection based on assembly. Though the increasing read length reduces the assembly difficulties caused by repeat regions, the high error rate of the new sequencing techniques brings other issues with it.

In this thesis I will introduce a novel approach for variation detection based on PHMM, which will 1) largely avoid the alignment uncertainty problem and the bias from the reference, and at the same time 2) handle sequencing error in a comprehensive way. This first point is realized by a Baum-Welch training of a HMM profile with all the reads. The second point is realized by HMM's strength at signal recognition.

In all our deeds, the proper value and respect for time determines success or failure.

Malcolm X

2 SSW Library: An SIMD Smith-Waterman C/C++ Library

The SSW Library is used to speed up any software that needs a Smith-Waterman algorithm.

2.1 Introduction

The Smith-Waterman-Gotoh algorithm (SW)[107,34] is the most influential algorithm for aligning a pair of sequences. It is an essential component of the majority of aligners from the classical

BLAST[5] to the more recent mappers. Although most of these aligners do not use SW directly to align a sequence to the whole genome sequence due to the quadratic time complexity of SW, they extensively use it for seed extension and for constructing the final alignment, and spend significant amount of CPU time on this algorithm. Due to the critical role of SW, many efforts have been made to accelerate SW, taking the advantages of special hardware such as single instruction multiple data (SIMD), field-programmable gate array (FPGA) and graphics processing unit (GPU)[75,76,52]. Among the three, SIMD based algorithms are most frequently used because they are compatible with most modern x86 CPUs. SIMD acceleration methods can be further divided into intra-sequence parallelization[116] and inter-sequence parallelization[98]. Inter-sequence parallelization is only useful when many pairs of sequencing reads are aligned simultaneously; intra-sequence parallelization however parallelizes for each single pairwise alignment, so it can be used more flexibly in various applications such as that needs aligning a single read against a potentially large genome reference sequence. Farrar's Striped SW[27] with SWPS3's[108] improvement is the fastest intra-sequence parallelized SIMD implementation running on x86 processors with the Streaming SIMD Extensions 2 (SSE2) instruction set. Indeed, Farrar's algorithm has been embedded in several popular genomic sequence mapping tools, such as BWA-SW[63], Bowtie2[58], Novoalign (http://www.novocraft.com/) and Stampy[81].

Though striped SW is tens of times faster than a standard SW implementation, only a few aligners have used this more advanced algorithm. There are several practical obstacles. Firstly, implementing a striped SW requires good understanding of SSE2 instructions and the more complex algorithm, which may take significant development time. Secondly, the original striped SW only gives the optimal alignment score but does not report the position or the detailed alignment, the information necessary for using SW as a component to construct the final alignment. How to report the position and alignment without affecting speed is non-trivial. Thirdly, while a few implementations report position and alignment, they are tightly integrated in a larger project and cannot be easily reused in other programs. Fourthly, when aligning a short read against a long sequence, we would like to know suboptimal alignments such that we can tell if the optimal position is trustworthy. Most existing libraries have not addressed this issue.

Although striped SW has been published for six years, we are still in lack of a fast, versatile and standalone library. This leads us to develop the SSW library, a light weighted but comprehensive C/C++ library for pairwise sequence alignment with the striped SW algorithm.

2.2 Results and Discussion

2.2.1 Implementation

To build a light-weight and easily reusable SIMD SW library for the genomic application development community, we made the SSW Library. It extends the Striped SW and SWPS3's SIMD implementations to provide the mapping location and detailed alignment information (traceback), without performance penalty. Though these features are crucial when integrating SW into other genomic analysis systems, among the existing SIMD SW implementations only SSEARCH provides them, and as discussed in the Performance: Short-Read Genomic Alignment section its performance in typical genomic alignment contexts is poor. The SSW library can also return the heuristic suboptimal (second-best) alignment score and location without additional computational cost, which enables the use of the method in contexts that exploit this information in mapping-quality estimation. We describe our efficient implementation of these features in the Methods section.

2.2.2 Usage

The SSW library is an application program interface (API) that can be used as a component of C/C++ software to perform optimal protein or genome sequence alignment. The library returns the SW score, alignment location and traceback of the optimal alignment, and the alignment score and location of the suboptimal alignment. We provide the library with an executable alignment tool that can be used directly to perform protein or DNA alignments. It is a demonstration of the API usage and a practical tool for accurate whole viral or bacterial genome alignment. Moreover, since this tool is sufficiently fast and memory-efficient for align-

Table 2.1: Command lines used for Figure 2.1.

SSW	./ssw_test -p -o12 -e2 database.fasta query.fasta
SSW-C	./ssw_test -pc -o12 -e2 database.fasta query.fasta
Farrar	./farrar blosum50.txt query.fasta database.fasta
SSEARCH	./ssearch36 -p -3 -d1 -T1 query.fasta database.fasta
CUDASW++ 2.0	./cudasw -db database.fasta -query query.fa -mat blosum50

ment to very large reference genome sequences, e.g. the human genome, it can also be used to validate alignments produced by heuristic read mappers. The instructions of how to install and run the library is described in the README file at the software website (https://github.com/mengyao/Complete-Striped-Smith-Waterman-Library). A test data set is also provided there.

2.2.3 Performance.

2.2.3.1 Protein Database Search.

We compared SSW's performance (with and without returning the detailed alignment, SSW-C and SSW, respectively) to Farrar's accelerated SW and SSEARCH (version 36.3.5c) on a Linux machine with 2GHz x86 64 AMD processors. We ran each program on a single thread. Since the optimal alignment scores for long DNA sequences given by SWPS3 are not consistent with others', we did not benchmark its running time here.

To measure the speed of protein database searching, we aligned five protein sequences (Q6GZW9 (75 aa), P14942 (192 aa), P42357 (551 aa), P07756 (1283 aa), and P19096 (2154 aa)) against the Uni-Prot Knowledgebase release 2013_09 (including Swiss-Prot and TrEMBL, a total of 13,823,121,038 aa residues in 43,362,837 sequences), by all four algorithms (see Figure 2.1). Since SSEARCH did not return alignment results against the entire Uni-Prot database, we were only able to test it against one quarter of the TrEMBL sequences (3,872,274,471 aa in 10,705,468 sequences). The command lines used for the database search are given in Table 2.1. Our SSW algorithm is the fastest or equally fast to SSEARCH across the entire protein sequence length range we tested.

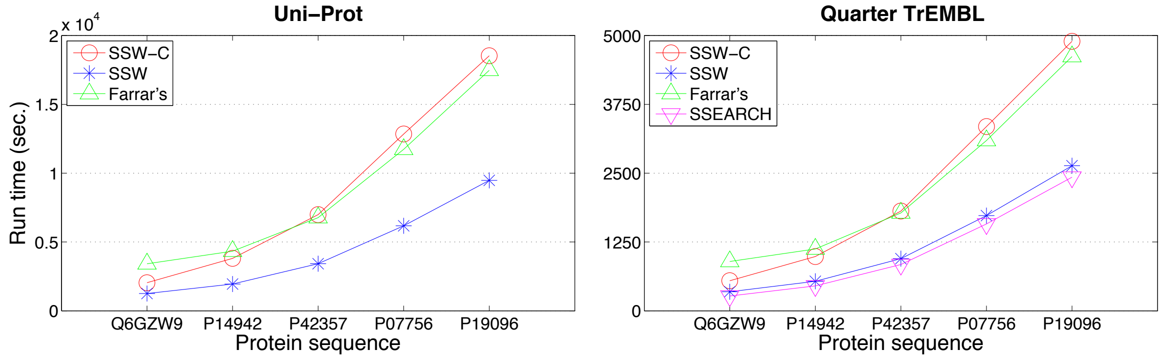


Figure 2.1: Running time of different SW implementations for protein database search. 5 query proteins were searched against the whole Uni-Prot database (left) and one quarter of the TrEMBL database (right). Running time is shown on the y-axis for SSW without (blue) and with (red) detailed alignment, Farrar's implementation (green) and SSEARCH (pink). All SW implementations used the BLOSUM50 scoring matrix with gap open penalty -12 and extension penalty -2.

We also compared the CPU SW implementations with one of the most popular GPU implementations, CUDASW++[75,76]. We were unable to obtain access to a system with a GPU card supporting CUDASW++ 3.0, so we ran CUDASW++ 2.0 (on a GeForce GTX 480 graphics card with 1.5G memory) for the comparison. Since our GPU host machine does not have sufficient storage for a larger protein database, such as TrEMBL, we aligned the five query proteins against the Swiss-Prot database (release 2013_09). The total running times in seconds of SSW, SSW-C, Farrar's, SSEARCH, and CUDASW++ 2.0 are 310.10, 597.35, 424.27, 270.94 and 66.75 respectively. This GPU SW is about four fold faster than the fasted CPU SW. However, we did see the difficulty of using GPU SW, e.g. hardware is not easy found and installation is not ordinary.

2.2.3.2 Short-Read Genomic Alignment

To benchmark genome sequence alignment, we tested the programs with both simulated data and real sequencing reads. We selected 1Kb - 10Mb regions from human genome chromosome 8, and using an Illumina read simulator (http://www.seqan.de/projects/mason/) we generated a thousand 100 bp-long sequences from these regions. We then aligned these reads back to their corresponding reference sequences with each of the five algorithms (including our naïve Smith-Waterman, https://github.com/wanpinglee/SmithWateman) and compared their running

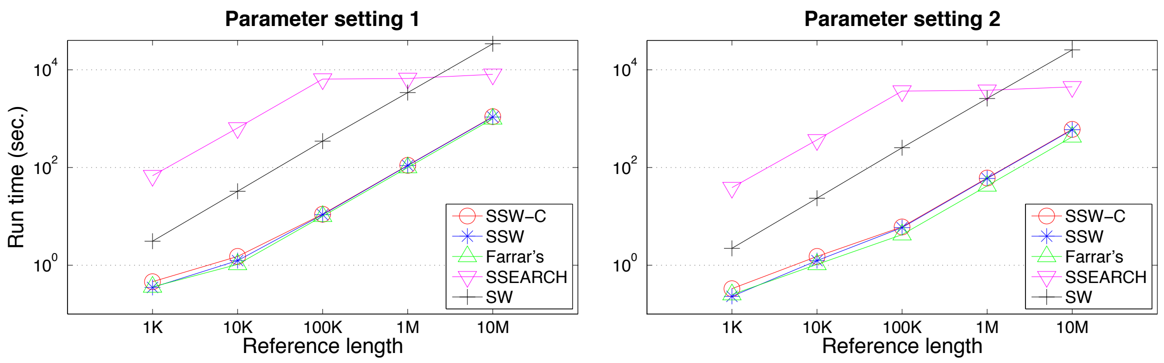


Figure 2.2: Running time of different SW implementations for simulated genomic read alignment. Running time of aligning 1,000 simulated Illumina reads to human reference sequences of various lengths. The log-scaled running time is shown on the y-axis for SSW without (blue) and with (red) detailed alignment, Farrar's implementation (green), SSEARCH (pink) and an ordinary SW implementation (black). All SW implementations were tested under two sets of SW parameters: scores of match, mismatch, gap open and extension are 2, -1, -2, and -1 respectively (left), and scores of match, mismatch, gap open and extension are 1, -3, -5, and -2 respectively (right).

times (see Figure 2.2). Each program was run on a single thread on a Linux machine with 2G MHz x86_64 AMD processors. Two SW parameter settings are employed in the experiments: setting 1 (scores of match, mismatch, gap open and extension are 2, -1, -2, and -1 respectively) and setting 2 (scores of match, mismatch, gap open and extension are 1, -3, -5, and -2 respectively). The command lines used for read alignments are shown in Table 2.2. Our SSW algorithm and Farrar's accelerated version are equally fast for the whole reference length spectrum.

For the comparisons on real sequencing datasets, we aligned four sets of a thousand reads representing three different sequencing technologies against four different reference genomes: (1) Applied Biosystems (ABI) capillary reads (1,388 bp average length) against the severe acute respiratory syndrome (SARS) virus (29,751 bp); (2) Ion Torrent reads (236 bp) against E. coli (4.94×10^3 bp); (3) Illumina reads (100 bp) against T. gondii (6.08×10^7 bp); and (4) Illumina reads against human genome chromosome 1 (2.49×10^8 bp) as shown in Figure 2.3. The same SW parameter settings as the tests on the simulated data sets are used in the experiment. The detailed genome and read information is described in Appendix A. The command lines used for read alignments are shown in Table 2.2. These results indicate that even while returning a full optimal alignment and one suboptimal score, our SSW algorithm is just as fast as Farrar's accelerated version.

Table 2.2: Command lines used for Figure 2.2 and 2.3.

	Parameter setting 1
SSW	./ssw_test -m2 -x1 -o2 -e1 reference.fa reads.fa
SSW-C	./ssw_test -m2 -x1 -o2 -e1 -c reference.fa reads.fa
Farrar	./farrar -i -2 -e -1 matrix.txt reads.fa reference.fa
SSEARCH	./ssearch36 -T1 -d1 -g -1 -f -2 -r +2/-1 -3 -n reads.fa reference.fa
SW	./sw -m2 -x1 -o2 -e1 reference.fa reads.fa
	Parameter setting 2
SSW	./ssw_test -m1 -x3 -o5 -e2 reference.fa reads.fa
SSW-C	./ssw_test -m1 -x3 -o5 -e2 -c reference.fa reads.fa
Farrar	./farrar -i -5 -e -2 matrix.txt reads.fa reference.fa
SSEARCH	./ssearch36 -T1 -d1 -g -2 -f -5 -r +1/-3 -3 -n reads.fa reference.fa
SW	./sw -m1 -x3 -o5 -e2 reference.fa reads.fa

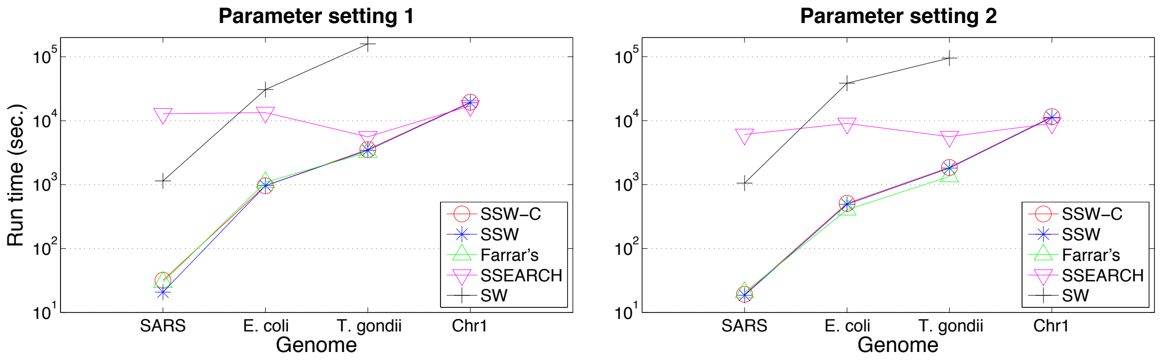


Figure 2.3: Running time of different SW implementations for real genomic read alignment. Running time of aligning 1,000 real sequencing reads to various microorganism genomes and the human chromosome 1 are shown. Farrar's implementation cannot handle long sequences as human chromosome 1, so its corresponding running time is not shown here. The log-scaled running time is shown on the y-axis for SSW without (blue) and with (red) detailed alignment, Farrar's implementation (green), SSEARCH (pink) and an ordinary SW implementation (black). All SW implementations were tested under two sets of SW parameters: scores of match, mismatch, gap open and extension are 2, -1, -2, and -1 respectively (left), and scores of match, mismatch, gap open and extension are 1, -3, -5, and -2 respectively (right).

Table 2.3: Comparison of the running time (seconds) between the banded SW engined MOSAIK and the SSW engined MOSAIK.
We aligned three million Illumina 100 bp reads and one million 454 reads against the human genome.

	Illumina 100 bp	454
Banded SW	70145.760	240535.730
SSW	38927.380	98198.990

We note that the relative performance of SSEARCH against our method is the worst when working with short target DNA sequences, which is exactly the context in which pairwise alignment is most likely to be used.

2.2.4 Applications

Here we demonstrate the utility of our SSW library as a component of four different biologically meaningful applications.

2.2.4.1 Primary short-read mapper.

To provide highly accurate alignments, most short-read mappers integrate an SW algorithm for a final "polishing" step. This step is especially important for aligning reads containing short insertions and deletions. Even though each SW run is short, it may be applied hundreds of millions of times within a single run of a mapper, and therefore even small inefficiencies result in wasteful resource usage. To quantify time savings with SSW, we compared the performance of our new method with the existing SW implementation within the MOSAIK mapping program[60], which uses SW for the final read alignments. We found that the SSW library achieves a two-fold speedup of the entire MOSAIK compared with the current banded SW implementation within it (see Table 2.3). Notably, MOSAIK is a multi-threaded program and thus the SSW component in MOSAIK is running in parallel.

2.2.4.2 Secondary short-read mapper.

Primary read mappers are often unable to map or properly align reads in structural variant (SV) regions, e.g. in regions of deletions, insertions, inversions, or translocations. Therefore, we developed a split-read aligner program, SCISSORS (https://github.com/wanpinglee/scissors) to map reads across structural variation event boundaries (breakpoints), rescuing reads not mapped, or inaccurately mapped by primary mapping approaches. We used our SSW library to align "orphaned" or severely "clipped" fragment-end read mates (in the case of read pairs where one end-mate is aligned with high mapping quality, but the other mate is either unmapped or mapped with many unaligned or "clipped-off" bases) to the genomic regions indicated by the well-mapped mates' coordinates. Inclusion of the SW mapping routines of our SSW library makes accurate and fast split-read alignment for SV detection possible. The split-read mapping functionality, using our SSW library, has also been implemented in the TANGRAM SV detection tool (https://github.com/jiantao/Tangram). TANGRAM is used intensively in the 1000 Genomes Project[1,2] to accurately detect MEIs (mobile element insertions) (http://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/working/20120815meicalls/BC/). In SCISSORS and TANGRAM, SSW is also called in multi-threaded way.

2.2.4.3 Read-overlap graph generation.

To evaluate evidence for putative SV and large INDEL calls generated by assembly methods, we can employ a read-overlap graph generated by exhaustive pairwise alignment of a set of reads which co-localize in a specific genomic region. We tested the effect of the SSW library in this application against a standard SW implementation (https://github/ekg/smithwaterman). To do the speed comparison test, we generated read-overlap graphs for the genomic region 20:21026000-21027500 using 22,543 reads (70 bp long) from 191 African samples in the 1000 Genomes Project dataset. The sample identifiers are listed in Appendix B. The running time of the read-overlap graph generation program RZMBLR (https://github.com/ekg/rzmblr) embedded with an ordinary SW (1795.66 seconds) is about twice of that embedded with our SSW library (973.02 seconds).

2.3 Conclusions

We developed and made available a fast SW library using SIMD acceleration. By returning not only the optimal alignment score but also the actual alignment, as well as a secondary optimal or suboptimal alignment score, the SSW library is suitable for inclusion into other heuristic genomic sequence analysis programs requiring local SW alignment. The most significant utility of our development, however, is that our algorithms can be readily integrated into C/C++ software without modification of the source code, accelerating development for larger software tools. SSW has already been adopted in four programs developed by our group: the primary read mapping tool MOSAIK, the split-read mapping program SCISSORS, the MEI detector TANGRAM, and the read-overlap graph generation program RZMBLR.

2.4 Methods

Our algorithmic improvements focused on speeding up the Farrar's implementation and gaining access to the optimal alignment (in addition to the optimal achievable score), as well as the score of the best secondary alignment. For speedup, we adopted the "lazy F loop" improvement proposed by SWPS3[116]. Furthermore we obtained additional alignment information compared to SWPS3 without slowing down the original algorithm: (1) we record the optimal alignment ending positions during the SIMD SW calculation and generate the detailed alignment by a reversed SIMD SW and a banded SW. When the score matrix is filled by the SIMD SW calculation, we store the maximal score of each column in a "max" array and record the complete column that has the maximal score of the whole matrix. Next, we locate the optimal alignment ending position on the reference and the query by seeking the maximal score in the array and the recorded column respectively. The reversed SIMD SW locates the best alignment beginning position from the ending position by calculating a much smaller scoring matrix. Then, the banded SW (whose band is defined by the beginning and ending positions) generates the detailed alignment. Since the alignment generation using the reversed SIMD SW and the banded SW only calculates a very small portion of the whole SW scoring matrix when the query sequence is much shorter than the target, in most cases of genome sequence alignment the corresponding time cost is trivial. (2) We determine the secondary alignment score by seeking the second largest score in the "max" array. To avoid a similar sub-alignment of the primary alignment returned, we mask the elements in the region of the primary alignment of the "max" array and locate the second largest score from the unmasked elements (Figure 2.4). As a crucial step for locating the alignment position and estimating the suboptimal score, the "max" array generation is completed by adding an SSE2 command in the inner loop of Farrar's implementation and another command in the outer loop, so that this additional time consumption is limited.

