博 士 論 文

Inferring Chromosome Structures with Bidirected Graphs Constructed from Genomic Structural Variations

(ゲノム構造多型に基づく双方向グラフを用いた染色体構造推定)

安 田 知 弘

INFERRING CHROMOSOME STRUCTURES WITH BIDIRECTED GRAPHS CONSTRUCTED FROM GENOMIC STRUCTURAL VARIATIONS
ゲノム構造多型に基づく双方向グラフを用いた染色体構造推定

by

Tomohiro Yasuda
安田知弘

A Doctor Thesis
博士論文

Submitted to
the Graduate School of the University of Tokyo
on December 12, 2014
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Information Science and Technology
in Computer Science

Thesis Supervisor: Satoru Miyano　宮野悟
Professor of Computer Science

ABSTRACT

The analysis of mutations in genomes are necessary to understand biological functions of genomes. Among various types of mutations, structure variations (SVs) are large scale mutations typically larger than 1 kb, and are attracting attentions. Examples of SVs include large deletions, insertions, inversions, translocations, and copy number variations. Next generation sequencing (NGS) technologies have made it possible to exhaustively detect SVs in genomes of thousands of individuals, including patients of cancers and serious congenital diseases. In addition to NGS technologies, a recent sequencing technology that can determine thousands of contiguous bases at once is now available. By using this technology, it would be possible to more easily and accurately obtain genome sequences in de novo manner.

To