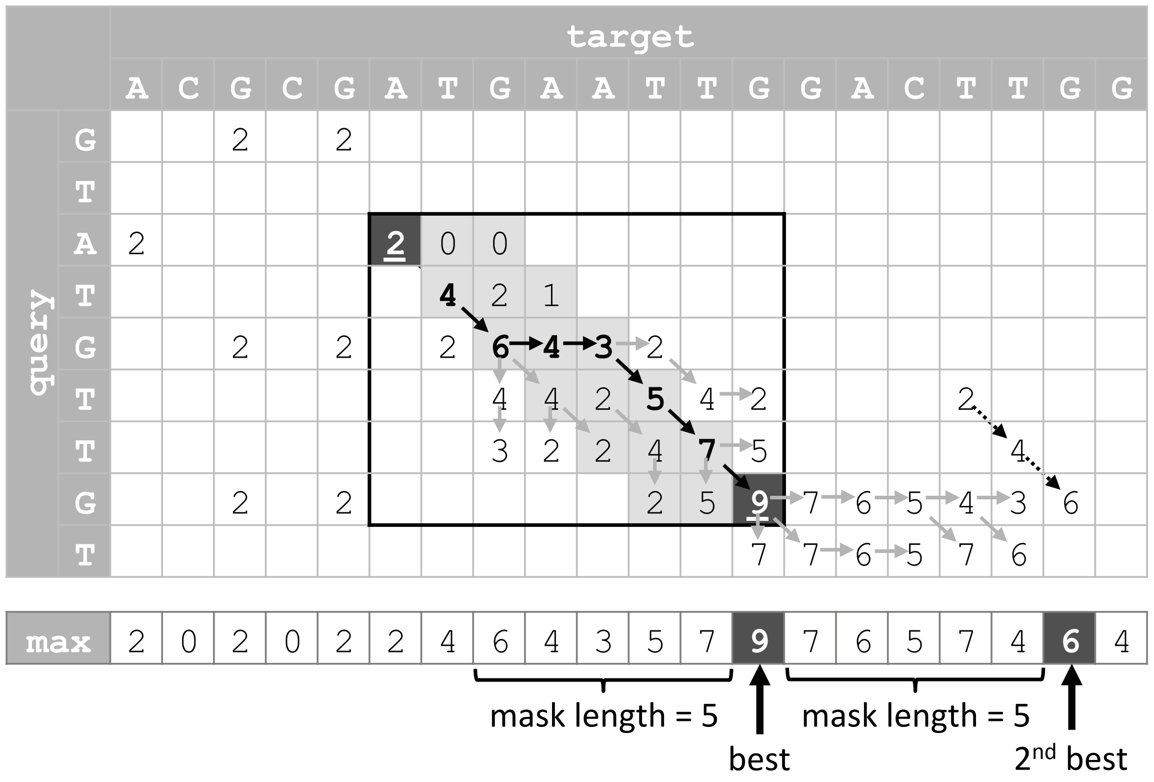


Figure 2.4: Illustration of alignment traceback and suboptimal alignment score determination. An example SW score matrix is shown (penalties for match, mismatch, gap open and extension are 2, -1, -2, and -1 respectively). The bottom row indicates the maximum score for each column. The algorithm locates the optimal alignment ending position (the black cell with score 9) using the array of maximum scores, and then traces back to the alignment start position (the black cell with score 2) by searching a much smaller, locally computed score matrix (circled by the black rectangle). Finally, a banded SW calculates the detailed alignment by searching the shaded sub-region. The scores connected by solid arrows belong to the optimal alignment. The max array records the largest score of each column. After the optimal alignment score (marked by "best") is found, its neighborhood is masked, and the second largest score is reported outside the masked region (marked by 2nd best). The scores connected by dashed-line arrows trace the suboptimal alignment.

We are the recipients of scientific method. We can each be a creative and active part of it if we so desire.

Kary Mullis

3

PHV: An accurate variant detector based on the Profile Hidden Markov Model

3.1 Introduction

The discovery of single nucleotide polymorphisms (SNPs), insertions and deletions (INDELs), and other kinds of genomic variation is very important to disease studies, association mapping and population genetics. However, considering that true variants have to be distinguished from false positives in large volumes of error-prone sequencing reads, variation detection on a whole genome population level remains a difficult task. One of the leading sources of incorrect SNP calling is errors caused by INDEL identification[65], which itself is a considerable problem. INDEL calling difficulties are caused by several reasons including: 1) INDELs occur about eight times less frequently than SNPs[14,80]; 2) reads bearing INDELs are generally more difficult to be correctly mapped[66]; 3) current mapping softwares treat reads separately and tend to align reads with mismatches rather than a gap; and 4) INDELs often cannot be uniquely mapped because of specific genome sequence patterns, for example repeats[4]. Consequently, the performance of variation detecting software is largely influenced by the INDEL sequencing error rate of the input data. New sequencing techniques such as PacBio[114], Ion Torrent (http://www.iontorrent.com) and Oxford Nanopore's nanopore sequencing[102] can generate longer reads and require much less DNA sample. However, the reads generated by the new sequencing techniques have a higher INDEL sequencing error rate, and will bring more challenges to the task of variation detection.

Great efforts have been put into the development of algorithms for calling SNPs and INDELs from next generation sequencing (NGS) data. Over the past few years, these efforts have led to significant improvements in the accuracy of variant calling, but despite the existence of numerous variation detection tools SNP calling and especially INDEL calling are still unsolved problems. In the 1000 Genomes Project, for example, the overlap calls between different groups are only about 20 precent when variations longer than one base pair are being called in low-entropy regions, even with a read coverage of 50. Some difficulties of variation detection stem from the basic methodology. The mainstream variant detection tools detect variations based on pileup data generated by alignment software, such as the Genome Analysis Toolkit (GATK)[85] and Samtools[66]. Mapping software treats reads separately, thus an inconsistent alignment of reads can dramatically weaken the true variation signal. Since variation detectors based on pileup data are unable to make full use of the correlated information between the sequencing reads, sequencing errors and the alignment uncertainty can cause big problems for variation calling. In order to resolve these issues, people have developed various work-around solutions, such as alignment filtering, read realignment (e.g. used in GATK and Samtools), and aligning reads to multiple locally assembled haplotypes (e.g. used in Dindel[4] and inGAP[95]).

However, these adjustment or local solutions still cannot solve the problem from the root. As the reads generated by NGS technologies have become longer, methods for whole genome assembly attracted people's attention again. Some tools were developed to assemble NGS reads and detect variation from the differences between the assembled genome and the reference, such as fermi[62], SOAPindel[73] and Cortex[45]. Although the assembly-based methods can avoid the alignment problem of the mapping-based variation detectors, they have alternative problems and limitations that are also difficult to resolve. Assembly-based methods have intrinsic difficulties to construct repeat regions or detect heterozygote variants. Due to the large time and space complexities of the assembly algorithms, running time and memory usage of many tools are still impractical for whole mammalian genome assembly. Moreover, whole genome *de novo* assembly usually requires deep sequencing reads, and for SNP detection, assembly-based methods still have not caught up with mapping-based methods.

With the emergence of the single-molecule sequencing technique and the processing of the cancer genome project, new challenges are being brought to the task of SNP and INDEL calling. In order to deal with data containing more INDEL sequencing errors and increased alignment uncertainty problems in low-entropy regions caused by complicated mutations, the historic and powerful signal recognition method hidden Markov model (HMM) was brought into the field of variation detection to specifically tackle the challenges from noisy and complex data. For example, VARiD uses an HMM containing 16 nitrogenous base double-mer states to gain better SNP and micro-INDEL detection for color-space reads[20]; DinDel uses a Bayesian network and the Viterbi algorithm to do local haplotype assembly for INDEL detection[4]; SNVHMM models each reference position with three genotype states to call SNPs from cancer data[8]; Samtools uses a profile HMM (PHMM) to calculate the Base Alignment Quality (BAQ)[61]; and PyroHMMvar and PyroHMMsnp use bi-factor hidden states to model homopolymer errors and to achieve a better accuracy of SNP and short INDEL detection for Ion Torrent and 454 data[117,118]. Summarizing the existing HMM for variation detection, the design of the hidden states is the key factor that decides what kinds of signal the HMM can capture.

For example, VARiD uses pairs of consecutive nucleotides as hidden states and its strength is modeling the color-space sequences; SNVHMM uses three genotypes at each reference position as hidden states for SNP genotyping, but it cannot do INDEL detection; and PyroHMMvar uses hidden states containing two factors (the homopolymer nucleotide and the consecutive count of this nucleotide) to emphasize the modeling of homopolymer, but this design complicates its SNP detection process and slows down the program. Compared to the above HMM designs, the basic global (for reads) to local (for reference) PHMM as used in BAQ directly represents all the information of the sequence alignment problem. Though this PHMM design is flexible for evaluating all kinds of genomic variation, BAQ's only function is calculating the alignment quality.

In order to provide an accurate and comprehensive tool for variation detection, we developed PHV (the PHMM variation detector). PHV uses a HMM profile similar to BAQ[61] to capture all the alignment information. Based on this hidden state design, it can detect SNPs, short-to-medium-sized MNPs (including inversions) and INDELs (including tandem repeats). PHV detects variations by individuals. For every 1000 bp of genomic region, PHV loads all reads that belong to one individual falling in this region, and uses them to train an HMM profile by the Baum-Welch algorithm. This way, variation signal from all reads can be represented combinedly by the trained profile. During the training, all possible alignments of each read are systematically considered by the forward-backward algorithm. The read mapping quality is incorporated in the convergence criterion calculation of the Baum-Welch training, so that the influence of the read on the trained profile is weighted by its mapping quality. Through Baum-Welch training, PHV fundamentally resolves the alternative alignment problem common to pileup data-based variation detectors. Moreover, PHV's forward-backward algorithm is written in a banded way. By limiting the calculation in this most frequently called function in a band, PHV not only makes better use of the read mapping information but it also avoids the general low-speed problem of varation detectors engined by HMM or expectation-maximization (EM) methods. While keeping a reasonable running time, PHV has a particular advantage with the error-prone data generated by new sequencing techniques such as PacBio and the low-quality regions of Illumina data. For Illumina data generally, PHV's accuracy of SNP and INDEL detection is on par with most popular tools such as Samtools and GATK.

3.2 Methods

3.2.1 Outline of PHV

PHV uses PHMM to detect genomic variation window by window. PHV's core algorithms are the PHMM construction, Baum-Welch training, the Viterbi algorithm for INDEL sequence calculation and the HMM profile interpretation. Within the sliding window, the main operations of PHV are:

Step 1. Fetch the reference genome sequence, and the reads that fall into the sliding window.

Step 2. Check the pileup data by genomic loci. If at least one locus in the window contains more than two variant alleles, Step 3 to 6 will be applied. Otherwise, this window will be skipped for variation detection.

Step 3. Construct the HMM profile and initialize parameters based on the reference sequence.

Step 4. Use the Baum-Welch algorithm to train the HMM profile. This Baum-Welch algorithm is a parameter training process to maximize the emission probability of all the reads from the HMM profile. During the training, Baum-Welch calls the banded forward-backward algorithm for each read to calculate the emission probability of this read under all possible alignments in the given band. When the Baum-Welch reaches the convergence point, the PHMM parameters are estimated to maximize the overall emission probability of all the reads from the profile.

Step 5. Use the banded Viterbi algorithm to generate the insertion and deletion sequences at candidate INDEL loci and the haplotypes of multiple nucleotide polymorphisms (MNPs).

Step 6. Interpret the trained profile to report variations and calculate the quality of the variation calls from the profile parameters.

The workflow of PHV is shown in Figure 3.1.

3.2.2 The HMM profile construction and parameter initialization

In the sliding window, PHV first fetches the corresponding reference genome sequence, and constructs an HMM profile based on this sequence. If the sliding window size is L, and the reference genome sequence within the window has L nucleotides, PHV will construct a profile as shown in Figure 3.2. The subscripts of the M, I and D states correspond to the coordinates of the reference sequence. I_0 and I_L states are for the insertions before and after the reference sequence in the sliding window. All reads that fall into this window start from the Start (S) state, go through the grey states and are emitted by them, and end at the End (E) state. We note the read sequence as x = $c_0c_1...c_lc_{l+1}$, where c_0 = '^' marks the start of the read and c_{l+1} = '\$' marks the end. As to emissions, the S state always emits the start symbol ('^') and the E state emits the end symbol ('\$'). D states are silent states that do not emit anything. M and I states can emit 'A', 'C', 'G', 'T' or 'N' (unknown nucleotide) with corresponding probabilities.

The emission probabilities of I states will be initialized as $\sim$0.25 (close to but smaller than 0.25) for 'A', 'C', 'G' and 'T', and as 0.001 for 'N' (the summary of the emission probabilities should be equal to 1.), For M states, the emission probabilities will be initialized according to the corresponding reference sequence. For example, if the reference nucleotide at position k is 'A', the probability of emitting the symbol 'A' from the M_k state will be initialized as $\sim$1, and the emission probabilities for other symbols will be initialized as 0.001. Note that the emission probability of any symbol can never be initialized as 0, otherwise these parameters will remain 0 forever while training. The transition probabilities of the directed edges will be initialized as follows:

For $k = 1 \dots L-2$,

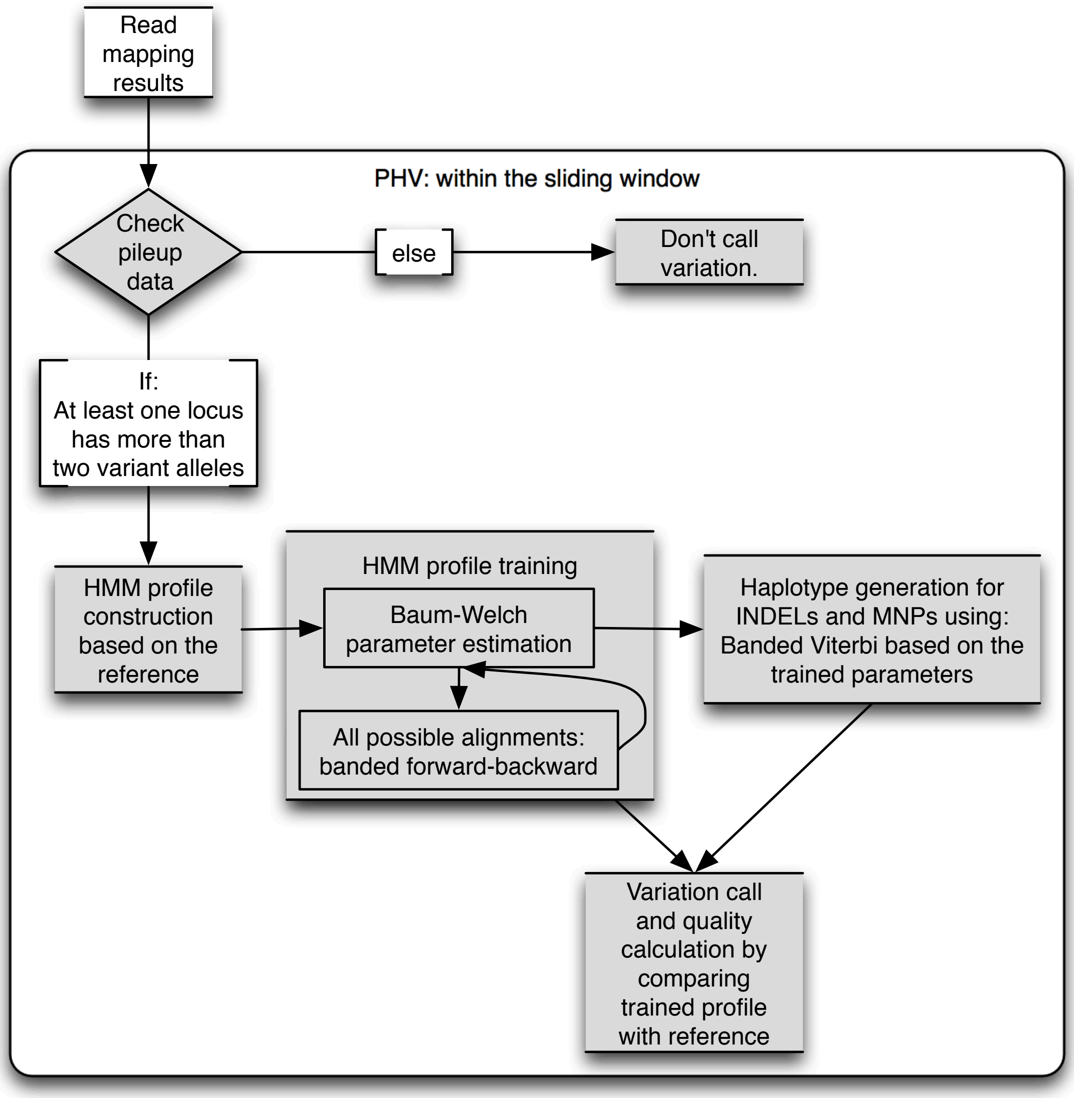

Read mapping results
PHV: within the sliding window
Check pileup data
else
Don't call variation.
If: At least one locus has more than two variant alleles
HMM profile construction based on the reference
HMM profile training
Baum-Welch parameter estimation
All possible alignments: banded forward-backward
Haplotype generation for INDELs and MNPs using: Banded Viterbi based on the trained parameters
Variation call and quality calculation by comparing trained profile with reference

Figure 3.1: The outline of the PHV's algorithm. PHV detects variations in the users' genomic regions of interest by a sliding window. Within the sliding window, PHV constructs an HMM profile according to the reference sequence, trains the profile using all the reads, and calls variations from the trained profile.

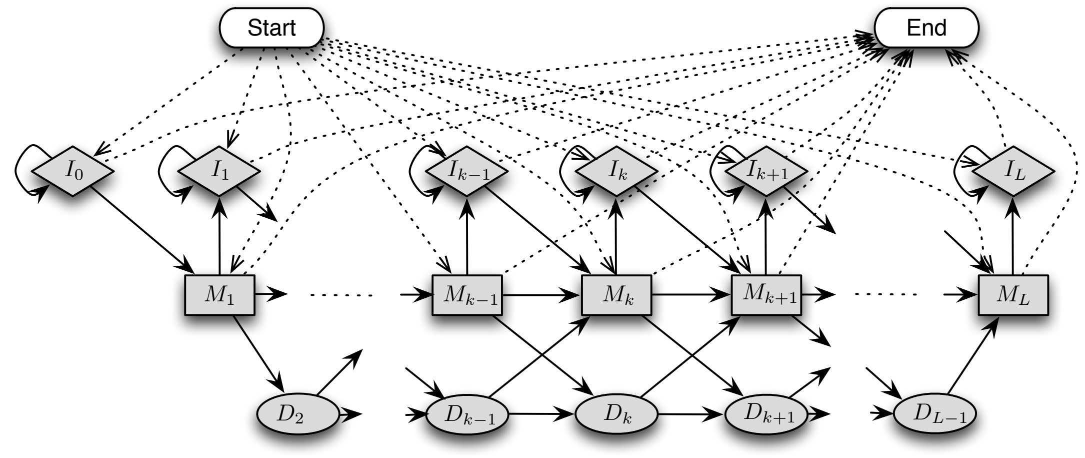


Figure 3.2: The HMM profile. It consists of five types of hidden states: alignment matches (M), insertions to the reference (I), deletions to the reference (D), and alignment start (Start) and end (End). The Start state points to every M and I state, while every M and I state points to the End state. In this profile, immediate connections between insertion and deletion states are not allowed.

$$t_{M_k,M_{k+1}} = (1 - 2 \times \alpha) \times (1 - \gamma) \qquad t_{I_k,M_{k+1}} = 1 - \gamma - \vartheta \times \gamma$$

$$t_{M_k,I_k} = t_{M_k,D_{k+1}} = \alpha \times (1 - \gamma) \qquad t_{I_k,I_k} = \vartheta \times \gamma$$

$$t_{M_k,E} = \gamma \qquad t_{I_k,E} = \gamma$$

$$t_{D_k,M_{k+1}} = \beta \qquad t_{S,M_{k+1}} = \delta / L$$

$$t_{D_k,D_{k+1}} = 1 - \beta \qquad t_{S,I_k} = (1 - \delta)/(L + 1)$$

α, β, γ, ϑ and δ are variables that can be set according to the INDEL error rate of the data. Notably, each set of the above equations has the summary value of 1, which is designed to fit the HMM property. When k = 0, L - 1 and L, the above equations will be adjusted according to the topology of the profile.

3.2.3 Systematically considering alternative alignments of each read: the forward-backward algorithm

Forward-backward algorithms are used to calculate the union probability of emitting each read from the PHMM by all possible alignments. All alternative read alignments are systematically considered in this algorithm. The equations that are used to fill the forward matrix f and the backward matrix b are listed as follows:

Forward algorithm:

Initialization: (i = 0, 1)

$$f_{0,S} = 1. \; f_{1,I_0} = e_{1,I_0} \cdot t_{S,I_0}.$$

$$For \;\; k = 1, \ldots, L: \;\; f_{1,M_k} = e_{1,M_k} \cdot t_{S,M_k}; \;\; f_{1,I_k} = e_{1,I_k} \cdot t_{S,I_k}.$$

Recursion: i = 2, ..., l;

For k = 2, ..., L - 1:

$$f_{i,M_k} = e_{i,M_k} \cdot (t_{M_{k-1},M_k} \cdot f_{i-1,M_{k-1}} + t_{I_{k-1},M_k} \cdot f_{i-1,I_{k-1}} + t_{D_{k-1},M_k} \cdot f_{i-1,D_{k-1}});$$

$$f_{i,I_k} = e_{i,I_k} \cdot (t_{M_k,I_k} \cdot f_{i-1,M_k} + t_{I_k,I_k} \cdot f_{i-1,I_k});$$

$$f_{i,D_k} = t_{M_{k-1},D_k} \cdot f_{i,M_{k-1}} + t_{D_{k-1},D_k} \cdot f_{i,D_{k-1}}.$$

When k = 0, 1 and L, the above equations will be adjusted according to the PHMM topology in Fig 3.2.

Termination: (i = l + 1)

$$f_{l+1,E} = \sum_{k=1}^{L} \cdot (t_{M_k,E} \cdot f_{l,M_k} + t_{I_k,E} \cdot f_{l,I_k}) + t_{I_0,E} \cdot f_{l,I_0}.$$

Backward algorithm:

Initialization: (i = l + 1, l)

$$b_{l+1,E} = 1. \ \ b_{l,I_0} = t_{I_0,E}.$$

$$For\ \ k = 1, \ldots, L: \ \ b_{l,M_k} = t_{M_k,E}; \ \ b_{l,I_k} = t_{I_k,E}.$$

Recursion: (i = l - 1, ..., 1)

For i = l - 1, ..., 2, k = l - 2, ..., 1:

$$b_{i,M_k} = e_{i+1,M_{k+1}} \cdot t_{M_k,M_{k+1}} \cdot b_{i+1,M_{k+1}} + e_{i+1,I_k} \cdot t_{M_k,I_k} \cdot b_{i+1,I_k} + t_{M_k,D_{k+1}} \cdot b_{i,D_{k+1}};$$

$$b_{i,I_k} = e_{i+1,M_{k+1}} \cdot t_{I_k,M_{k+1}} \cdot b_{i+1,M_{k+1}} + e_{i+1,I_k} \cdot t_{I_k,I_k} \cdot b_{i+1,I_k};$$

$$b_{i,D_k} = e_{i+1,M_{k+1}} \cdot t_{D_k,M_{k+1}} \cdot b_{i+1,M_{k+1}} + t_{D_k,D_{k+1}} \cdot b_{i,D_{k+1}}.$$

When k = L, L - 1, 0 or i = 1, the above equations will be adjusted according to the PHMM topology in Fig 3.2.

Termination: (i = 0)

$$b_{0,S} = \sum_{k=1}^{L} \cdot (e_{1,M_k} \cdot t_{S,M_k} \cdot b_{1,M_k} + e_{1,I_k} \cdot t_{S,I_k} \cdot b_{1,I_k}) + e_{1,I_0} \cdot t_{S,I_0} \cdot b_{1,I_0}.$$ [61]

For the above forward-backward algorithm description, i numerates the read sequence, k numerates the reference sequence, l is the length of the read sequence, L is the length of the reference sequence, $e_{i,A}$ denotes the probability of emitting nucleotide c_i from state A, and $t_{A,B}$ denotes the transition probability from state A to state B.

Since the forward-backward algorithm has the time complexity O(L · l), in PHV we use a banded approach to reduce the running time. The banded forward-backward algorithm only computes the values in one diagnostic band in the forward matrix and in the same band in the backward matrix. The beginning position of the band middle line is the read mapping location. The bandwidth is a user-set parameter that limits the length of the detectable INDEL.

3.2.4 HMM profile parameter estimation by maximizing the emission probability of all reads: the Baum-Welch algorithm

Emission probabilities of all the M and I states and transition probabilities of all the solid edges in Figure 3.2 are estimated by the Baum-Welch training. This algorithm is described as follows:

```
Baum-Welch algorithm
Initialization:
        Set emission and transition parameters neutrally as stated in the initialization step.
Recurrence:
        Set all the E and T variables to 0.
        For each read sequence $x^j$ | j = 1 ... n:
                Calculate $f^j_{i,A_k}$ for read sequence $x^j$ using the forward algorithm.
                Calculate $b^j_{i,A_k}$ for read sequence $x^j$ using the backward algorithm.
        Add the contribution of read sequence $x^j$ to E and T as the following equations stated:
                $E_{M_k}(b) = \sum_j \frac{1}{P(x^j)} \sum_{\{i|c^j_i=b\}} f^j_{i,M_k} \cdot b^j_{i,M_k}$;
                $E_{I_k}(b) = \sum_j \frac{1}{P(x^j)} \sum_{\{i|c^j_i=b\}} f^j_{i,I_k} \cdot b^j_{i,I_k}$.

                $T_{M_k,M_{k+1}} = \sum_j \frac{1}{P(x^j)} \sum_{i=1}^{l-1} f^j_{i,M_k} \cdot t_{M_k,M_{k+1}} \cdot e^j_{i+1,M_{k+1}} \cdot b^j_{i+1,M_{k+1}}$;
                $T_{M_k,I_k} = \sum_j \frac{1}{P(x^j)} \sum_{i=1}^{l-1} f^j_{i,M_k} \cdot t_{M_k,I_k} \cdot e^j_{i+1,I_k} \cdot b^j_{i+1,I_k}$;
                $T_{M_k,D_{k+1}} = \sum_j \frac{1}{P(x^j)} \sum_{i=1}^{l} f^j_{i,M_k} \cdot t_{M_k,D_{k+1}} \cdot b^j_{i,D_{k+1}}$;
                $T_{I_k,M_{k+1}} = \sum_j \frac{1}{P(x^j)} \sum_{i=1}^{l-1} f^j_{i,I_k} \cdot t_{I_k,M_{k+1}} \cdot e^j_{i+1,M_{k+1}} \cdot b^j_{i+1,M_{k+1}}$;
                $T_{I_k,I_k} = \sum_j \frac{1}{P(x^j)} \sum_{i=1}^{l-1} f^j_{i,I_k} \cdot t_{I_k,I_k} \cdot e^j_{i+1,I_k} \cdot b^j_{i+1,I_k}$;
                $T_{D_k,M_{k+1}} = \sum_j \frac{1}{P(x^j)} \sum_{i=1}^{l-1} f^j_{i,D_k} \cdot t_{D_k,M_{k+1}} \cdot e^j_{i+1,M_{k+1}} \cdot b^j_{i+1,M_{k+1}}$;
                $T_{D_k,D_{k+1}} = \sum_j \frac{1}{P(x^j)} \sum_{i=1}^{l} f^j_{i,D_k} \cdot t_{D_k,D_{k+1}} \cdot b^j_{i,D_{k+1}}$.
```

Calculate the new model parameters using the following equations:

$e_{A_k}(b) = \frac{E_{A_k}(b)}{\sum_{b'} E_{A_k}(b')}$ and $t_{A,B} = \frac{T_{A,B}}{\sum_{B'} T_{A,B'}}$

Calculate the new log likelihood of the model: $\sum_j 10^{-\frac{Q^j}{10}} \times logP(x^j)$, where Q^j is the mapping quality of read j.

Termination:

Stop if the log likelihood does not change any more or the recurrence exceeds 10 iterations.

For the above Baum-Welch algorithm description, $e_{A_k}(b)$ denotes the probability of emitting symbol b from state A_k; $P(x^j) = f^j_{l+1,E} = b^j_{0,S}$ is the probability of emitting the read sequence x^j from the PHMM.

3.2.5 SNP calling and INDEL break point detection by interpreting the trained profile

When the parameters of the HMM profile are all estimated, PHV checks along the “main path” (the path indicated by the connected match states shown in Figure 3.2) for variations. This strategy is based on the assumption that most read alignments match the reference sequence.

1. If the alignments get through a match state with enough certainty, PHV will check which nucleotides are emitted by this match state. If a nucleotide that is different from the reference allele is likely to be emitted, a SNP will be called. Figure 3.3 shows a simple example of how a SNP is called from the trained profile.

When the transition probability of an edge that connects two match states is not high enough, alternative paths that go through deletion or insertion states will be checked.

2. One path containing deletion state(s) will be monitored whenever one edge that points to a deletion state from a match state has a considerably high transition probability. In this condition, a deletion break-point is detected. In high-entropy regions, all paths from this deletion state will be traced until longer deletions no longer have significant probabilities. From one deletion state, the alignments can continue by going to the next deletion state or returning back

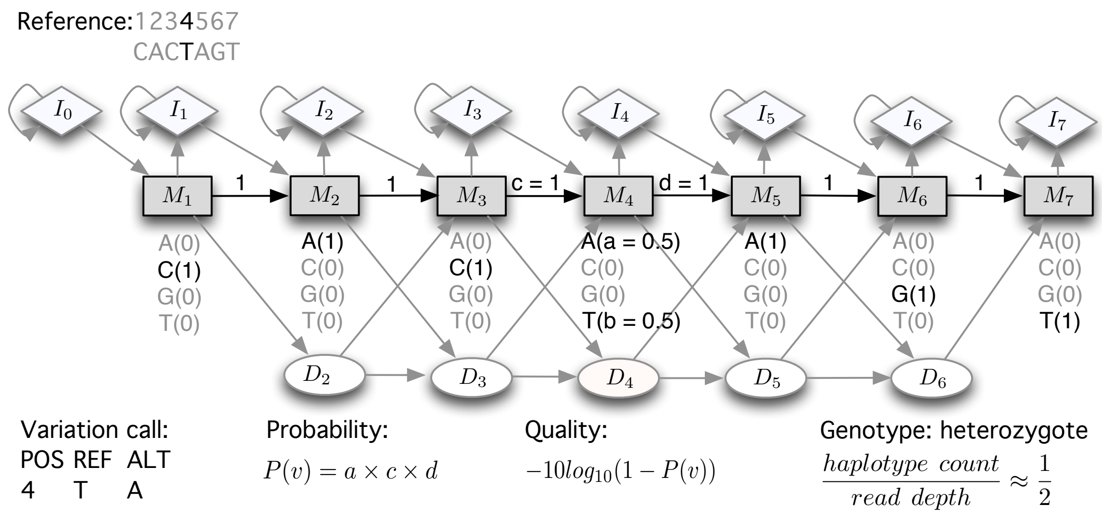


Figure 3.3: A simple example to show how a SNP is called from the trained HMM profile. This profile is initialized from the reference genomic sequence "CACTAGT" and trained by some reads that are aligned to this reference. After the Baum-Welch training, the transition probabilities between match states are all close to 1. This indicates most of the reads are emitted by the path that is constructed by match states. All the match states are most likely to emit the same nucleotides as the reference sequence, except for the M_4 state. M_4 has an approximately 50% probability to emit the reference allele 'T' and another 50% probability to emit an alternative allele 'A'. In this case, a heterozygote SNP at position 4 is called. The quality of this SNP calling is calculated using the formulas listed below the profile.

to the main path. The length of the detected deletion will be determined by which deletion state the alignment returns from. A deletion is called when a path containing deletion states has a high enough probability of emitting the given reads compared to the main path. Figure 3.4 shows such a simple example. In homopolymer or tandem-repeat regions, the deletion length is difficult be to determined; as an effective solution, the Viterbi algorithm is called at the deletion break point to generate the accurate deletion sequence.

3. If some reads can be aligned with insertions, the transition probabilities of the edges that connect to the corresponding insertion states will no longer be 0. Whenever one edge that points to an insertion state from a match state has a high enough transition probability, an insertion break-point is detected. In the example shown in Figure 3.5, the self-transition probability of the I_5 state is 0, indicating only one nucleotide is inserted between reference positions 5 and 6. In this case, a single-nucleotide insertion can be called directly. If the self-transition probability of the insertion state is high, it means a sequence is inserted. In this condition, a Viterbi algorithm will be called to generate the exact insertion sequence.

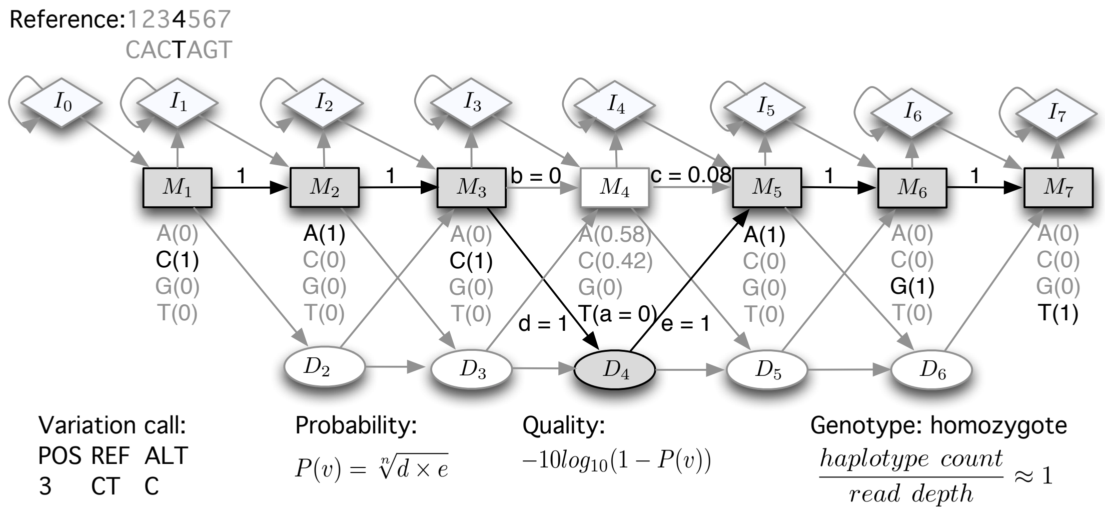


Figure 3.4: A simple example to show how a deletion is called from the trained HMM profile. This profile is initialized from the reference genomic sequence "CACTAGT" and trained by some reads that are aligned to this reference. Since the edge from M_3 to M_4 does not have a high enough transition probability, PHV checks alternative paths after the M_3 state. The edge from M_3 to D_4 has a high transition probability, so the edges from D_4 are further checked. In this example, among D_4's out degrees, only the edge that points to the M_5 state has a high transition probability. In this case, a deletion after position 3 is called. The quality of this deletion call is calculated using the formulas listed below the profile, where n is the length of the deletion (count of the grey D states). The probabilities of taking the path M_3->M_4->M_5, or alternatively M_3->D_4->M_5 are compared, and the genotype of the individual is determined based on this comparison.

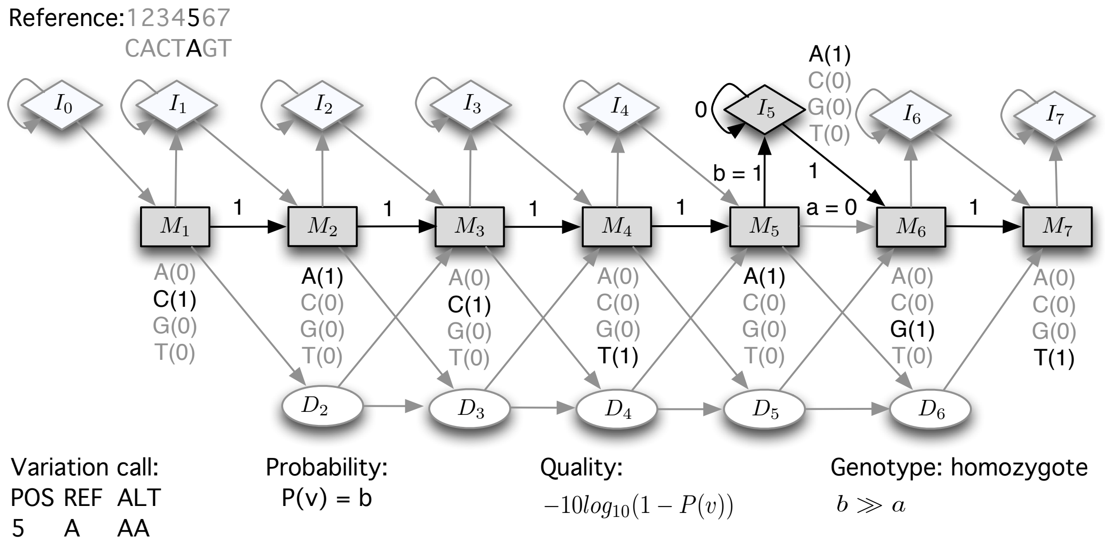


Figure 3.5: A simple example to show how an insertion is called from the trained HMM profile. This profile is initialized from the reference genomic sequence "CACTAGT" and trained by some reads that are aligned to this reference. In the profile, since the edge from M_5 to M_6 does not have a high enough transition probability, PHV checks the alternative paths after the M_5 state. The edge from M_5 to I_5 has a high transition probability. In this case, an insertion after position 5 is called. The quality of this insertion call is calculated using the formulas listed below the profile. The probabilities of taking the path M_5->I_5-> ... ->I_5->M_6, or alternatively M_5->M_6 are compared, and the genotype of the individual is determined based on this comparison.

4. Other kinds of variation can also be represented by the trained profile and can be called by interpreting it accordingly. For example, multiple-nucleotide polymorphisms (MNP), including inversions, are represented as multiple connected match states emitting alternative alleles; tandem repeats are represented as an insertion or deletion.

3.2.6 Generation of accurate INDEL sequences and of genotypes of heterozygote MNPs using the Viterbi algorithm

After the parameters of the PHMM are estimated, all the insert sequences, the delete sequences in homopolymer and tandem-repeat regions, and the genotypes of heterozygote MNPs at the candidate loci are generated by the banded Viterbi algorithm. In this Viterbi algorithm, the band is set identically to the forward-backward algorithm to achieve consistency.

Viterbi algorithm

Initialization: (i = 0) $v_{0,S} = 1$.

Recursion: i = 1, … , l;

$$For\ \ k = 3, \ldots, L - 1:$$

$$v_{i,M_k} = e_{i,M_k} \cdot max \begin{cases} v_{i-1,M_{k-1}} \cdot t_{M_{k-1},M_k} \\ v_{i-1,I_{k-1}} \cdot t_{I_{k-1},M_k} \\ v_{i,D_{k-1}} \cdot t_{D_{k-1},M_k} \end{cases};$$

$$v_{i,I_k} = e_{i,I_k} \cdot max \begin{cases} v_{i-1,M_k} \cdot t_{M_k,I_k} \\ v_{i-1,I_k} \cdot t_{I_k,I_k} \end{cases};$$

$$v_{i,D_k} = max \begin{cases} v_{i,M_{k-1}} \cdot t_{M_{k-1},D_k} \\ v_{i,D_{k-1}} \cdot t_{D_{k-1},D_k} \end{cases}.$$

When $k = 0, 1, 2$ and L, the above equations will be adjusted according to the PHMM topology in Fig3.2.

```
Termination: (i = l + 1)
        $v_{l+1,E} = \max\{v_{l,A_k} \cdot t_{A_k}, E \mid A = M, I; k = 0, ..., L; \text{except } \mathcal{M}_0\}$.
Back track from state E:
        (The back track result is a state sequence P. The elements in P are noted as $P_j$,
        $P_j \in \mathcal{M}, I, D$.)
        When i = l, j = 0:
                $P_0 = argmax_{A_k \in \{M,I\}} \{t_{A_k,E} \cdot v_{l+1,E} \mid k = 0, ..., L\}$
        While i $\geq$ 0 do:
                ++j;
                If current state is $\mathcal{M}_k$:
                        $P_j = argmax_{A_k \in \{M,I,D\}}\{t_{A_{k-1},M_k} \cdot v_{i,M_k}\}$
                        - -i;
                If current state is $I_k$:
                        $P_j = argmax_{A_k \in \{M,I\}}\{t_{A_k,I_k} \cdot v_{i,I_k}\}$
                        - -i;
                If current state is $D_k$:
                        $P_j = argmax_{A_k \in \{M,D\}}\{t_{A_{k-1},D_k} \cdot v_{i,D_k}\}$
```

For the above Viterbi algorithm description, i numerates the read sequence; k numerates the reference sequence.

3.2.7 Calling INDELs in homopolymer and tandem repeat regions

Homopolymer and tandem-repeat regions are prone to causing sequencing errors, leading to inconsistent homopolymer length or repeat segment count in the reads. This is a difficult problem for traditional INDEL detection tools based on pileup data, but it can be naturally resolved by PHV. True variations are represented as common signals from the reads; while sequencing errors affect reads differently. The statistical results of all the reads can be told by the trained profile, and INDEL breakpoints can be detected in such low entropy regions if a true variation

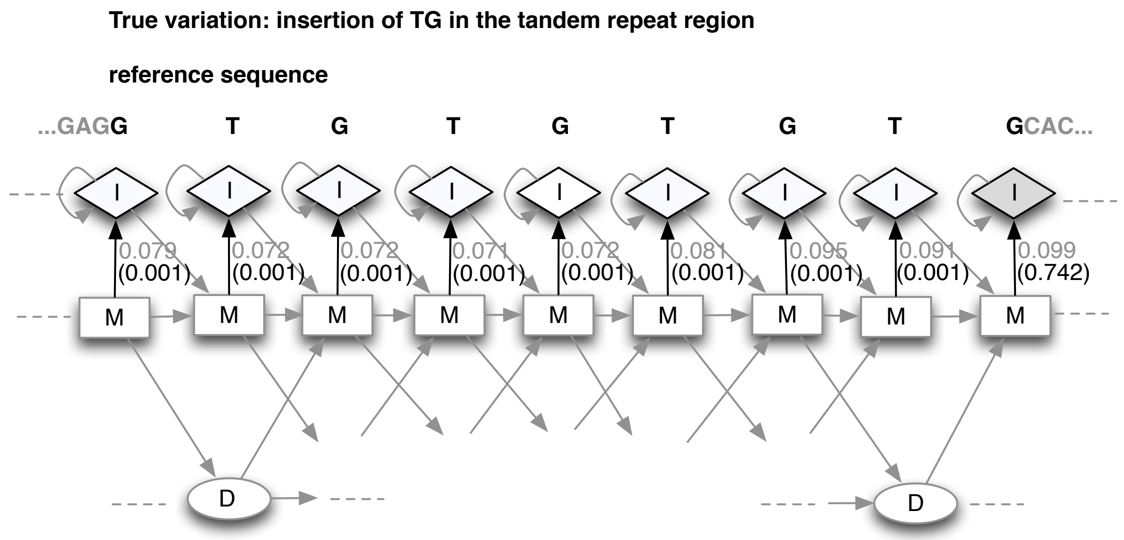


Figure 3.6: An example of grouping the INDEL signal in homopolymer or tandem-repeat regions. This figure shows a tandem-repeat region of the HMM profile, which is trained by the simulated Illumina sequenced reads of human chromosome 20. The corresponding reference sequence is shown by the black letters above the trained profile. The true variation in this tandem-repeat region is an insertion of "TG". The grey numbers that are by the edges from states M to states I indicate the transition probabilities directly after the Baum-Welch training. When PHV detects the homopolymer or tandem-repeat region, it groups the INDEL signal to the edge that carries the strongest signal (here, it is the edge that points to the grey insertion state). The black numbers in the round brackets are the transition probabilities after this adjustment.

exists. With the help of the Viterbi algorithm, an accurate INDEL sequence can be generated for each read based on the trained profile.

If the sequence of the INDEL has a pattern similar to the homopolymer or tandem-repeat unit in which the INDEL happens (e.g. an insertion of "ATAT" in the tandem repeat region "ATATATATATAT"), an INDEL signal-grouping step is needed to detect this INDEL. Since this kind of INDEL has an equal chance of happening at alternative reference positions, the INDEL signal is evenly spread in the whole low-entropy region in the trained HMM profile of PHV. In such a case, such INDELs cannot be directly detected. Therefore, whenever a homopolymer or tandem-repeat region is found in the sliding window in PHV, the INDEL signal is grouped to the strongest point in such regions after the profile is trained. One example is shown in Figure 3.6.

3.3 Results

To estimate the specificity and sensitivity of PHV's variation detection, we applied PHV to both simulated sequencing data and to real data, and we compared PHV's performance to the currently most popular variation detection tools, GATK, Samtools, and Freebayes.

The true variation between an individual genome and the reference genome is always unknown, while for the simulated genome, everything is known for certain. Theoretically, the true variation has to be known for us to evaluate the results of variation detection. This is why we do the performance comparison using simulated genomic data.

On the other hand, the read simulation cannot always emulate the sequencing process, and in some cases the real data can be far more complicated than the simulated genome. This is mainly caused by four reasons: 1) The real variations carried by individual genomes can be complicated beyond the ability of the simulation programs. 2) The sequencing errors generated by simulators tend to be random. But in reality, sequencing errors are more likely to happen at certain genomic regions or at certain positions of the reads. The simulators are still not good at representing such special error patterns. 3) Though read simulators are designed for specific sequencing platforms according to the property of the sequencing technique, some real sequencing errors are still unpredictable. 4) Most existing simulators are good at simulating SNP errors, but their simulation of the more complicated INDEL sequencing errors is still far from ideal. Simulated INDEL errors are usually very different from those truly happening during the sequencing process. Therefore, it is necessary to also test PHV with real data. Though in real data the true genomic variations cannot be known with certainty, we can still use certain methods to show the software performance.

3.3.1 PHV's variation detection on simulated data

We first tested PHV's performance with simulated Illumina sequence data of human chromosome 20. We randomly implanted 1542 SNPs, 16 MNPs, 186 insertions and 214 deletions (the

INDELs have different sizes) in the human reference genome of chromosome 20 (hg37) using mutatrix (https://github.com/ekg/mutatrix) to form an artificial individual genome. The insert size varies from 1 bp to 13 bp with an average length of 1.65 bp and a standard deviation of 1.46 bp. The deletion size varies from 1 bp to 49 bp with an average length of 1.78 bp and a standard deviation of 3.44 bp. We simulated 100 bp long Illumina sequence reads from this artificial individual with an average coverage of 9.42 using the Mason with default error module.[79,78,77] Then we aligned the reads from the mutant genome to the reference genome with Mosaik, and called variations from this aligned BAM file with PHV, GATK, Samtools and Freebayes. The command lines used by these tools are listed as follows:

```
PHV:        phv -s16 REFERENCE.fasta ALIGNMENT.bam 20 > RESULT.vcf
GATK:       java -jar gatk/GenomeAnalysisTKLite-2.2-15-g4828906/GenomeAnalysisTKLite.jar
            -T UnifiedGenotyper -glm BOTH -R REFERENCE.fasta -I ALIGNMENT.bam
            -o RESULT.vcf
Samtools:   samtools mpileup -uf REFERENCE.fasta ALIGNMENT.bam | bcftools view
            -bvcg -> samtools.raw.bcf
            bcftools view samtools.raw.bcf | vcfutils.pl varFilter -D 100 > RESULT.vcf
Freebayes:  freebayes -b ALIGNMENT.bam -f REFERENCE.fasta -v RESULT.vcf
```

After the variations were called, we split each result file into three dependent on the type of variation: 1) SNPs and MNPs, 2) insertions, and 3) deletions. For each type of variation, we compared the calls from different tools using a receiver operating characteristic (ROC) curve. The SNP and MNP call comparison is shown in Figure 3.7. According to this ROC curve, PHV's performance is the best: a quality threshold of PHV achieves the best sensitivity and false detect rate (FDR) combination compared to all others.

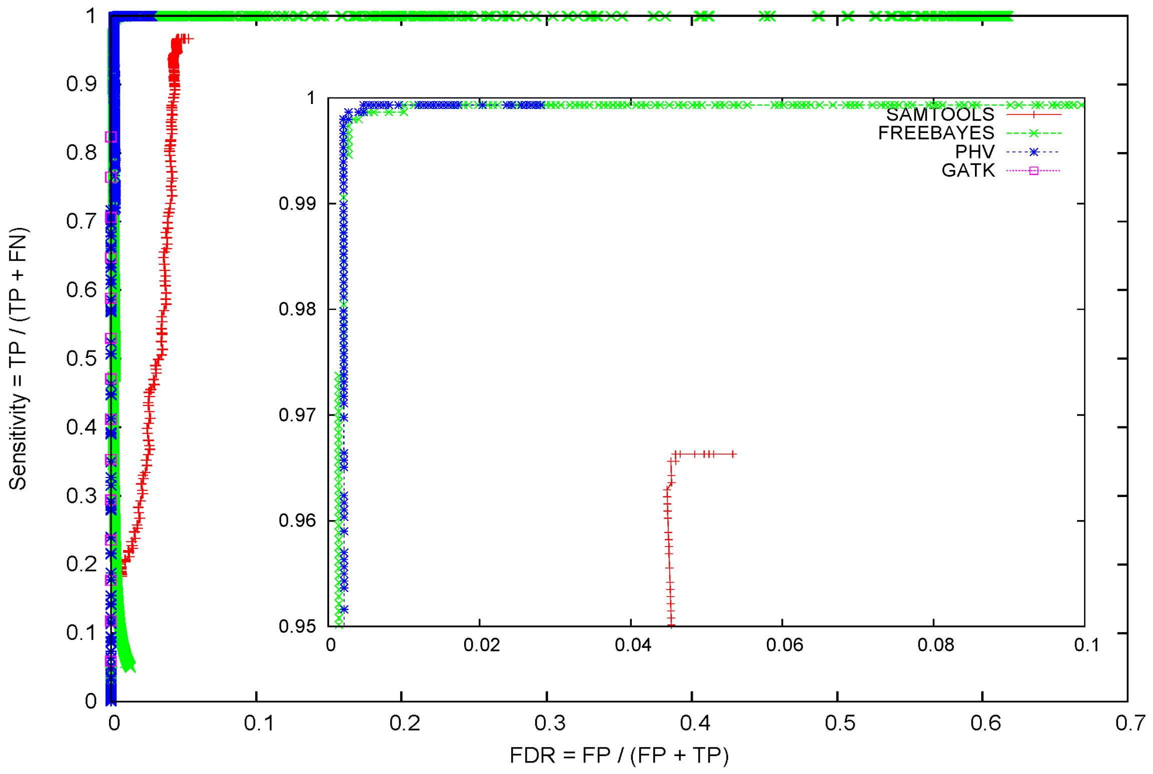


Figure 3.7: The SNP and MNP call comparison of PHV, GATK, Samtools and Freebayes. The x-axis shows the false detect rate (FDR). FDR = FP/(FP + TP), where FP is the false-positive count, TP is the true-positive count. The y-axis shows the sensitivity of the tools. Sensitivity = TP/(TP + FN), where FN is the false-negative count. The red, green, blue and pink lines correspond to Samtools, Freebayes, PHV and GATK, respectively. To generate this ROC curve, SNPs and MNPs were cumulatively counted by quality thresholds, and FP, TP and FN were counted by comparison to the true variations given by mutatrix. In the ROC curve, each point corresponds to a quality threshold. The inner graph is a zoom-in figure of the left top part of the large graph.

3.3.2 PHV's variation detection on real data

Considering the challenge of working out the answer keys of variation calling on real data, we chose to use the data set on which current SNP/INDEL callers can work best, the Illumina platform deep sequenced single individual NA12878. Before systematic real data testing, we did some quick testing of PHV on small genomic regions of this data set, and compared its results to Samtools' and Freebayes'. By choosing the software that we are most familiar with, we can guarantee proper software usage. Also, by using the standard testing data set of the 1000 Genomes Project (http://www.broadinstitute.org/gsa/wiki/index.php/NA12878_test_data), we expected the results of SNP and INDEL calling generated by Samtools and Freebayes to be close to the truth.

Figure 3.8 shows the result of the comparison of SNP and INDEL calling on two randomly chosen 10 Kbp regions of human chromosome 20. Samtools used default parameters. Freebayes used parameter –left-align-indels. Its results were filtered by a quality score of > 10. PHV used a band width of 16. Its results were filtered by a quality score of > 2.5. The majority (79) of the variation loci detected by the three software packages were the same. I checked ten loci among these 79 shared loci. All three software returned the exact same variant allele calls for these ten loci, so I believe the three software gave exactly the same call for most of the 79 loci. I checked all loci called by only one or two software packages by looking at the alignments carefully, and found all variations called by both PHV and Freebayes and most of the ones called by PHV only are good calls, whereas most of the variation reported by Samtools and Freebayes but not by PHV are bad calls (Figure 3.8 panel A). Here, I defined the good calls as variant alleles strongly supported by the alignments, meaning the read depth is at least six, and the reads containing that variant allele are at least one third of the total reads. I defined the bad calls as variation sites without enough evidence supporting the existence of that variant allele in NA12878. Figure 3.8 panel B shows an example of a good call by Freebayes and PHV (this variation was not deteced by Samtools). The outputs of Freebayes and PHV at this site are simply shown as follows:

Freebayes:					
CHROM	POS	ID	REF	ALT	QUAL
20	1000562	.	TTTCA	T	1944.68
PHV:					
CHROM	POS	ID	REF	ALT	QUAL
20	1000562	.	TTTCA	T	3.759449
20	1000566	.	ATTCA	A	4.485771

At this site, Freebayes gave one call, the deletion of the sequence"TTCA" downstream of the genome location 1000562 of chromosome 20 (labeled by the left red line in Figure 3.8 panel B); while PHV gave two calls, deletions of the sequence "TTCA" downstream of the genome locations 1000562 and 1000566 (labeled by the right red line in Figure 3.8 panel B), respectively, with similar quality scores. The deletion downstream of position 1000562 is directly supported by the alignments. However, the reference sequence at this site has a repeat of "TTCA". Under this condition, the deletion can happen at either the first or the second locus. Based on all possible alignments of the reads, PHV gave a more objective call of the variation. The alternative calls given by PHV can also be combined by post-processing if needed. Figure 2.2 panel C shows a variation locus wrongly called by Freebayes and Samtools but not by PHV. The outputs of Freebayes and Samtools at this site are simply shown as follows:

Samtools:					
CHROM	POS	ID	REF	ALT	QUAL
20	10008028	.	A	T	30.15
20	10008029	.	T	A	107.84

Freebayes:					
CHROM	POS	ID	REF	ALT	QUAL
20	10008028	.	AT	TA	50000
20	10008029	.	T	TA	2715.55

Though both Samtools and Freebayes gave high quality scores at their detected variation loci, if we check the alignments carefully, we will find the SNP “T” at locus 10008028, which is supported by some reads, should actually be aligned to locus 10008029, so that only an insertion of“A” after the reference “T” at locus 10008029 is supported by the reads. This kind of misleading alignment is easily resolved by PHV, and does not lead to a false positive call. Notably, shown here are just two examples representing the advantage of the PHMM algorithm in SNP and INDEL calling.

As a next step, I tested PHV’s potential for dealing with sequencing data carrying a higher INDEL error rate. I used the 454 sequenced NA12878 data for testing (with an indel error rate of 1%). I simply chose the following longest target regions on chromosome 20 and filtered out the parts with a read depth of less than six.

20	3075422	3077716	target.7716
20	33553632	33556247	target.7791
20	43944639	43946096	target.7813
20	61664317	61668028	target.7879

Seeing how the testing data set is very small and the 454 data is noisy (so that the reliable calls

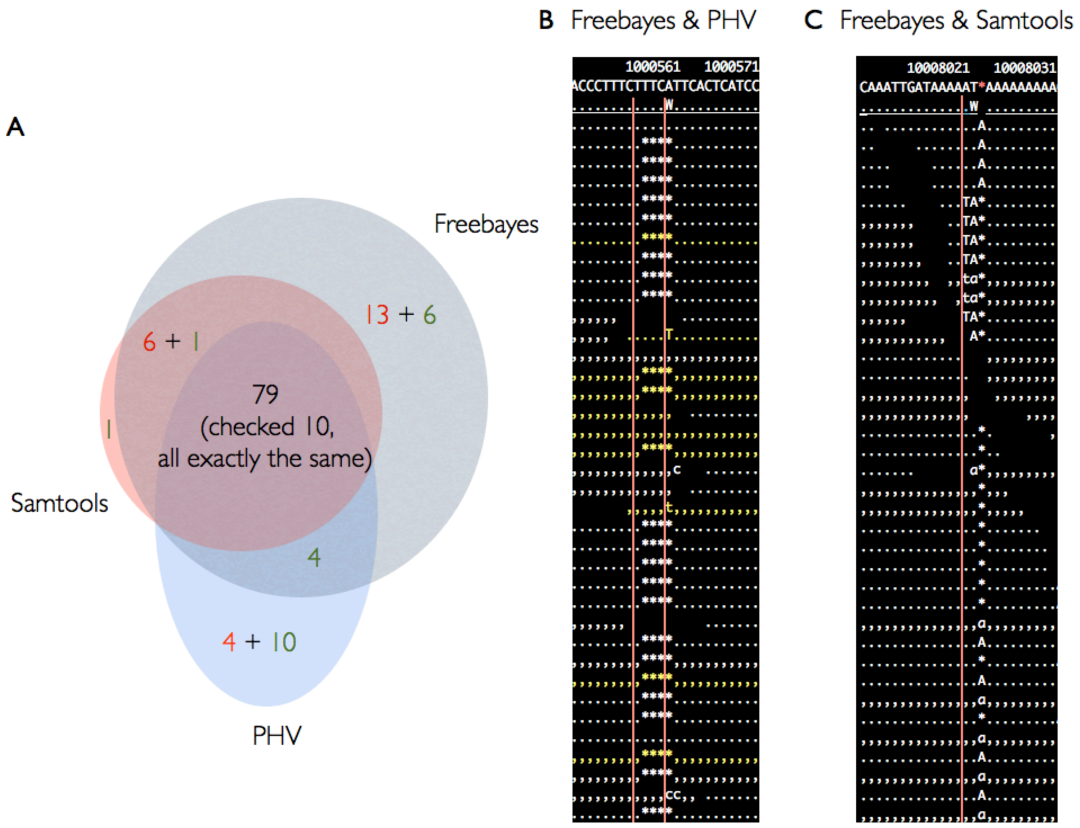


Figure 3.8: Comparison of the SNP and INDEL calling results of Samtools, Freebayes and PHV on two randomly chosen 10 Kbp genomic regions of chromosome 20 of the NA12878 Illumina sequencing data. (A) shows the Venn diagram of the variation loci called by the three softwares. The red numbers show the counts of bad calls, while the green numbers show the counts of good calls. (B) shows the alignments of one example variation locus called by Freebayes and PHV (labeled by the left red line) and one locus called by PHV only (labeled by the right red line). (C) shows the alignments of one example variation locus wrongly called by both Freebayes and Samtools.

are limited), the detected variation sites reported by the three tools are almost exactly the same. All three gave eight consensus good calls. Only Samtools gave two extra calls, one good call and one bad call. Though the results of the three tools are similar, it is worth to mention that the complexity of the parameter settings is different. The command lines used by the three tools to generate such outputs are as follows:

```
PHV:         phv -s16 REFERENCE.fasta ALIGNMENT.bam REGION |
             vcffilter -f "QUAL > 2.5" > RESULT.vcf
Samtools:    samtools mpileup -ugf REFERENCE.fasta ALIGNMENT.bam -r
             REGION | bcftools view -bvcg -> samtools.raw.bcf
             bcftools view samtools.raw.bcf | vcfutils.pl varFilter
             -D 200 | vcffilter -f "QUAL > 10" > RESULT.vcf
Freebayes:   freebayes -b ALIGNMENT.bam -f REFERENCE.fasta -v
             freebayes.vcf –region REGION -O –left-align-indels
             –min-alt-fraction 0.5
             cat freebayes.vcf | vcffilter -f "QUAL > 65" > RESULT.vcf
```

Among the three tools, Samtools and PHV both used the default setting, while Freebayes' best performance command lines were worked out through more than ten trials and conversations with the author. The results of our comparisons proved PHV's potential ability of detecting variation using low-quality sequencing data as well as its robustness and simple usage.

3.3.3 Software speed and availability

To call variation on human chromosome 20 with a read coverage of 9.42, PHV takes about five hours for bandwidth 16 (for detecting INDELs up to 16 bp long) and about 16 hours for bandwidth 50. This speed is similar to GATK's, and is very fast compared to any HMM-based variation caller (e.g. Dindel[4] and PyroHMMvar[118] or lobSTR[36]). PHV is open source software under MIT license. All of PHV's source code is publicly available at GitHub (https://github.com/mengyao/PHMM-SNP-INDEL-caller).

3.4 Discussion

3.4.1 Influential method details

One common issue of the HMM algorithms is the numerical stability. Even using the "double" variable type, PHV may run into the floating-point underflow problem when multiplying many probabilities in the forward, backward or Viterbi algorithms. To solve this problem, we compute $\tilde{f}_{i,A_k} = \frac{f_{i,A_k}}{\sum_{A'} f_{i,A'}}$ and $\tilde{b}_{i,A_k} = \frac{b_{i,A_k}}{\sum_{A'} f_{i,A'}}$ instead of f_{i,A_k} and b_{i,A_k} in the forward-backward algorithm, and log-transformed v_{i,A_k} in the Viterbi algorithm during the real computational implementation. Similar scaling methods are mentioned in R. Durbin's book[24].

In PHV, the read information and the alignment information from the BAM file are extensively used during variation calling compared to most other HMM-based variation detectors. The read mapping quality generated by the mapping tool is used during the Baum-Welch training to weigh the contribution of the read to the trained HMM profile. The read mapping location is used to determine the core of the band in the banded forward-backward algorithm and in the banded Viterbi algorithm, as shown in Figure 3.9. PHV is the first variation detection tool that uses a banded PHMM. Through this banded approach, the running time of the PHMM algorithms is dramatically reduced, and practical variation detection on a whole-genome level is realized using PHMM. PHV is the first PHMM-based variation detection tool that can do fast whole-genome variation calling without pre-training, which would require extra data.

In PHV, the Baum-Welch training is triggered by a very loose criterion: more than two reads have a different signal (SNP or INDEL with a base quality higher than nine) compared to the reference genome at any position in the sliding window. Most popular variation callers including GATK and Freebayes only do local realignment when multiple INDEL signals are shown in the pileup data. If a true INDEL is misaligned as a mismatch by the mapping tool and an

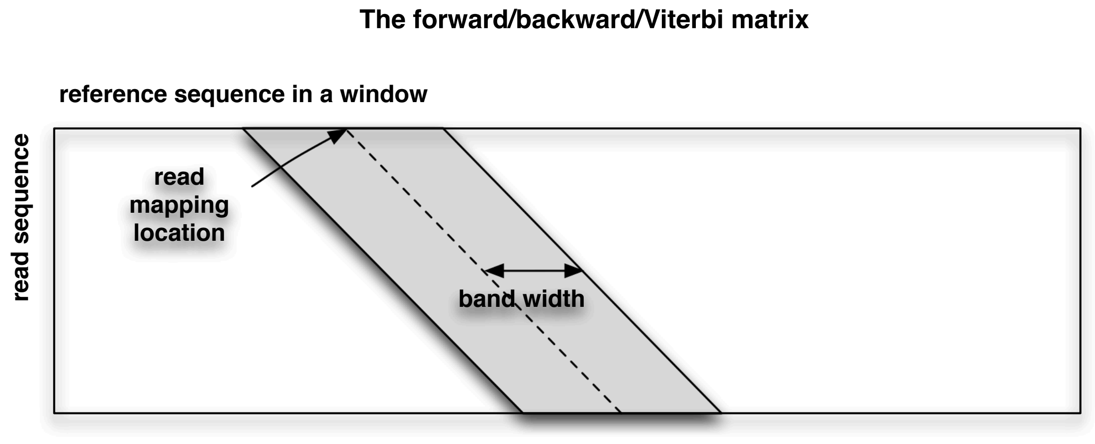

The forward/backward/Viterbi matrix
reference sequence in a window
read sequence
read mapping location
band width

Figure 3.9: The banded forward/backward/Viterbi matrix. The big rectangle indicates the forward, backward or Viterbi matrix. The long edge corresponds to the reference sequence in the sliding window. The short edge corresponds to the read sequence. In either of the standard forward, backward or Viterbi algorithm, all values in such a matrix need to be calculated. In PHV, we use a banded approach to speed up the matrix calculation by eliminating the calculation in a user-customized band. The band core is determined by the read mapping location. The bandwidth is set by the user through a command line parameter.

INDEL is not shown in the original pileup data, these traditional tools will have no chance of detecting this INDEL. In this described case, on the other hand, PHV still can find the INDEL with the Baum-Welch training. Also, the HMM parameters of the trained profile are a systematic, statistical summary of all possible alignments of the reads, rather than one better alignment being compared to the original one. Compared to traditional tools, this difference results in a special advantage when alternative alignments are similarly good. In this case, PHV can give a more objective report of the variation with the quality score calculated from the probability of all the read alignments that support this variation in the given band.

3.4.2 Insights from the Result

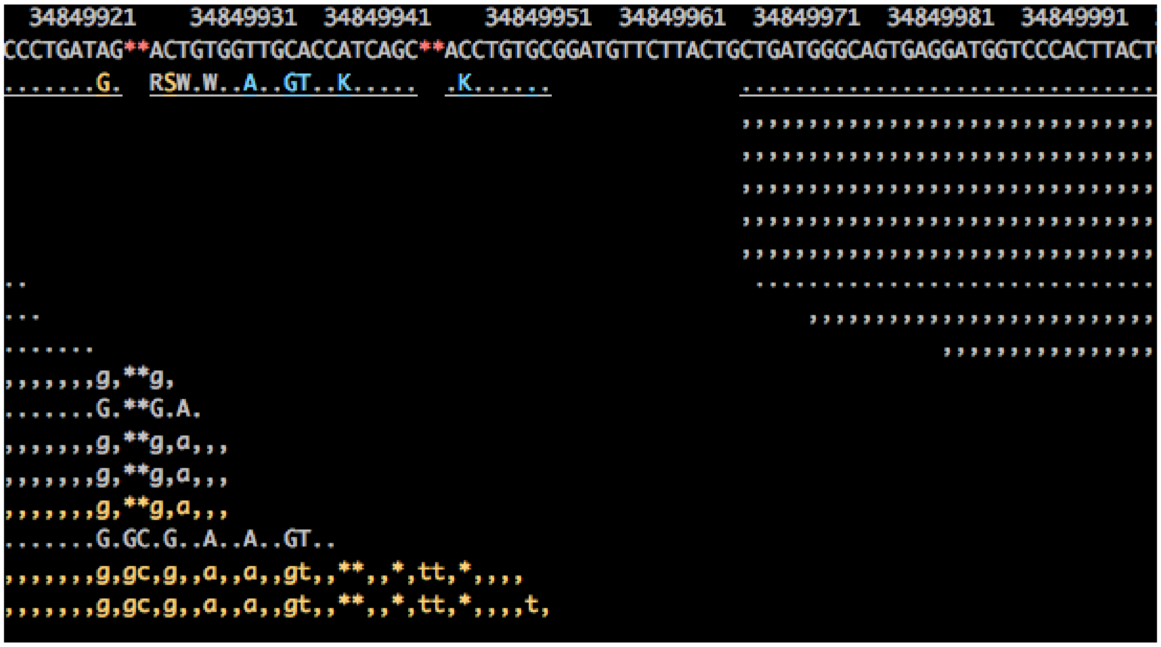
We compared PHV's deletion calling on simulated, Illumina-sequenced data of human chromosome 20 to both GATK and Samtools. The data processing procedure is described in the Result section. PHV returned 16 false negative deletion calls and two false positive deletion calls on this data set (the true count of deletions is 214). I checked these errors by rerunning PHV multiple times on the corresponding problematic locations with a surrounding genomic sequence of 100 bp, 200 bp, 500 bp, 1 Kbp and 2 Kbp, respectively. None of these errors reoccurred when I ran PHV locally, independent of the length of the surrounding region. Then I changed the sliding window length of PHV, and found the count of such wrong calls changing accordingly. These results indicate that all these deletion errors may relate to the sliding window. If we filter out these errors, PHV's deletion call on this simulated data is error-free at its best quality threshold, and its ROC curve is better than both GATK's and Samtools'. GATK had one false negative deletion call at its best quality threshold, while Samtools had one false positive deletion call and three false negative deletion calls at its best quality threshold.

Notably, on this testing data set both GATK and Samtools did not call a 49 bp true deletion, but PHV called it accurately. Figure 3.10 is a screenshot of Samtools tview that shows the alignment at the locus of this 49 bp deletion. From this screenshot we can tell that Mosaik misaligned the corresponding reads with many mismatches and small INDELs at a wrong location. The true deletion is not shown in the alignment. Based on this alignment result, GATK called several low-quality SNPs, and Samtools did not call anything at this region, while PHV correctly called the following deletion (with a band width of 50):

CHROM	POS	REF	ALT	QUAL
20	34849925	TAGACTGTGGTTGCACCATCAGCACCTGTGCGGATGTTCTTACTGCTGAT	T	10.0342

This is a promising result that demonstrates PHV's ability to avoid the misleading information from the alignment result and to give correct medium-sized INDEL calls. To confirm PHV's performance on detecting such variation, we still need more tests on real data.

For the insertion calls on the simulated data of human chromosome 20, we also compared PHV's result with GATK and Samtools. There are 186 true insertions in the mutant genome. In total, PHV and Samtools both called 185 true insertions (one false-negative call) without any false-positive calls. GATK called 182 true insertions (four false-negative calls) without any false-positive calls. This means PHV achieved good sensitivity and specificity for the insertion calling. However, PHV's ROC curve is not as good as those of GATK or Samtools. Samtools has one false-negative call and zero false-positive calls, GATK has four false-negative calls and



```
  34849921    34849931  34849941    34849951  34849961  34849971  34849981  34849991
CCCTGATAG**ACTGTGGTTGCACCATCAGC**ACCTGTGCGGATGTTCTTACTGCTGATGGGCAGTGAGGATGGTCCCACTTACT
.......G.  RSW.W..A..GT..K.....  .K......              ...............................
                                                       ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
                                                       ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
                                                       ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
                                                       ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
                                                       ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
..                                                      ..............................
...                                                         ,,,,,,,,,,,,,,,,,,,,,,,,,,,
.......                                                               ,,,,,,,,,,,,,,,,
,,,,,,,g,**g,
.......G.**G.A.
,,,,,,,g,**g,a,,,
,,,,,,,g,**g,a,,,
,,,,,,,g,**g,a,,,
.......G.GC.G..A..A..GT..
,,,,,,,g,gc,g,,a,,a,,gt,,**,,*,tt,*,,,,
,,,,,,,g,gc,g,,a,,a,,gt,,**,,*,tt,*,,,,t,
```

Figure 3.10: A screen shot of Samtools tview. This screen shot shows the read alignment on human chromosome 20 from position 34,849,919 to 34,850,000. The reads are a simulated, Illumina-sequenced mutant genome, and were aligned by Mosaik. The first line shows the reference coordinates. The second line is the reference sequence. The third line is the consensus read sequence. The following lines are reads. '.' indicates forward aligned reads with the same nucleotide as the reference. ',' indicates the reverse complement aligned reads with the same nucleotide as the reference. White colored reads have a mapping quality of >= 30. Yellow colored reads have a mapping quality of 20-29.

zero false-positive calls at their best quality threshold, respectively. PHV has four false-negative calls and 27 false-positive calls at its best quality threshold. This result suggests PHV's method of calculating the insertion quality is not proper and needs to be improved.

3.4.3 Future work

The first issue to address is solving the known problems of PHV, which were discovered through the test on simulated data. We will fix the false-positive bug and improve the calculation of the insertion quality. Currently, insertion quality is calculated from a combined evaluation of the transition probabilities on the corresponding edges of the trained HMM profile and the ratio of differently aligned reads (the reads may be aligned to support the insertion or the reference allele). To improve this quality calculation, we plan to try two solutions: the first is to use the alignment qualities generated by the Viterbi algorithm to weigh the contribution of the read to the observed haplotype (insertion or the reference allele). If this simple improvement can generate ideal results, we will stop here. If not, we would like to try using the Bayesian method to calculate the insertion genotype and quality based on the trained profile and each read alignment by the Viterbi algorithm. This Bayesion method is similar to that used in PyroHMMvar[118]. For a diploid genome, the genotype is $g = (h_1, h_2)$, where h_1 and h_2 represent the two chromosomes. r_i denotes the ith read sequenced from one of the two chromosomes, and R denotes all the read sequences. p(g|R) is the probability of the genotype based on the observed read sequences. We will choose the genotype g* that maximize p(g|R) as the genotype and calculate the genotype quality based on p (g*|R).

$$p(g|R) = p(R|g)p(g),$$

$$p(R|g) = \prod_{i=1}^{n}[p(r_i|h_1)p(h_1|g) + p(r_i|h_2)p(h_2|g)],$$

where $p(h_1|g) = p(h_2|g) = 0.5, p(r_i|h_1) and p(r_i|h_2)$ can be assigned as the transition probability of the edge $M_b- > M_{b+1}$ or $M_b- > I_b$ on the trained profile depending on the assumed genotype g, where b is the insertion position. If we assume the insertion rate is q_g, we can set $p(g) = q_g$ for heterozygous insertions and $p(g) = \frac{1}{2}q_g$ for homozygouss insertion based on the Wright-Fisher model.

To further tune and benchmark PHV with real data, we plan to use two data sets: 1) the Illumina and PacBio sequenced NA12878, and 2) the Illumina and PacBio sequenced male haploid genome CHM1. NA12878 is the mother of the CEU trio. This individual has been whole-genome deep-sequenced multiple times. The genomic variations of NA12878 have been called by most mainstream variation detectors. The high confidence calls have been generated by evaluating the calls from different tools, and are publically available. This makes NA12878 a good test and tuning dataset for PHV's development. The CHM1 data were obtained from the human haploid cell line CHM1htert. CHM cells are from abnormal human gestation: the fertilization of an genome-free egg by a haploid sperm that duplicates to restore diploidy.[46] As a result, the sequencing data of CHM1 are haploid genome data, which has been used to benchmark various popular mapping tools and variation detection tools. (http://arxiv.org/pdf/1404.0929.pdf) The basic idea of this test is that any heterozygous variation called from CHM1 data is suspected to be wrong. The existing benchmarking results show all the popular tools called many heterozygous variations. Among them, 80-90% of the heterozygous INDELs and up to 60% of the heterozygous SNPs are in the low-complexity regions, which is only 2% of the human genome. From these numbers, we can see that existing variation detection tools do not work well in regions of low complexity. One of the main reasons why we developped PHV was to achieve accurate calls in low complexity genomic regions, therefore the CHM1 data is a very suitable dataset for us to test and tune PHV, too. For the tuning strategy, we plan to check each major change of PHV with simulated data sets and real data sets together. Only this way can we get a more comprehensive understanding of the influences of any changes on the results of variation calling.

Moreover, to prepare for the expected difficulties in low-complexity genomic regions, we will consider using simulated annealing for the Baum-Welch training to prevent the system from falling into local optima. This function will only be called when necessary. If needed, we can also speed up PHV with multiple-thread technique to parallel variation calling at different genomic regions. On the other hand, to complete the function of PHV, the Viterbi algorithm can also be used to return a set of adjusted alignments that maximize the probability of emitting all the reads from the trained profile. The adjusted BAM file[64] represents the best alignment based on the given data (reference and reads), and can assist biologists in viewing the evidence of their variants of interest in a straightforward manner.

4 Concluding Remarks

4.1 Summary

During my PhD studies under the supervision of my mentor Dr. Gabor T. Marth, I mainly worked on the detection of genomic variation using NGS data through the SSW and PHV projects. I also contributed to variation detection in the cancer genome project (including germ-line and somatic mutation) and the development of detection methods. In this thesis, I focused on the objectives of my two main projects:

SSW Library: A SIMD Smith-Waterman C/C++/Python library

PHV: an accurate caller of short-to-medium-sized genomic variation based on PHMM

Here I would like to summarize the significant contribution of these two novel tools to genomic variation detection studies, the powerful mathematical models and algorithms that I used, and the high performance computing techniques that I applied during the tool development.

To maintain the brevity of this thesis, I am not including the source code of my programs into this body of writing. Instead, I provide all my programs along with the software development records on GitHub. The relevant information is also summarized in this section.

Finally, I list useful extensions that we can continue to work on in the SSW and PHV projects. I also state my opinion on the future of algorithm development for NGS data analysis.

4.1.1 Our contribution to the genomic variation detection studies

To get clear of the DNA sequences is critical to a deep understanding of the genetic insight of any biological system. As NGS techniques have become cheaper and faster for generating DNA sequences over time, many breakthrough studies have been achieved in various areas of biology. At the same time, the analysis and understanding of the great amount of data that are produced by NGS is a difficult task and remains a great challenge.[93] Among the NGS analysis tasks, detection of genomic variation is one of the major problems facing bioinformaticians. In this field, many efforts have been made, and great improvements have been achieved in the past few years. All the tools that contribute to the detection of genomic variation (including mapping tools, assemblers, variation calling tools for detecting different types of variations, quality control tools, etc.) are tested and selected by applying them to data generated by various sequencing platforms and for different research scenarios. Based on peoples' experiences, pipelines for sequencing data analysis are built by selecting different tools to achieve the best detection of variation on different data sets and for different purposes.

Depending on what kind of variation we are looking for, the pipelines and methods for variation detection are different. For example, copy number variation can be detected by evaluating the read depth at different genomic regions, and detection of mobile element insertion can be realized by aligning paired reads to the standard genome reference and mobile element references. For most genomic variation detection (no matter what kind of variation), the pipelines for data processing will use one of two approaches: variation detection based on alignment, and variation detection based on assembly. Due to the short read length produced by current NGS techniques, hard to resolve repeat regions remain a big problem for interpreting the assembled graph. As a result, most variation detection relies on the alignment to a reference genome, although the incorporation of local or short-range assembly can largely improve alignment-based variation detection.

As the most accurate approximate string-matching algorithm, the SW algorithm and its variations (the Needleman-Wunsch algorithm, the prefix-suffix alignment algorithm, ect.) are the best choices for the alignment step of many tools in genomic variation detection pipelines, such as mapping tools, assembly tools, structure variation detection tools, and short SNP and INDEL calling tools. The SSW library described in this thesis is the first speed and memory efficient SW library. Besides its elaborately designed features for usage (lightweight, user friendly API for multiple programming languages, multithread support, sub-optimal alignment score availability, etc.), SSW strongly supports the development of efficient new alignment-embedded algorithms. At another point in the genomic variation detection pipelines, PHV opens a new direction for the development of short to medium-sized variation detection methods. Although HMM has been used in several variation detection tools, direct variation breakpoint calling from a Baum-Welch algorithm-trained PHMM has never been seen. This straightforward and most statistically comprehensive interpretation of sequencing read information is a more reasonable design compared to most existing short SNP and INDEL calling methods. With its potential for handling low-quality reads, PHV may become a very suitable tool for the analysis of single-molecule sequencing reads in the near future.

4.1.2 The novel dynamic programming algorithms

Here I summarize the novel algorithms that I developed to support the functions of my software.

4.1.2.1 The SIMD SW library that return full SW results and estimate the sub-optimal alignment score and location on the reference sequence

Most existing SIMD SW implementations only calculate an optimal SW score. One of the main reasons that they do not return other SW results (including detailed alignments) is the special difficulty caused by the vectorization and rearrangement during the calculation of the SW matrix. For example, the vectorisation approaches of some classical SIMD implementations are shown in Figure 4.1. When the values in the SW matrix are calculated by vectors, and changed by later adjustments, the dependences of the cells in the original SW algorithm become very difficult to be traced and recorded. The original SW backtrack algorithm cannot be applied to the SIMD SW at all, and adapting it would cost a lot (in terms of running time and memory usage). Therefore, we avoided trying to develop a backtrack algorithm for SIMD SW. Instead, we solved this problem in a totally different way: we only record the optimal alignment location on the reference and read sequences during the SW matrix calculation, and from that position apply an SIMD SW reversely to locate the optimal alignment beginning position; lastly, we use a banded SW to find the alignment path (generating a detailed alignment). Because the reference sequence is usually much longer than the read sequence for genomic alignments, the reverse SIMD SW matrix is usually much smaller than the entire SW matrix. When both the beginning and ending locations of the optimal alignment are known, the band of the last banded SW is very narrow in most cases. As a result, in most cases SSW's running time for returning a detailed alignment is almost the same as for returning only the optimal score for genomic sequence alignments.

4.1.2.2 The banded PHMM-based short-to-medium-sized variation caller

In the PHV project, the novel design of the algorithm and the unique way of solving the variation detection problem are much more straighforward and inspirational. Since the input of PHV is a BAM file, it is easy to mistake its variation detection as realignment-based, which is what other popular variation callers do. However, there are two fundamental differences be-

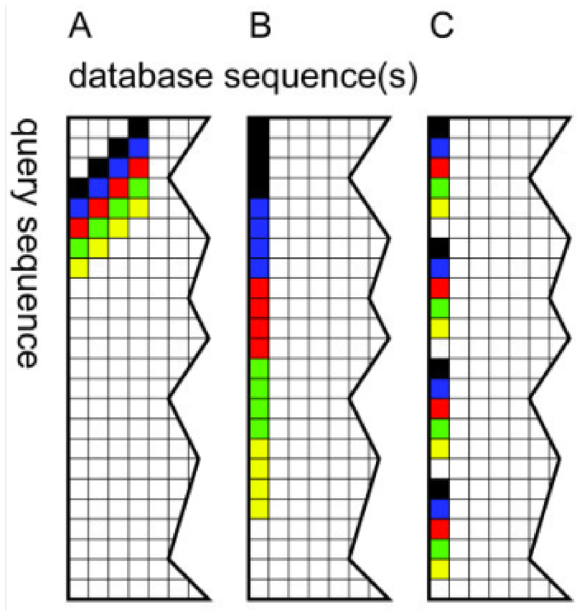


Figure 4.1: The vectorization methods of inter-sequence SIMD SW algorithms. (This figure is revised from Figure 1 of Rognes[98]) The figure shows SW matrices with the first five query vectors marked by colors. For simplicity, vectors only containing four elements (panel A and B) or six elements (panel C) are shown here, while in most real implementations vectors containing 16 elements are used. (A) Vectors are along the anti-diagonal, as used by Wozniak et al.[116] (B) Vectors are along the query, as used by Rognes and Seeberg.[99] (C) Elements of each vector are taken from each color group of panel B in order. This is used by Farrar[27] and SSW[120].

tween PHV and other variation callers: 1) other callers' realignment step is triggered when they sense a misalignment may happen, while PHV applies Baum-Welch training at all sites where a variation may occur. PHV's variation calling never relies on the given alignment results at any place of the genome. 2) Other variation callers (including most HMM-based variation callers) call variations based on one alignment result for each read (either the input alignment or the adjusted alignment), while PHV calls variation based on a statistical summary of all possible alignments of each read. This allows PHV to interpret the data more systematically and objectively.

Moreover, PHV's HMM profile design is totally different from that of most other HMM-based variation callers (such as Dindel[4] and VARiD[20]). PHV's HMM profile is close to the basic PHMM described in the book written by R. Durbin *et al.*[24], but it does not allow a direct connection between insertion states and deletion states (such cases should be described as SNP or MNP). Within the sliding window, PHV's HMM profile represents a semi-global alignment between the reference and the read sequences. This profile design is the most natural and flexible representation of the sequence relationships, so it has the powerful potential to detect any kind of variation including SNPs, MNPs, INDELs, inversions, tandem repeats, etc. A very similar profile design is also used by the base alignment quality (BAQ) estimation[61]. Compared to the HMM profile for BAQ calculation, PHV's profile has an additional insertion state before the first match state to allow the alignment between the read tail and the beginning of the profile. Although this irregular profile shape complicated the HMM algorithms, it represents the read-to-profile relationship more comprehensively than BAQ. PyroHMMvar is also based on a PHMM similar to the basic PHMM, but it uses the nodes of dimers (nucleotide type, repeat count).[118] This profile design targets the homopolymer problem but it loses the flexible and natural representation of the sequence relationships. As a result, PyroHMMvar has to add complicated steps to restore the sequence relationship before variation calling: building the weighted alignment graph and aligning the genotype to the reference. Also, during this process, only the best alignment of each read (generated by the Viterbi algorithm) is used for variation calling. Contrarily, PHV's variation calling is much more straightforward, fast and objective.

From a technical point of view, PHV is the first HMM-based variation caller that uses banded HMM algorithms: a banded forward-backward algorithm and a banded Viterbi algorithm. Due to the usage of the banded forward-backward algorithm in the Baum-Welch training, the difficulty of the algorithm implementation is also increased: the read contribution to the profile is only added within the band of the read location. The banded approach is very important for PHV to achieve its fast speed and low memory usage. The complexities of both properties are changed from O(rl) to O(bl), where l is the length of the reference sequence, r is the read length, and b is the band width. This change becomes more important when dealing with long -read data.

During PHV's variation detection, the HMM profile training process is also a machine-learning process. Compared to other HMM or machine learning-based variation callers such as *SVM²*[15] and SNVHMM[8], PHV does not need pre-training using extra data. This not only allows for the simple usage of the software, but it also results in a theoretical advantage for data interpretation. By training the profile using the data that we are calling variation from, PHV avoids the bias brought in by an unrelated training data set.

4.1.3 Important algorithms and mathematical models used in the main projects

The main algorithm and mathematical models we used in the SSW and PHV projects include: the SW algorithm, HMM and the Bayesian model.

4.1.3.1 Dynamic programming algorithms for approximate string matching

Dynamic programming algorithms are still the most accurate string-matching methods that allow mismatches and INDELs,but at the same time they have high space and time complexities; also, due to "dynamic programming" (each value's calculation depends on the previous ones), it is particularly difficult to reduce the running time of these algorithms. The accuracy and cost issues greatly inconvenience users who rely on an approximate string-matching module in their programs, but this shortcoming inspires improvements of the algorithms, specifically SW. The SW algorithm can be simply described as follows:

Step 1. SW matrix calculation using the following equations:

$$H_{0,0} = E_{1,1} = F_{1,1} = 0,$$

$$H_{i,j} = max \begin{cases} 0 \\ E_{i,j} \\ F_{i,j} \\ H_{i-1,j-1} - W(d_i, q_j) \end{cases},$$

$$E_{i+1,j} = max \begin{cases} 0 \\ E_{i,j} - G_{ext} \\ H_{i,j} - G_{init} \end{cases},$$

$$F_{i,j+1} = max \begin{cases} 0 \\ F_{i,j} - G_{ext} \\ H_{i,j} - G_{init} \end{cases},$$

where i denotes the reference sequence d's coordinates, j denotes the read sequence q's coordinates, H denotes the substitution scores, E denotes the deletion scores, F denotes the insertion scores, W denotes the substitution penalty, G_{init} denotes the gap opening penalty, and G_{ext} denotes the gap extension penalty.

During the SW matrix calculation, the value from which the current value is calculated needs to be recorded.

Step 2. Backtrace:

Locate the largest score in the SW matrix, and backtrace from which values this largest score is calculated. This path marks the detailed alignment.

The SW algorithm is a semi-alignment algorithm (only the best parts of the two strings are aligned), and it is the most widely used approximate dynamic-programming string-matching algorithm. Actually, variations of SW are also very useful in various cases, such as the Needleman-Wunsch algorithm (global alignment), the semi-global dynamic programming alignment algorithm, and the prefix-suffix alignment algorithm. These algorithms are different from SW at the following points: 1) how to initialize the H, E and F values; 2) whether to allow negative values during matrix calculation; 3) from where in the matrix to do the backtrace, and at which point the backtrace stops. Although these differences seem trivial, they can cause many difficulties in the vectorized implementations. The main reasons are: 1) the vectorized SW implementations are so different from the original algorithm that almost no line of code shows similarity to the original SW implementation; technically, they are different algorithms in terms of matrix calculation and backtrace, and consequently, large changes in the computer programs may be needed to implement the stated algorithm differences; 2) the limitation of SSE2 intrinsic codes; the SSE2 intrinsic code set only contains a small amount of functions, so that some data processing is difficult to be realized for them; 3) the requirement for highly efficient code; although the functions of the variant dynamic programming algorithms can be implemented by some revision of the code, it is very difficult to maintain the speed, since the structure of the SIMD SW algorithm is very tight, and many details of the vectorized SW implementations are specifically designed for efficiency. Any change to the SIMD SW algorithm may considerably increase its running time.

4.1.3.2 The hidden Markov model

HMM is a probabilistic model that is generally applicable to the modeling of time series and linear sequences.[25] Depending on the problems that we are aiming to solve, HMM can have very flexible designs. Basically, HMM assumes the observed data (typically linear sequences) are emitted from some hidden states that have different probabilities to emit different residues in the observed sequences. The change of the sequence contents is caused by the transitions between the hidden states. By using HMM, we are trying to estimate the probability properties of the underlying hidden states from the observed data, and to find out how the observed data is generated from these states. HMM has been most widely used for word recognition in human speech[97], and it was first used to solve biological problems in 1989[17]. Profile HMM was first used in biology for multiple sequence alignment in 1994[53]. The PHMM described in this thesis is still very similar to the profile structure of the original PHMM by Krogh *et al.* PHMM also has interesting parallels with dynamic programming algorithms for approximate string matching. For example, the Viterbi algorithm of PHMM is functionally the same as the SW algorithm; both of them do the pairwise alignment. The differences are: 1) the SW algorithm aligns two sequences, while Viterbi of the PHMM aligns one sequence to a profile; 2) SW evaluates the penalties for sequence differences, while Viterbi of the PHMM evaluates the probability of emitting the sequence from different hidden states of the profile. Even with these differences, from the mathematical formulas of the algorithms we can still tell SW and Viterbi are intrinsically the same: they use similar methods to solve similar problems. In summary, the HMM containing several simple but elegant algorithms is a versatile and powerful tool that can be used to solve many kinds of problems. The introduction of HMM into biology largely improved both the mathematical modeling of biological data and the computational solutions of the biological questions such as gene and motif finding, protein structure prediction, and biological sequence alignment. We appreciate the creative applications of HMM in the field of biology achieved by previous work, and we try to provide new ways to use HMM in this thesis.

4.1.3.3 The Bayesian model

Another simple but powerful mathematical model mentioned in this thesis is the Bayesian model. The Bayesian model is an inverse probability model that allows us to make predictions from the observed data and from previous knowledge. What the Bayesian model does is clearly stated in the following Bayes rule:

$$P(hypothesis|data) = \frac{P(data|hypothesis) \cdot P(hypothesis)}{P(data)}$$

A more detailed description of the Bayesian model for genotype calling was described in the Introduction part of this thesis, so we will not repeat it here. Similar to HMM, the Bayesian model is also very versatile and can be applied to many biological problems. As mentioned in this thesis, the Bayesian model is the key component of the mainstream detection methods of genomic variation. We can also incorporate this model into our PHV project to improve the accuracies of variation calling and quality assignment.

4.1.4 High performance computing

The SSW and PHV projects both highly rely on high-performance computing due to the high time and space complexities of the dynamic programming algorithms. The high-performance computing strategies used in my work can be summarized as: 1) using banded dynamic programming to reduce the amount of calculation; 2) using the SSE2 intrinsic codes to fully utilize the CPU power (low-level parallelization); and 3) using certain coding tricks to make the implementations efficient. Since the banded approach has been described extensively in other parts of the thesis, here we only explain the later two strategies.

Most current computers (powered by 64-bit CPUs with multimedia extensions) support the SIMD operations. SIMD operations are suitable for improving the efficiency of simple, repetitive calculations of large amounts of data. The SIMD operations are realized through SIMD instructions including MMX, SSE, SSE2 intrinsics, etc. In this thesis, we used SSE2 intrinsic codes to speed up the SW algorithm, since SSE2 is the most powerful SIMD instruction that is supported by most current CPUs. Current CPU registers are 128 bit (16 byte), which can hold multiple data, for example 16 8-bit integers (data type _m128i) or four single-precision floating-point data (data type_m128d). Consequently, by using the SSE2 intrinsic codes we can calculate multiple data at the same time using one instruction. A simple example to explain the SIMD operation is shown in Figure 4.2. The SSE2 instructions and _m128 data types used in the SSW project are in the header file emmintrin.h. By including this header, the SSE2 intrinsic codes can be directly embedded in the C/C++ programs. To do one calculation with vectors con-

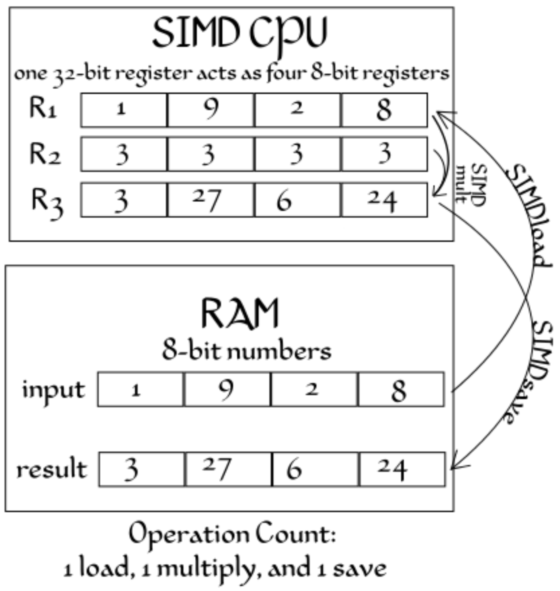


Figure 4.2: The SIMD tripling of a vector that contains four numbers (The figure is from http://en.wikipedia.org/wiki/SIMD). The upper part of this figure shows three registers that hold the data for this calculation. To keep it simple, the registers in this figure are 32 bit, while most current processors have 128-bit registers. The processor loads four numbers from the memory at once, multiplies them together with one SIMD instruction, and saves the results at once back to the memory.

taining 16 integers, we need to allocate two pieces of _m128i type memory, assign 16 integers to each piece of the memory, load the memory contents to two CPU registers, do the vector calculation, and save the vector result back to one piece of the _m128i type memory. Each step corresponds to one SSE2 intrinsic code line.

Besides the main speedup strategies (the banded approach and the usage of SIMD instructions), many other details during programming are designed for the consideration of efficiency. For example: I use the macros of bit calculation to replace ordinary calculation; I use direct mapping by arrays to replace the decision control statements (such as "if... else", switch); I use the memory size of an exponential number of base 2 (such as 64, 128, 256...) when designing data structures, so that the computer can handle these memory faster; I use the "LIKELY", "UNLIKELY" keywords to lead the CPU behavior; I use less branches (such as "if... else" statement) in the inner loop; etc. Of course, the overall software design is more important for achieving efficiency and for keeping the flexibility for further extensions. Due to the technical difficulties I encountered during software development (for SSW we needed to pursue ultimate efficiency, while for PHV we needed to achieve better results and handle an efficient, large software design), I have begun to appreciate the art of programming.

4.1.5 Software availability

Both SSW and PHV are open source tools under MIT licenses. The source code, testing data set, software development records, issue discussions and my contact information for software maintenance are available at GitHub:

SSW: https://github.com/mengyao/Complete-Striped-Smith-Waterman-Library

PHV: https://github.com/mengyao/PHMM-SNP-INDEL-caller

The user manuals of SSW and PHV are in Appendix C and D, respectively.

4.2 Future Directions

4.2.1 Very useful extensions of SSW and PHV projects

Since the publication of the SSW Library, we have received a lot of attention and support from users. People use SSW either as a convenient alignment tool or a component in their own software. Since the usage of SSW is versatile, we were contacted by many users to talk about their various needs. These needs are also the expected expansions that we would like to continue to work on for the SSW project. We see some users would like us to add Needleman-Wunsch, semi-global alignment, and split-alignment functions. Some people would like to access more output information and do longer sequence alignments. In my opinion, the most useful extension is to realize the Needleman-Wunsch, semi-global alignment, and other SW variations for different approximate string-matching purposes. This improvement will extend SSW to a very useful SIMD approximate string-matching package, so that biological research can really benefit from the SIMD theory. As we explained previously, the variations of SIMD SW are not easily achieved by revising SSW, so providing a multi-functional library is a non-trivial task. We also would like to allow SSW to give more output information such as the MD tag of the SAM format to facilitate variation detection without a reference sequence. We are also looking forward to the wide availability of CPUs that support SSE4 intrinsics. With the limitations imposed by the instruction set to become fewer, SSW will easily be able to handle reads twice as long as current ones.

For the PHV project, the most useful extension is realizing variation calling from multiple individual samples. To do this, we can train one HMM profile for each individual and statistically combine the variation information from each profile before the variation calling. By this design, the individual tags of the reads are used, and the variation signal from different individuals can help to deal with the issue of weak signal in low-coverage data. Another useful improvement is to wrap the banded PHMM into a user-friendly library. Although there are already PHMM libraries available, a banded version is still lacking. Considering the high time and space complexities of PHMM algorithms, a banded (or even SIMD) PHMM library will be very useful for the related software development.

4.2.2 Looking forward to the time of long read

The development of NGS data analysis methods follows the development of NGS technique and the changes of the data. Currently, many efforts are made to generate longer reads that can carry more genomic information, to reduce the analysis difficulties and to generate more reliable post-processing results such as genomic variation. In the not too distant future, we expect to enter the era of long reads. By that time, assembly-based methods will become much more widely used as the most suitable approach for the detection of genomic variation for and haplotype generation. At the same time, read alignment will still be important for handling low-coverage data. Although the methods for short read alignments are already well developed, there is still a lot of space for improvement in terms of long read alignments (especially low-quality long reads). Also, methods of aligning reads to the genomic sequences of multiple individuals will be needed, and any such methods will require our contribution to their continuous improvement. As the development of the NGS analysis methods progresses, I believe that the content of this thesis will inspire our further efforts to accomodate both current biological research and the long-read data of the future.

A

The genomes and reads used for generating Figure 2.3

SARS: The genome and reads were downloaded from the web site http://www.bcgsc.ca/project/sars/SARS/. The SARS reference genome file is TOR2_finished_genome_assembly_290403(Release 3). The 1000 reads were randomly chosen from the file TOR2_genome_shotgun_120403.fasta.

E. coli: The genome is from *E. coli* strain 536, downloaded from http://www.ncbi.nlm.nih.gov/nucleotide/NC_008253. The query sequences are Ion Torrent sequenced *E. coli* strain DH10B (C23-140, 318 PGM Run, 11/2011). Read length: 25-540 bp. Most reads are 200 bp.

T. gondii: The genome was downloaded from http://toxodb.org/common/downloads/release-7.0/TgondiiGT1/fasta/. The sequences of 14 *T. gondii* chromosomes were extracted from the file TgondiiGT1Genomic_ToxoDB-7.0.fasta and were used as the reference genome. The Illumina sequenced transcriptome of bone marrow-derived macrophages infected with *T. gondii* GT1 strain were downloaded from the NCBI Short Read Archive (SRR1002971) and the first 1000 reads in this file were used as the query sequences.

Chr1: The genome is human reference genome GRCh37. The read file was downloaded from the short read archive (SRR835935). The first 1000 reads were used as the queries.

B

The sample identifiers of AFR in the 1000 Genomes Project

HG01879, HG01880, HG01886, HG01896, HG01914, HG01915, HG01958, HG01985, HG01986, HG02013, HG02014, HG02051, HG02449, HG02470, HG02471, HG02489, NA18486, NA18487, NA18488, NA18507, NA18516, NA18520, NA18523, NA18867, NA18868, NA18873, NA18874, NA18908, NA18910, NA18917, NA18923, NA18924, NA18933, NA18934, NA19092, NA19119, NA19130, NA19131, NA19141, NA19143, NA19144, NA19152, NA19153, NA19159, NA19160, NA19171, NA19172, NA19189, NA19197, NA19198, NA19200, NA19204, NA19213, NA19223, NA19235, NA19236, NA19247, NA19248, NA19311, NA19312, NA19313, NA19315, NA19316, NA19317, NA19318, NA19319, NA19321, NA19324, NA19327, NA19328, NA19331, NA19332, NA19334, NA19338, NA19346, NA19347, NA19350, NA19355, NA19359, NA19360, NA19371, NA19372, NA19373, NA19374, NA19375, NA19376, NA19377, NA19379, NA19380, NA19381, NA19382, NA19383, NA19384, NA19385, NA19390, NA19391, NA19393, NA19394, NA19395, NA19396, NA19397, NA19398, NA19399, NA19401, NA19403, NA19404, NA19428, NA19429, NA19430, NA19431, NA19434, NA19435, NA19436, NA19437, NA19438, NA19439, NA19440, NA19443, NA19444, NA19445, NA19446, NA19448, NA19449, NA19451, NA19452, NA19453, NA19455, NA19456, NA19457, NA19461, NA19462, NA19463, NA19466, NA19467, NA19468, NA19469, NA19470, NA19471, NA19472, NA19473, NA19474, NA19625, NA19700, NA19701, NA19703, NA19704, NA19707, NA19711, NA19712, NA19713, NA19818, NA19819, NA19834, NA19835, NA19900, NA19901, NA19904, NA19908, NA19909, NA19914, NA19916, NA19917, NA19920, NA19921, NA19982, NA19985, NA20126, NA20127, NA20276, NA20278, NA20281, NA20282, NA20287, NA20289, NA20291, NA20294, NA20296, NA20299, NA20314, NA20317, NA20322, NA20332, NA20334, NA20336, NA20340, NA20341, NA20342, NA20344, NA20346, NA20348, NA20356

C

The user manual of the SSW Library

C.1 Overview

SSW is a fast implementation of the Smith-Waterman algorithm, which uses the Single-Instruction Multiple-Data (SIMD) instructions to parallelize the algorithm at the instruction level. SSW library provides an API that can be flexibly used by programs written in C, C++ and other languages. We also provide a software that can do protein and genome alignment directly. Current version of our implementation is 50 times faster than an ordinary Smith-Waterman. It can return the Smith-Waterman score, alignment location and traceback path (cigar) of the optimal alignment accurately; and return the sub-optimal alignment score and location heuristically.

Note: When SSW open a gap, the gap open penalty alone is applied.

C.2 C/C++ interface

C.2.1 How to use the API

The API files include ssw.h and ssw.c, which can be directly used by any C or C++ program. For the C++ users who are more comfortable to use a C++ style interface, an additional C++ wrapper is provided with the file ssw_cpp.cpp and ssw_cpp.h.

To use the C style API, please:

1. Download ssw.h and ssw.c, and put them in the same folder of your own program files.
2. Write #include "ssw.h" into your file that will call the API functions.
3. The API files are ready to be compiled together with your own C/C++ files.

The API function descriptions are in the file ssw.h. One simple example of the API usage is example.c. The Smith-Waterman penalties need to be integers. Small penalty numbers such as: match: 2, mismatch: -1, gap open: -3, gap extension: -1 are recommended, which will lead to shorter running time.

To use the C++ style API, please:

1. Download ssw.h, ssw.c, ssw_cpp.cpp and ssw_cpp.h and put them in the same folder of your own program files.
2. Write #include "ssw_cpp.h" into your file that will call the API functions.
3. The API files are ready to be compiled together with your own C/C++ files.

The API function descriptions are in the file ssw_cpp.h. A simple example of using the C++ API is example.cpp.

C.2.2 Speed and memory usage of the API

Test data set: Target sequence: reference genome of E. coli strain 536 (4,938,920 nucleotides) from NCBI Query sequences: 1000 reads of Ion Torrent sequenced E. coli strain DH10B (C23-140, 318 PGM Run, 11/2011), read length: 25-540 bp, most reads are 200 bp

CPU time:

AMD CPU: default penalties: 880 seconds; -m1 -x3 -o5 -e2: 460 seconds

Intel CPU: default penalties: 960 seconds; -m1 -x3 -o5 -e2: 500 seconds

Memory usage: 40MB

C.2.3 Install the software

1. Download the software from https://github.com/mengyao/Complete-Striped-Smith-Waterman-Library.

2. cd src

3. make

4. The executable file will be ssw_test.

C.2.4 Run the software

```
Usage: ssw_test [options] ... <target.fasta> <query.fasta>(or <query.fastq>)
Options:
    -m N    N is a positive integer for weight match in genome sequence
alignment. [default: 2]
    -x N    N is a positive integer. -N will be used as weight mismatch
in genome sequence alignment. [default: 2]
    -o N    N is a positive integer. -N will be used as the weight for the
gap opening. [default: 3]
    -e N    N is a positive integer. -N will be used as the weight for the
gap extension. [default: 1]
    -p   Do protein sequence alignment. Without this option, the ssw_test
will do genome sequence alignment.
    -a FILE FILE is either the Blosum or Pam weight matrix. [default: Blosum50]
    -c   Return the alignment path.
    -f N    N is a positive integer. Only output the alignments with the
Smith-Waterman score >= N.
    -r   The best alignment will be picked between the original read
alignment and the reverse complement read alignment.
    -s   Output in SAM format. [default: no header]
    -h   If -s is used, include header in SAM output.
```

C.2.5 Software output

The software can output SAM format or BLAST like format results.

C.2.5.1 An example of the SAM format output:

```
@HD VN:1.4  SO:queryname
@SQ SN:chr1 LN:1001
1   0   chr1    453 5   3M2D3M1D4M2D6M1D5M1D5M2I7M  *   0   0
CCAGCCCAAAATCTGTTTTAATGGTGGATTTGTGT *   AS:i:37 NM:i:11 ZS:i:28
2    0   chr1    523 4   2M1D32M1D3M1D6M1D8M *   0   0
GAAGAGTTAATTTAAGTCACTTCAAACAGATTACGTATCTTTTTTTTCCCT *   AS:i:42 NM:i:16 ZS:i:41
3   0   chr1    120 4   2M1I4M3D3M1I7M2I9M2D6M1I8M  *   0   0
AACAACAGAAGTTAATTAGCTTCAAAAATACTTTATATTTGCAA    *   AS:i:32 NM:i:16 ZS:i:29
4  0   chr1    302 4   8M1D8M1D3M2D6M1D4M2I2M1D2M3D5M1I4M  *   0   0
CATTTATTGTTGTTTTTAAAGATTAAATGATTAAATGTTTCAAAA   *   AS:i:32 NM:i:18 ZS:i:30
5  0   chr1    4   5   4M2D4M1D9M1I3M4I16M1I3M1D4M2D5M *   0   0
ACAGTGATGCCAAGCCAGTGGGTTTTAGCTTGTGGAGTTCCATAGGAGCGATGC  *   AS:i:30 NM:i:22 ZS:i:23
6   0   chr1    142 4   4M3I5M4D10M2D4M1I2M2I6M5D1M1D6M2D3M *   0   0
AATAACCATAAAAATGGGCAAAGCAGCCTTCAGGGCTGCTGTTTCTA *   AS:i:26 NM:i:25 ZS:i:26
...
```

Please check the document "The SAM Format Specification" at: http://samtools.sourceforge.net/SAM1.pdf for the full description for detailed statement of the SAM format. The additional optional field "ZS" indicates the suboptimal alignment score. For example, the 1st record in the upper example means the optimal alignment score of the given sequence is 37; the suboptimal alignment score is 28; the mismatch and INDEL base count within the aligned fragment of the read is 11.

C.2.5.2 An example of the BLAST like output:

```
target_name: chr1
query_name: 6:163296599:F:198;None;None/1
optimal_alignment_score: 37 sub-optimal_alignment_score: 28 strand: +
target_begin: 453   target_end: 492 query_begin: 17 query_end: 51

Target:      453    CCAATGCCACAAAACATCTGTCTCTAACTGGTG--TGTGTGT    492
                    |||  ||| ||||  |||||| | ||| |||||  |*|||||
Query:        17    CCA--GCC-CAAA--ATCTGT-TTTAA-TGGTGGATTTGTGT    51

target_name: chr1
query_name: 3:153409880:F:224;None;3,153410143,G,A/1
optimal_alignment_score: 42 sub-optimal_alignment_score: 41 strand: +
target_begin: 523   target_end: 577 query_begin: 3  query_end: 53
```

```
Target:      523    GAGAGAGAAAATTTCACTCCCTCCATAAATCTCACAGTATTCTTTTCTTTTTCCT    577
                    || ||||**|||||*|*||*||*||*|*|**|*|| ||| |||||| ||||*|||
Query:         3    GA-AGAGTTAATTTAAGTCACTTCAAACAGATTAC-GTA-TCTTTT-TTTTCCCT    53
...
```

C.3 Python interface

C.3.1 How to use the python wrapper ssw_wrap.py

A Python wrapper partially implements the c library functionality (only DNA sequences can be aligned for now). c libraries are completely integrated in a simple module that do not require any C programming knowledge to be used. Briefly, An aligner object can be initialized with alignment parameters and a reference subject sequence, then the object method align can be called with a query sequence and filtering conditions (min score and min length) as many time as desired. Depending of the score and length requested an python object PyAlignRes will be eventually returned.

To use the python wrapper, please:

1. Compile the scr folder by either using the makefile or by compiling a dynamic shared library with gcc:
   gcc -Wall -O3 -pipe -fPIC -shared -rdynamic -o libssw.so ssw.c ssw.h
2. libssw.so and ssw_wrap.py can them be put in the same folder of your own program files.
3. Depending of the LINUX OS version installed it may be required to modify the LD_LIBRARY_PATH environment variable to use the dynamic library libssw.so by one of the 2 following possibilities:
   Export the path or the directory containing the library LD_LIBRARY_PATH=path_of_the_library
   For a definitive inclusion edit /etc/ld.so.conf and add the path of the lib directory. Then, update the cache by using /sbin/ldconfig as root.
4. In a python script or in a interactive interpreter the main class can be imported with:
   from ssw_wrap import Aligner
5. Instantiate the Aligner class with initial parameters, including the reference subject

sequence:

Aligner(myref, match=2, mismatch=2, gap_open=3, gap_extension=1, report_secondary=False, report_cigar=False)

6. Call the object align method with a query sequence, the minimal score and length for the alignment to be reported:

   res = ssw.align(myquery, min_score=10, min_len=20)

7. Parse the returned PyAlignRes object for alignment result description.

C.3.2 Run pyssw standalone

```
Usage: pyssw.py -s subject.fasta -q fastq (or fasta) [Facultative options]

Options:
 --version             show program’s version number and exit
 -h, --help            show this help message and exit
 -s SUBJECT, --subject=SUBJECT Path of the fasta file containing the subject
genome sequence [REQUIRED]
 -q QUERY, --query=QUERY   Path of the fastq or fasta file containing the short
read to be aligned [REQUIRED]
 -t QTYPE, --qtype=QTYPE   Type of the query file = fastq or fasta. [default:fastq]
 -m MATCH, --match=MATCH   positive integer for weight match in genome sequence
alignment. [default: 2]
 -x MISMATCH, --mismatch=MISMATCH  positive integer. The negative value will
be used as weight mismatch in genome sequence alignment.[default: 2]
 -o GAP_OPEN, --gap_open=GAP_OPEN  positive integer. The negative value will
be used as weight for the gap opening. [default: 3]
 -e GAP_EXTEND, --gap_extend=GAP_EXTEND    positive integer. The negative value
will be used as weight for the gap opening. [default: 1]
 -f MIN_SCORE, --min_score=MIN_SCORE   integer. Consider alignments having a score
<= MIN_SCORE as not aligned. [default: 0]
 -l MIN_LEN, --min_len=MIN_LEN integer. Consider alignments having a length <= as
not aligned. [default: 0]
 -r, --reverse         Flag. Align query in forward and reverse orientation and
choose the best alignment. [Set by default]
 -u, --unaligned       Flag. Write unaligned reads in sam output [Unset by default]
```

C.3.3 pyssw output

For now, the program only output a SAM like file. The file encoding is sightly more complete than the version created by the C/C++ software and is fully compatible with downstream NGS utilities such as samtools. The MAPQ score field is set to 0, but the ssw score is reported in the AS optional tag.

```
@HD VN:1.0  SO:unsorted
@SQ SN:Virus_genome LN:2216
@PG ID:Striped-Smith-Waterman   PN:pyssw    VN:0.1
@CO Score_values = match 2, mismatch 2, gap_open 3, gap_extend 1
@CO Filter Options = min_score 250, min_len 0
1  16  Virus_genome    1773    0   151M    *   0   0    TTGTTT...TATTAC    AS:i:302
2  0   Virus_genome    1985    0   21M1I129M   *   0   0    TTTAAA...CTCGCT    AS:i:289
3  16  Virus_genome    716 0   36M2D115M   *   0   0   CTTACC...GGAATT    AS:i:294
...
```

C.3.4 Speed and memory usage of pyssw

Intel Core i5 3320PM CPU @ 2.60GHz x 2

1. E. coli K12 U0096.3 (4,641,652 nucleotides) vs 10,000 illumina reads (50pb) from CLoVR public repository

   pyssw.py -s Ecoli_K12_U0096.3.fa -q Ecoli_illumina_10k.fastq -r -f 80

   Time : 55m 40s

2. Short virus reference (2,216 nt) vs 100,000 illumina Miseq reads (150pb) (see demo dataset)

   pyssw.py -s Virus_genome.fa.gz -q 100k_illumina1.fastq.gz -f 250

   Time : 60s

C.4 Please cite this paper, if you need:

http://dx.plos.org/10.1371/journal.pone.0082138

D
The user manual of PHV

D.1 Overview

PHV is an accurate SNP and INDEL caller based on the Profile Hidden Markov Model. Different from most other variant callers, PHV does not rely on the single alignment generated by the aligners. Instead, it considers all possible alignments of the reads together by learning from the given data. Therefore, PHV can give more accurate variant calls. On our testing data set, PHV's SNP detection is already better than GATK and Samtools. PHV takes reference file (in FASTA format) and alignemnt file (in BAM format) as inputs, and outputs the genomic variations (in VCF format).

D.2 Install PHV

1. Download the software from https://github.com/mengyao/PHMM-SNP-INDEL-caller.

2. cd src

3. make

4. The executable file will be region.

D.3 Run PHV

```
Usage:   region [options] <reference.fasta> <alignment.bam> [region1 [...]]

Options:
-s N N is the largest detectable INDEL length. [default: 100]
Notes:

     A region should be presented in one of the following formats:
     ‘chr1’, ‘chr2:1,000’ and ‘chr3:1,000-2,000’. When a region is
     specified, the input alignment file must be an indexed BAM file.
```

D.4 Output of PHV

The output of PHV is the genomic variation in VCF format. Please see the description of VCF format at http://www.1000genomes.org/wiki/Analysis/Variant%20Call%20Format/vcf-variant-call-format-version-41.

References

[1] 1000 Genomes Project Consortium, Abecasis, G. R., Altshuler, D., Auton, A., Brooks, L. D., Durbin, R. M., Gibbs, R. A., Hurles, M. E., & McVean, G. A. (2010). A map of human genome variation from population-scale sequencing. *Nature*, 467(7319), 1061–73.

[2] 1000 Genomes Project Consortium, Abecasis, G. R., Auton, A., Brooks, L. D., DePristo, M. A., Durbin, R. M., Handsaker, R. E., Kang, H. M., Marth, G. T., & McVean, G. A. (2012). An integrated map of genetic variation from 1,092 human genomes. *Nature*, 491(7422), 56–65.

[3] Abouelhoda, M. I., Kurtz, S., & Ohlebusch, E. (2004). Replacing suffix trees with enhanced suffix arrays. *J Discrete Algorithms*, 2, 53–86.

[4] Albers, C. A., Lunter, G., MacArthur, D. G., McVean, G., Ouwehand, W. H., & Durbin, R. (2011). Dindel: accurate indel calls from short-read data. *Genome Res*, 21(6), 961–73.

[5] Altschul, S. F., Gish, W., Miller, W., Myers, E. W., & Lipman, D. J. (1990). Basic local alignment search tool. *J Mol Biol*, 215(3), 403–10.

[6] Anson, E. L. & Myers, E. W. (1997). Realigner: a program for refining dna sequence multi-alignments. *J Comput Biol*, 4(3), 369–83.

[7] Baer, R., Bankier, A. T., Biggin, M. D., Deininger, P. L., Farrell, P. J., Gibson, T. J., Hatfull, G., Hudson, G. S., Satchwell, S. C., & Séguin, C. (1984). Dna sequence and expression of the b95-8 epstein-barr virus genome. *Nature*, 310(5974), 207–11.

[8] Bian, J., Liu, C., Wang, H., Xing, J., Kachroo, P., & Zhou, X. (2013). Snvhmm: predicting single nucleotide variants from next generation sequencing. *BMC Bioinformatics*, 14, 225.

[9] Bowman, S., Churcher, C., Badcock, K., Brown, D., Chillingworth, T., Connor, R., Dedman, K., Devlin, K., Gentles, S., Hamlin, N., Hunt, S., Jagels, K., Lye, G., Moule, S., Odell, C., Pearson, D., Rajandream, M., Rice, P., Skelton, J., Walsh, S., Whitehead, S., & Barrell, B. (1997). The nucleotide sequence of saccharomyces cerevisiae chromosome xiii. *Nature*, 387(6632 Suppl), 90–3.

[10] Bussey, H., Kaback, D. B., Zhong, W., Vo, D. T., Clark, M. W., Fortin, N., Hall, J., Francis Ouellette, B. F., Keng, T., Barton, A., Su, Y., Davies, C. J., & Storms, R. K. (1995).

The nucleotide sequence of chromosome i from saccharomyces cerevisiae. *Nature*, 92, 3809–13.

[11] Bussey, H., Storms, R. K., Ahmed, A., Albermann, K., Allen, E., Ansorge, W., Araujo, R., Aparicio, A., Barrell, B., Badcock, K., Benes, V., Botstein, D., Bowman, S., Brücknr, M., Carpenter, J., Cherry, J. M., Chung, E., Churcher, C., Coster, F., Davis, K., Davis, R. W., Dietrich, F. S., Delius, H., DiPaolo, T., Dubois, E., Dusterh□, A., Duncan, M., Floeth, M., Fortin, N., Friesen, J. D., Fritz, C., Goffeau, A., Hall, J., Hebling, U., Heumann, K., Hilbert, H., Hillier, L., other members of the Genome Sequencing Center, Hunicke-Smith, S., Hyman, R., Johnston, M., Kalman, S., Kleine, K., Komp, C., Kurdi, O., Lashkari, D., Lew, H., A., L., Lin, D., Louis, E. J., Marathe, R., Messenguy, F., Mewes, H. W., Mirtipati, S., Moesti, D., Muller-Auer, S., Namath, A., Nentwich, U., Oefner, P., Pearson, D., Petel, F. X., Pohl, T. M., Purnelle, B., Rajandream, M. A., Rechmann, S., Rieger, M., Riles, L., Roberts, D., Schafer, M., Scharfe, M., Scherens, B., Schramm, S., Schroder, M., Sdicu, A. M., Tettelin, H., Urrestarazu, L. A., Ushinsky, S., Vierendeels, F., Vissers, S., Voss, H., Walsh, S. V., Wambutt, R., Wang, Y., Wedler, E., Wedler, H., Winnett, E., Zhong, W.-W., Zollner, A., Vo, D. H., & Hani, J. (1997). The nucleotide sequence of saccharomyces cerevisiae chromosome xvi. *Nature*, 387(29), 103–5.

[12] C. elegans Sequencing Consortium (1998). Genome sequence of the nematode c. elegans: a platform for investigating biology. *Science*, 282(5396), 2012–8.

[13] Cao, X., Li, S. C., & Tung, A. K. H. (2005). Eland: Efficient large-scale alignment of nucleotide databases. *Lecture Notes in Computer Science*, 3453, 4–15.

[14] Cartwright, R. A. (2009). Problems and solutions for estimating indel rates and length distributions. *Mol Biol Evol*, 26(2), 473–80.

[15] Chiara, M., Pesole, G., & Horner, D. S. (2012). Svm^2: an improved paired-end-based tool for the detection of small genomic structural variations using high-throughput single-genome resequencing data. *Nucleic Acids Res*, 40(18), e145.

[16] Churcher, C., Bowman, S., Badcock, K., Bankier, A., Brown, D., Chillingworth, T., Connor, R., Devlin, K., Gentles, S., Hamlin, N., Harris, D., Horsnell, T., Hunt, S., Jagels, K., Jones, M., Lye, G., Moule, S., Odell, C., Pearson, D., Rajandream, M., Rice, P., Rowley, N., Skelton, J., Smith, V., & Barrell, B. (1997). The nucleotide sequence of saccharomyces cerevisiae chromosome ix. *Nature*, 387(6632 Suppl), 84–7.

[17] Churchill, G. A. (1989). Stochastic models for heterogeneous dna sequences. *Bull. Math. Biol.*, (51), 79–94.

[18] Cox, A. J. (2007). Eland: Efficient large-scale alignment of nucleotide databases. *Illumina*, San Diego.

[19] Dahm, R. (2008). Discovering dna: Friedrich miescher and the early years of nucleic acid research. *Hum Genet*, 122(6), 565–81.

[20] Dalca, A. V., Rumble, S. M., Levy, S., & Brudno, M. (2010). Varid: a variation detection framework for color-space and letter-space platforms. *Bioinformatics*, 26(12), i343–9.

[21] Dietrich, F. S., Mulligan, J., Hennessy, K., Yelton, M. A., Allen, E., Araujo, R., Aviles, E., Berno, A., Brennan, T., Carpenter, J., Chen, E., Cherry, J. M., Chung, E., Duncan, M., Guzman, E., Hartzell, G., Hunicke-Smith, S., Hyman, R. W., Kayser, A., Komp, C., Lashkari, D., Lew, H., Lin, D., Mosedale, D., & Davis, R. W. (1997). The nucleotide sequence of saccharomyces cerevisiae chromosome v. *Nature*, 387(6632 Suppl), 78–81.

[22] Dujon, B., Albermann, K., Aldea, M., Alexandraki, D., Ansorge, W., Arino, J., Benes, V., Bohn, C., Bolotin-Fukuhara, M., Bordonné, R., Boyer, J., Camasses, A., Casamayor, A., Casas, C., Chéret, G., Cziepluch, C., Daignan-Fornier, B., Dang, D. V., de Haan, M., Delius, H., Durand, P., Fairhead, C., Feldmann, H., Gaillon, L., & Kleine, K. (1997). The nucleotide sequence of saccharomyces cerevisiae chromosome xv. *Nature*, 387(6632 Suppl), 98–102.

[23] Dujon, B., Alexandraki, D., André, B., Ansorge, W., Baladron, V., Ballesta, J. P., Banrevi, A., Bolle, P. A., Bolotin-Fukuhara, M., & Bossier, P. (1994). Complete dna sequence of yeast chromosome xi. *Nature*, 369(6479), 371–8.

[24] Durbin, R. (1998). *Biological sequence analysis: probabalistic models of proteins and nucleic acids*. Cambridge, UK: Cambridge University Press.

[25] Eddy, S. R. (1998). Profile hidden markov models. *Bioinformatics*, 14(9), 755–63.

[26] Ewing, B. & Green, P. (1998). Base-calling of automated sequencer traces using phred. ii. error probabilities. *Genome Res*, 8(3), 186–94.

[27] Farrar, M. (2007). Striped smith-waterman speeds database searches six times over other simd implementations. *Bioinformatics*, 23(2), 156–61.

[28] Feldmann, H., Aigle, M., Aljinovic, G., André, B., Baclet, M. C., Barthe, C., Baur, A., Bécam, A. M., Biteau, N., Boles, E., Brandt, T., Brendel, M., Brückner, M., Bussereau, F., Christiansen, C., Contreras, R., Crouzet, M., Cziepluch, C., Démolis, N., Delaveau, T., Doignon, F., Domdey, H., Düsterhus, S., Dubois, E., Dujon, B., El Bakkoury, M., Entian, K. D., Feurmann, M., Fiers, W., Fobo, G. M., Fritz, C., Gassenhuber, H., Glandsdorff, N., Goffeau, A., Grivell, L. A., de Haan, M., Hein, C., Herbert, C. J., Hollenberg, C. P., Holmstrøm, K., Jacq, C., Jacquet, M., Jauniaux, J. C., Jonniaux, J. L., Kallesøe, T., Kiesau, P., Kirchrath, L., Kötter, P., Korol, S., Liebl, S., Logghe, M., Lohan, A. J., Louis, E. J., Li, Z. Y., Maat, M. J., Mallet, L., Mannhaupt, G., Messenguy, F., Miosga, T., Molemans, F., Müller, S., Nasr, F., Obermaier, B., Perea, J., Piérard, A., Piravandi, E., Pohl, F. M., Pohl, T. M., Potier, S., Proft, M., Purnelle, B., Ramezani Rad, M., Rieger, M., Rose, M., Schaaff-Gerstenschläger, I., Scherens, B., Schwarzlose, C., Skala, J., Slonimski, P. P., Smits, P. H., Souciet, J. L., Steensma, H. Y., Stucka, R., Urrestarazu, A., van der Aart, Q. J., van Dyck, L., Vassarotti, A., Vetter, I., Vierendeels, F., Vissers, S., Wagner, G., de Wergifosse, P., Wolfe, K. H., Zagulski, M., Zimmermann, F. K., Mewes, H. W., & Kleine, K. (1994). Complete dna sequence of yeast chromosome ii. *EMBO J*, 13(24), 5795–809.

[29] Ferragina, P. & Manzini, G. (2000). Opportunistic data structures with applications. *Proceedings of the 41st Symposium on Foundations of Computer Science (FOCS 2000)*, Redondo Beach, CA, USA, 390–8.

[30] Fleischmann, R. D., Adams, M. D., White, O., Clayton, R. A., Kirkness, E. F., Kerlavage, A. R., Bult, C. J., Tomb, J. F., Dougherty, B. A., & Merrick, J. M. (1995). Whole-genome random sequencing and assembly of haemophilus influenzae rd. *Science*, 269(5223), 496–512.

[31] Fred, S., Air, G. M., Barrell, B. G., Brown, N. L., Coulson, A. R., Fiddes, J. C., Hutchison III, C. A., Slocombe, P. M., & Smith, M. (1977). Nucleotide sequence of bacteriophage ϕx174 dna. *Nature*, 265(24), 687–95.

[32] Galibert, F., Alexandraki, D., Baur, A., Boles, E., Chalwatzis, N., Chuat, J. C., Coster, F., Cziepluch, C., De Haan, M., Domdey, H., Durand, P., Entian, K. D., Gatius, M., Goffeau, A., Grivell, L. A., Hennemann, A., Herbert, C. J., Heumann, K., Hilger, F., Hollenberg, C. P., Huang, M. E., Jacq, C., Jauniaux, J. C., Katsoulou, C., & Karpfinger-Hartl, L. (1996). Complete nucleotide sequence of saccharomyces cerevisiae chromosome x. *EMBO J*, 15(9), 2031–49.

[33] Garrson, E. & Marth, G. (2012). Haplotype-based variant detection from short-read sequencing. *arXiv preprint*, (1207.3907 q-bio.GN).

[34] Gotoh, O. (1982). An improved algorithm for matching biological sequences. *J Mol Biol*, 162(3), 705–8.

[35] Gusfield, D. (1997). Algorithms on strings, trees and sequendes. *Cambridge University Press*, University of California, Davis.

[36] Gymrek, M., Golan, D., Rosset, S., & Erlich, Y. (2012). lobstr: A short tandem repeat profiler for personal genomes. *Genome Res*, 22(6), 1154–62.

[37] Heger, M. (2013). Platform comparison study finds pacbio yields most complete assembly of e. coli genomes. *Genomeweb*, (15).

[38] Hoffmann, S., Otto, C., Kurtz, S., Sharma, C. M., Khaitovich, P., Vogel, J., Stadler, P. F., & Hackermüller, J. (2009). Fast mapping of short sequences with mismatches, insertions and deletions using index structures. *PLoS Comput Biol*, 5(9), e1000502.

[39] Hohn, B. (1983). Dna sequences necessary for packaging of bacteriophage lambda dna. *Proc Natl Acad Sci U S A*, 80(24), 7456–60.

[40] Homer, N., Merriman, B., & Nelson, S. F. (2009). Bfast: an alignment tool for large scale genome resequencing. *PLoS One*, 4(11), e7767.

[41] Homer, N. & Nelson, S. F. (2010). Improved variant discovery through local realignment of short-read next-generation sequencing data using srma. *Genome Biol*, 11(10), R99.

[42] Huse, S. M., Huber, J. A., Morrison, H. G., Sogin, M. L., & Welch, D. M. (2007). Accuracy and quality of massively parallel dna pyrosequencing. *Genome Biol*, 8(7), R143.

[43] Idury, R. M. & Waterman, M. S. (1995). A new algorithm for dna sequence assembly. *J Comput Biol*, 2(2), 291–306.

[44] International Human Genome Sequencing Consortium (2004). Finishing the euchromatic sequence of the human genome. *Nature*, 431(7011), 931–45.

[45] Iqbal, Z., Caccamo, M., Turner, I., Flicek, P., & McVean, G. (2012). De novo assembly and genotyping of variants using colored de bruijn graphs. *Nat Genet*, 44(2), 226–32.

[46] Jacobs, P. A., Wilson, C. M., Sprenkle, J. A., Rosenshein, N. B., & Migeon, B. R. (1980). Mechanism of origin of complete hydatidiform moles. *Nature*, 286(5774), 714–6.

[47] Jacq, C., Alt-Mörbe, J., Andre, B., Arnold, W., Bahr, A., Ballesta, J. P., Bargues, M., Baron, L., Becker, A., Biteau, N., Blöcker, H., Blugeon, C., Boskovic, J., Brandt, P., Brückner, M., Buitrago, M. J., Coster, F., Delaveau, T., del Rey, F., Dujon, B., Eide, L. G., Garcia-Cantalejo, J. M., Goffeau, A., Gomez-Peris, A., & Zaccaria, P. (1997). The nucleotide sequence of saccharomyces cerevisiae chromosome iv. *Nature*, 387(6632 Suppl), 75–8.

[48] Jiang, H. & Wong, W. H. (2008). Seqmap: mapping massive amount of oligonucleotides to the genome. *Bioinformatics*, 24(20), 2395–6.

[49] Johnson, D. S., Mortazavi, A., Myers, R. M., & Wold, B. (2007). Genome-wide mapping of in vivo protein-dna interactions. *Science*, 316(5830), 1497–502.

[50] Johnston, M., Andrews, S., Brinkman, R., Cooper, J., Ding, H., Dover, J., Du, Z., Favello, A., Fulton, L., & Gattung, S. (1994). Complete nucleotide sequence of saccharomyces cerevisiae chromosome viii. *Science*, 265(5181), 2077–82.

[51] Johnston, M., Hillier, L., Riles, L., Albermann, K., André, B., Ansorge, W., Benes, V., Brückner, M., Delius, H., Dubois, E., Düsterhöft, A., Entian, K. D., Floeth, M., Goffeau, A., Hebling, U., Heumann, K., Heuss-Neitzel, D., Hilbert, H., Hilger, F., Kleine, K., Kötter, P., Louis, E. J., Messenguy, F., Mewes, H. W., & Hoheisel, J. D. (1997). The nucleotide sequence of saccharomyces cerevisiae chromosome xii. *Nature*, 387(6632 Suppl), 87–90.

[52] Korpar, M. & Šikic, M. (2013). Sw
# -gpu-enabled exact alignments on genome scale. *Bioinformatics*, 29(19), 2494–5.

[53] Krogh, A., Brown, M., Mian, I. S., Sjölander, K., & Haussler, D. (1994). Hidden markov models in computational biology. applications to protein modeling. *J Mol Biol*, 235(5), 1501–31.

[54] Kurtz, S., Phillippy, A., Delcher, A. L., Smoot, M., Shumway, M., Antonescu, C., & Salzberg, S. L. (2004). Versatile and open software for comparing large genomes. *Genome Biol*, 5(2), R12.

[55] Lam, T. W., Sung, W. K., Tam, S. L., Wong, C. K., & Yiu, S. M. (2008). Compressed indexing and local alignment of dna. *Bioinformatics*, 24(6), 791–7.

[56] Lander, E. S., Linton, L. M., Birren, B., Nusbaum, C., Zody, M. C., Baldwin, J., Devon, K., Dewar, K., Doyle, M., FitzHugh, W., Funke, R., Gage, D., Harris, K., Heaford, A., Howland, J., Kann, L., Lehoczky, J., LeVine, R., McEwan, P., McKernan, K., Meldrim, J., Mesirov, J. P., Miranda, C., Morris, W., Naylor, J., Raymond, C., Rosetti, M., Santos, R., Sheridan, A., Sougnez, C., Stange-Thomann, N., Stojanovic, N., Subramanian, A., Wyman, D., Rogers, J., Sulston, J., Ainscough, R., Beck, S., Bentley, D., Burton, J., Clee, C., Carter, N., Coulson, A., Deadman, R., Deloukas, P., Dunham, A., Dunham, I., Durbin, R., French, L., Grafham, D., Gregory, S., Hubbard, T., Humphray, S., Hunt, A., Jones, M., Lloyd, C., McMurray, A., Matthews, L., Mercer, S., Milne, S., Mullikin, J. C., Mungall, A., Plumb, R., Ross, M., Shownkeen, R., Sims, S., Waterston, R. H., Wilson, R. K., Hillier, L. W., McPherson, J. D., Marra, M. A., Mardis, E. R., Fulton, L. A., Chinwalla, A. T., Pepin, K. H., Gish, W. R., Chissoe, S. L., Wendl, M. C., Delehaunty, K. D., Miner, T. L., Delehaunty, A., Kramer, J. B., Cook, L. L., Fulton, R. S., Johnson, D. L., Minx, P. J., Clifton, S. W., Hawkins, T., Branscomb, E., Predki, P., Richardson, P., Wenning, S., Slezak, T., Doggett, N., Cheng, J. F., Olsen, A., Lucas, S., Elkin, C., Uberbacher, E., Frazier, M., Gibbs, R. A., Muzny, D. M., Scherer, S. E., Bouck, J. B., Sodergren, E. J., Worley, K. C., Rives, C. M., Gorrell, J. H., Metzker, M. L., Naylor, S. L., Kucherlapati, R. S., Nelson, D. L., Weinstock, G. M., Sakaki, Y., Fujiyama, A., Hattori, M., Yada, T., Toyoda, A., Itoh, T., Kawagoe, C., Watanabe, H., Totoki, Y., Taylor, T., Weissenbach, J., Heilig, R., Saurin, W., Artiguenave, F., Brottier, P., Bruls, T., Pelletier, E., Robert, C., Wincker, P., Smith, D. R., Doucette-Stamm, L., Rubenfield, M., Weinstock, K., Lee, H. M., Dubois, J., Rosenthal, A., Platzer, M., Nyakatura, G., Taudien, S., Rump, A., Yang, H., Yu, J., Wang, J., Huang, G., Gu, J., Hood, L., Rowen, L., Madan, A., Qin, S., Davis, R. W., Federspiel, N. A., Abola, A. P., Proctor, M. J., Myers, R. M., Schmutz, J., Dickson, M., Grimwood, J., Cox, D. R., Olson, M. V., Kaul, R., Raymond, C., Shimizu, N., Kawasaki, K., Minoshima, S., Evans, G. A., Athanasiou, M., Schultz, R., Roe, B. A., Chen, F., Pan, H., Ramser, J., Lehrach, H., Reinhardt, R., McCombie, W. R., de la Bastide, M., Dedhia, N., Blöcker, H., Hornischer, K., Nordsiek, G., Agarwala, R., Aravind, L., Bailey, J. A., Bateman, A., Batzoglou, S., Birney, E., Bork, P., Brown, D. G., Burge, C. B., Cerutti, L., Chen, H. C., Church, D., Clamp, M., Copley, R. R., Doerks, T., Eddy, S. R., Eichler, E. E., Furey, T. S., Galagan, J., Gilbert, J. G., Harmon, C., Hayashizaki, Y., Haussler, D., Hermjakob, H., Hokamp, K., Jang, W., Johnson, L. S., Jones, T. A., Kasif, S., Kaspryzk, A., Kennedy, S., Kent, W. J., Kitts, P., Koonin, E. V., Korf, I., Kulp, D., Lancet, D., Lowe, T. M., McLysaght, A., Mikkelsen, T., Moran, J. V., Mulder, N., Pollara, V. J., Ponting, C. P., Schuler, G., Schultz, J., Slater, G., Smit, A. F., Stupka, E., Szustakowski, J., Thierry-Mieg, D., Thierry-Mieg, J., Wagner, L., Wallis, J., Wheeler, R., Williams, A., Wolf, Y. I., Wolfe, K. H., Yang, S. P., Yeh, R. F., Collins, F., Guyer, M. S., Peterson, J., Felsenfeld, A., Wetterstrand, K. A., Patrinos, A., Morgan, M. J., de Jong, P., Catanese, J. J., Osoegawa, K., Shizuya, H., Choi, S., Chen, Y. J., Szustakowki, J., & International Human Genome Sequencing Consortium (2001). Initial sequencing and analysis of the human genome. *Nature*, 409(6822), 860–921.

[Langmead] Langmead, B. Overlap layout consensus assembly. Johns Hopkins, Whiting School of Engineering.

[58] Langmead, B. & Salzberg, S. L. (2012). Fast gapped-read alignment with bowtie 2. *Nat Methods*, 9(4), 357–9.

[59] Langmead, B., Trapnell, C., Pop, M., & Salzberg, S. L. (2009). Ultrafast and memory-efficient alignment of short dna sequences to the human genome. *Genome Biol*, 10(3), R25.

[60] Lee, W.-P., Stromberg, M. P., Ward, A., Stewart, C., Garrison, E. P., & Marth, G. T. (2014). Mosaik: a hash-based algorithm for accurate next-generation sequencing short-read mapping. *PLoS One*, 9(3), e90581.

[61] Li, H. (2011). Improving snp discovery by base alignment quality. *Bioinformatics*, 27(8), 1157–8.

[62] Li, H. (2012). Exploring single-sample snp and indel calling with whole-genome de novo assembly. *Bioinformatics*, 28(14), 1838–44.

[63] Li, H. & Durbin, R. (2010). Fast and accurate long-read alignment with burrows-wheeler transform. *Bioinformatics*, 26(5), 589–95.

[64] Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., Durbin, R., & 1000 Genome Project Data Processing Subgroup (2009a). The sequence alignment/map format and samtools. *Bioinformatics*, 25(16), 2078–9.

[65] Li, H. & Homer, N. (2010). A survey of sequence alignment algorithms for next-generation sequencing. *Brief Bioinform*, 11(5), 473–83.

[66] Li, H., Ruan, J., & Durbin, R. (2008a). Mapping short dna sequencing reads and calling variants using mapping quality scores. *Genome Res*, 18(11), 1851–8.

[67] Li, M., Ma, B., Kisman, D., & Tromp, J. (2003). Patternhunter ii: highly sensitive and fast homology search. *Genome Inform*, 14, 164–75.

[68] Li, M., Ma, B., Kisman, D., & Tromp, J. (2004). Patternhunter ii: highly sensitive and fast homology search. *J Bioinform Comput Biol*, 2(3), 417–39.

[69] Li, R., Li, Y., Fang, X., Yang, H., Wang, J., Kristiansen, K., & Wang, J. (2009b). Snp detection for massively parallel whole-genome resequencing. *Genome Res*, 19(6), 1124–32.

[70] Li, R., Li, Y., Kristiansen, K., & Wang, J. (2008b). Soap: short oligonucleotide alignment program. *Bioinformatics*, 24(5), 713–4.

[71] Li, R., Yu, C., Li, Y., Lam, T.-W., Yiu, S.-M., Kristiansen, K., & Wang, J. (2009c). Soap2: an improved ultrafast tool for short read alignment. *Bioinformatics*, 25(15), 1966–7.

[72] Li, R., Zhu, H., Ruan, J., Qian, W., Fang, X., Shi, Z., Li, Y., Li, S., Shan, G., Kristiansen, K., Li, S., Yang, H., Wang, J., & Wang, J. (2010). De novo assembly of human genomes with massively parallel short read sequencing. *Genome Res*, 20(2), 265–72.

[73] Li, S., Li, R., Li, H., Lu, J., Li, Y., Bolund, L., Schierup, M. H., & Wang, J. (2013). Soapindel: efficient identification of indels from short paired reads. *Genome Res*, 23(1), 195–200.

[74] Liu, L., Li, Y., Li, S., Hu, N., He, Y., Pong, R., Lin, D., Lu, L., & Law, M. (2012). Comparison of next-generation sequencing systems. *J Biomed Biotechnol*, 2012, 251364.

[75] Liu, Y., Schmidt, B., & Maskell, D. L. (2010). Cudasw++2.0: enhanced smith-waterman protein database search on cuda-enabled gpus based on simt and virtualized simd abstractions. *BMC Res Notes*, 3, 93.

[76] Liu, Y., Wirawan, A., & Schmidt, B. (2013). Cudasw++ 3.0: accelerating smith-waterman protein database search by coupling cpu and gpu simd instructions. *BMC Bioinformatics*, 14, 117.

[77] Luke, S., C. Balan, G., Panait, L., Cioffi-Revilla, C., & Paus, S. (2003). Mason: A new multi-agent simulation toolkit. *Proceedings of the Agent 2003 Conference.*

[78] Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K., & Balan, G. (2004). Mason: A new multi-agent simulation toolkit. *Proceedings of the 2004 SwarmFest Workshop.*

[79] Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K., & Balan, G. (2005). Mason: A multi-agent simulation environment. *In Simulation: Transactions of the society for Modeling and Simulation International.*, 82(7), 517–527.

[80] Lunter, G. (2007). Probabilistic whole-genome alignments reveal high indel rates in the human and mouse genomes. *Bioinformatics*, 23(13), i289–96.

[81] Lunter, G. & Goodson, M. (2011). Stampy: a statistical algorithm for sensitive and fast mapping of illumina sequence reads. *Genome Res*, 21(6), 936–9.

[82] Ma, B., Tromp, J., & Li, M. (2002). Patternhunter: faster and more sensitive homology search. *Bioinformatics*, 18(3), 440–5.

[83] Marco-Sola, S., Sammeth, M., Guigó, R., & Ribeca, P. (2012). The gem mapper: fast, accurate and versatile alignment by filtration. *Nat Methods*, 9(12), 1185–8.

[84] Mardis, E. R. (2008). The impact of next-generation sequencing technology on genetics. *Trends Genet*, 24(3), 133–41.

[85] McKenna, A., Hanna, M., Banks, E., Sivachenko, A., Cibulskis, K., Kernytsky, A., Garimella, K., Altshuler, D., Gabriel, S., Daly, M., & DePristo, M. A. (2010). The genome analysis toolkit: a mapreduce framework for analyzing next-generation dna sequencing data. *Genome Res*, 20(9), 1297–303.

[86] Meek, C., Patel, J. M., & Kasetty, S. (2003). Oasis: an online and accurate technique for local-alignment searches on biological sequences. *Proceedings of 29th International Conference on Very Large Data Bases (VLDB 2003)*, Berlin, 910–21.

[87] Metzker, M. L. (2010). Sequencing technologies - the next generation. *Nature Reviews*, 11, 31–45.

[88] Mount, D. W. (2004). *Bioinformatics: sequence and genome analysis*. Cold Spring Harbor, N.Y.: Cold Spring Harbor Laboratory Press, 2nd ed edition.

[89] Murakami, Y., Naitou, M., Hagiwara, H., Shibata, T., Ozawa, M., Sasanuma, S., Sasanuma, M., Tsuchiya, Y., Soeda, E., & Yokoyama, K. (1995). Analysis of the nucleotide sequence of chromosome vi from saccharomyces cerevisiae. *Nat Genet*, 10(3), 261–8.

[90] Myers, E. W. (1995). Toward simplifying and accurately formulating fragment assembly. *J Comput Biol*, 2(2), 275–90.

[91] Myers, G. (1999). A fast bit-vector algorithm for approximate string matching based on dynamic programming. *Journal of the ACM*, 46(3), 395–415.

[92] Oliver, S. G., van der Aart, Q. J., Agostoni-Carbone, M. L., Aigle, M., Alberghina, L., Alexandraki, D., Antoine, G., Anwar, R., Ballesta, J. P., & Benit, P. (1992). The complete dna sequence of yeast chromosome iii. *Nature*, 357(6373), 38–46.

[93] Pavlopoulos, G. A., Oulas, A., Iacucci, E., Sifrim, A., Moreau, Y., Schneider, R., Aerts, J., & Iliopoulos, I. (2013). Unraveling genomic variation from next generation sequencing data. *BioData Min*, 6(1), 13.

[94] Philippsen, P., Kleine, K., Pöhlmann, R., Düsterhöft, A., Hamberg, K., Hegemann, J. H., Obermaier, B., Urrestarazu, L. A., Aert, R., Albermann, K., Altmann, R., André, B., Baladron, V., Ballesta, J. P., Bécam, A. M., Beinhauer, J., Boskovic, J., Buitrago, M. J., Bussereau, F., Coster, F., Crouzet, M., D'Angelo, M., Dal Pero, F., De Antoni, A., Del Rey, F., Doignon, F., Domdey, H., Dubois, E., Fiedler, T., Fleig, U., Floeth, M., Fritz, C., Gaillardin, C., Garcia-Cantalejo, J. M., Glansdorff, N. N., Goffeau, A., Gueldener, U., Herbert, C., Heumann, K., Heuss-Neitzel, D., Hilbert, H., Hinni, K., Iraqui Houssaini, I., Jacquet, M., Jimenez, A., Jonniaux, J. L., Karpfinger, L., Lanfranchi, G., Lepingle, A., Levesque, H., Lyck, R., Maftahi, M., Mallet, L., Maurer, K. C., Messenguy, F., Mewes, H. W., Mösti, D., Nasr, F., Nicaud, J. M., Niedenthal, R. K., Pandolfo, D., Piérard, A., Piravandi, E., Planta, R. J., Pohl, T. M., Purnelle, B., Rebischung, C., Remacha, M., Revuelta, J. L., Rinke, M., Saiz, J. E., Sartorello, F., Scherens, B., Sen-Gupta, M., Soler-Mira, A., Urbanus, J. H., Valle, G., Van Dyck, L., Verhasselt, P., Vierendeels, F., Vissers, S., Voet, M., Volckaert, G., Wach, A., Wambutt, R., Wedler, H., Zollner, A., & Hani, J. (1997). The nucleotide sequence of saccharomyces cerevisiae chromosome xiv and its evolutionary implications. *Nature*, 387(6632 Suppl), 93–8.

[95] Qi, J., Zhao, F., Buboltz, A., & Schuster, S. C. (2010). ingap: an integrated next-generation genome analysis pipeline. *Bioinformatics*, 26(1), 127–9.

[96] Quail, M. A., Smith, M., Coupland, P., Otto, T. D., Harris, S. R., Connor, T. R., Bertoni, A., Swerdlow, H. P., & Gu, Y. (2012). A tale of three next generation sequencing platforms: comparison of ion torrent, pacific biosciences and illumina miseq sequencers. *BMC Genomics*, 13, 341.

[97] Rabiner, L. R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proc. IEEE*, (77), 257–86.

[98] Rognes, T. (2011). Faster smith-waterman database searches with inter-sequence simd parallelisation. *BMC Bioinformatics*, 12, 221.

[99] Rognes, T. & Seeberg, E. (2000). Six-fold speed-up of smith-waterman sequence database searches using parallel processing on common microprocessors. *Bioinformatics*, 16(8), 699–706.

[100] Ronaghi, M., Karamohamed, S., Pettersson, B., Uhlén, M., & Nyrén, P. (1996). Real-time dna sequencing using detection of pyrophosphate release. *Anal Biochem*, 242(1), 84–9.

[101] Rumble, S. M., Lacroute, P., Dalca, A. V., Fiume, M., Sidow, A., & Brudno, M. (2009). Shrimp: accurate mapping of short color-space reads. *PLoS Comput Biol*, 5(5), e1000386.

[102] Rusk, N. (2009). Cheap third-generation sequencing. *Nature Method*, (6), 244.

[103] Rusk, N. (2011). Torrents of sequence. *Nature Methods*, 8, 44.

[104] Sachidanandam, R., Weissman, D., Schmidt, S. C., Kakol, J. M., Stein, L. D., Marth, G., Sherry, S., Mullikin, J. C., Mortimore, B. J., Willey, D. L., Hunt, S. E., Cole, C. G., Coggill, P. C., Rice, C. M., Ning, Z., Rogers, J., Bentley, D. R., Kwok, P. Y., Mardis, E. R., Yeh, R. T., Schultz, B., Cook, L., Davenport, R., Dante, M., Fulton, L., Hillier, L., Waterston, R. H., McPherson, J. D., Gilman, B., Schaffner, S., Van Etten, W. J., Reich, D., Higgins, J., Daly, M. J., Blumenstiel, B., Baldwin, J., Stange-Thomann, N., Zody, M. C., Linton, L., Lander, E. S., Altshuler, D., & International SNP Map Working Group (2001). A map of human genome sequence variation containing 1.42 million single nucleotide polymorphisms. *Nature*, 409(6822), 928–33.

[105] Sanger, F., Nicklen, S., & Coulson, A. R. (1992). Dna sequencing with chain-terminating inhibitors. 1977. *Biotechnology*, 24, 104–8.

[106] Simpson, J. T. & Durbin, R. (2012). Efficient de novo assembly of large genomes using compressed data structures. *Genome Res*, 22(3), 549–56.

[107] Smith, T. F. & Waterman, M. S. (1981). Identification of common molecular subsequences. *J Mol Biol*, 147(1), 195–7.

[108] Szalkowski, A., Ledergerber, C., Krähenbühl, P., & Dessimoz, C. (2008). Swps3 - fast multi-threaded vectorized smith-waterman for ibm cell/b.e. and x86/sse2. *BMC Res Notes*, 1, 107.

[109] Tettelin, H., Agostoni Carbone, M. L., Albermann, K., Albers, M., Arroyo, J., Backes, U., Barreiros, T., Bertani, I., Bjourson, A. J., Brückner, M., Bruschi, C. V., Carignani, G., Castagnoli, L., Cerdan, E., Clemente, M. L., Coblenz, A., Coglievina, M., Coissac, E., Defoor, E., Del Bino, S., Delius, H., Delneri, D., de Wergifosse, P., Dujon, B., & Kleine, K. (1997). The nucleotide sequence of saccharomyces cerevisiae chromosome vii. *Nature*, 387(6632 Suppl), 81–4.

[110] Thompson, J. F. & Steinmann, K. E. (2010). Single molecule sequencing with a heliscope genetic analysis system. *Curr Protoc Mol Biol*, Chapter 7, Unit7.10.

[111] Venter, J. C., Adams, M. D., Myers, E. W., Li, P. W., Mural, R. J., Sutton, G. G., Smith, H. O., Yandell, M., Evans, C. A., Holt, R. A., Gocayne, J. D., Amanatides, P., Ballew, R. M., Huson, D. H., Wortman, J. R., Zhang, Q., Kodira, C. D., Zheng, X. H., Chen, L., Skupski, M., Subramanian, G., Thomas, P. D., Zhang, J., Gabor Miklos, G. L., Nelson, C., Broder, S., Clark, A. G., Nadeau, J., McKusick, V. A., Zinder, N., Levine, A. J., Roberts, R. J., Simon, M., Slayman, C., Hunkapiller, M., Bolanos, R., Delcher, A., Dew, I., Fasulo, D., Flanigan, M., Florea, L., Halpern, A., Hannenhalli, S., Kravitz, S., Levy, S., Mobarry, C., Reinert, K., Remington, K., Abu-Threideh, J., Beasley, E., Biddick, K., Bonazzi, V., Brandon, R., Cargill, M., Chandramouliswaran, I., Charlab, R., Chaturvedi, K., Deng, Z., Di Francesco, V., Dunn, P., Eilbeck, K., Evangelista, C., Gabrielian, A. E., Gan, W., Ge, W., Gong, F., Gu, Z., Guan, P., Heiman, T. J., Higgins, M. E., Ji, R. R., Ke, Z., Ketchum, K. A., Lai, Z., Lei, Y., Li, Z., Li, J., Liang, Y., Lin, X., Lu, F., Merkulov, G. V., Milshina, N., Moore, H. M., Naik, A. K., Narayan, V. A., Neelam, B., Nusskern, D., Rusch, D. B., Salzberg, S., Shao, W., Shue, B., Sun, J., Wang, Z., Wang, A., Wang, X., Wang, J., Wei, M., Wides, R., Xiao, C., Yan, C., Yao, A., Ye, J., Zhan, M., Zhang, W., Zhang, H., Zhao, Q., Zheng, L., Zhong, F., Zhong, W., Zhu, S., Zhao, S., Gilbert, D., Baumhueter, S., Spier, G., Carter, C., Cravchik, A., Woodage, T., Ali, F., An, H., Awe, A., Baldwin, D., Baden, H., Barnstead, M., Barrow, I., Beeson, K., Busam, D., Carver, A., Center, A., Cheng, M. L., Curry, L., Danaher, S., Davenport, L., Desilets, R., Dietz, S., Dodson, K., Doup, L., Ferriera, S., Garg, N., Gluecksmann, A., Hart, B., Haynes, J., Haynes, C., Heiner, C., Hladun, S., Hostin, D., Houck, J., Howland, T., Ibegwam, C., Johnson, J., Kalush, F., Kline, L., Koduru, S., Love, A., Mann, F., May, D., McCawley, S., McIntosh, T., McMullen, I., Moy, M., Moy, L., Murphy, B., Nelson, K., Pfannkoch, C., Pratts, E., Puri, V., Qureshi, H., Reardon, M., Rodriguez, R., Rogers, Y. H., Romblad, D., Ruhfel, B., Scott, R., Sitter, C., Smallwood, M., Stewart, E., Strong, R., Suh, E., Thomas, R., Tint, N. N., Tse, S., Vech, C., Wang, G., Wetter, J., Williams, S., Williams, M., Windsor, S., Winn-Deen, E., Wolfe, K., Zaveri, J., Zaveri, K., Abril, J. F., Guigó, R., Campbell, M. J., Sjolander, K. V., Karlak, B., Kejariwal, A., Mi, H., Lazareva, B., Hatton, T., Narechania, A., Diemer, K., Muruganujan, A., Guo, N., Sato, S., Bafna, V., Istrail, S., Lippert, R., Schwartz, R., Walenz, B., Yooseph, S., Allen, D., Basu, A., Baxendale, J., Blick, L., Caminha, M., Carnes-Stine, J., Caulk, P., Chiang, Y. H., Coyne, M., Dahlke, C., Mays, A., Dombroski, M., Donnelly, M., Ely, D., Esparham, S., Fosler, C., Gire, H., Glanowski, S., Glasser, K., Glodek, A., Gorokhov, M., Graham, K., Gropman, B., Harris, M., Heil, J., Henderson, S., Hoover, J., Jennings, D., Jordan, C., Jordan, J., Kasha, J., Kagan, L., Kraft, C., Levitsky, A., Lewis, M., Liu, X., Lopez, J., Ma, D., Ma-

joros, W., McDaniel, J., Murphy, S., Newman, M., Nguyen, T., Nguyen, N., Nodell, M., Pan, S., Peck, J., Peterson, M., Rowe, W., Sanders, R., Scott, J., Simpson, M., Smith, T., Sprague, A., Stockwell, T., Turner, R., Venter, E., Wang, M., Wen, M., Wu, D., Wu, M., Xia, A., Zandieh, A., & Zhu, X. (2001). The sequence of the human genome. *Science*, 291(5507), 1304–51.

[112] Weese, D., Emde, A.-K., Rausch, T., Döring, A., & Reinert, K. (2009). Razers–fast read mapping with sensitivity control. *Genome Res*, 19(9), 1646–54.

[113] Weese, D., Holtgrewe, M., & Reinert, K. (2012). Razers 3: faster, fully sensitive read mapping. *Bioinformatics*, 28(20), 2592–9.

[114] Wei, N., Bemmels, J. B., & Dick, C. W. (2014). The effects of read length, quality and quantity on microsatellite discovery and primer development: from illumina to pacbio. *Molecular ecology resources*, 14(5), 953–965.

[115] Wheeler, D. A., Srinivasan, M., Egholm, M., Shen, Y., Chen, L., McGuire, A., He, W., Chen, Y.-J., Makhijani, V., Roth, G. T., Gomes, X., Tartaro, K., Niazi, F., Turcotte, C. L., Irzyk, G. P., Lupski, J. R., Chinault, C., Song, X.-z., Liu, Y., Yuan, Y., Nazareth, L., Qin, X., Muzny, D. M., Margulies, M., Weinstock, G. M., Gibbs, R. A., & Rothberg, J. M. (2008). The complete genome of an individual by massively parallel dna sequencing. *Nature*, 452(7189), 872–6.

[116] Wozniak, A. (1997). Using video-oriented instructions to speed up sequence comparison. *Comput Appl Biosci*, 13(2), 145–50.

[117] Zeng, F., Jiang, R., & Chen, T. (2013a). Pyrohmmsnp: an snp caller for ion torrent and 454 sequencing data. *Nucleic Acids Res*, 41(13), e136.

[118] Zeng, F., Jiang, R., & Chen, T. (2013b). Pyrohmmvar: a sensitive and accurate method to call short indels and snps for ion torrent and 454 data. *Bioinformatics*, 29(22), 2859–68.

[119] Zerbino, D. R. & Birney, E. (2008). Velvet: algorithms for de novo short read assembly using de bruijn graphs. *Genome Res*, 18(5), 821–9.

[120] Zhao, M., Lee, W.-P., Garrison, E. P., & Marth, G. T. (2013). Ssw library: an simd smith-waterman c/c++ library for use in genomic applications. *PLoS One*, 8(12), e82138.

[121] Zhao, Z. & Boerwinkle, E. (2002). Neighboring-nucleotide effects on single nucleotide polymorphisms: a study of 2.6 million polymorphisms across the human genome. *Genome Res*, 12(11), 1679–86.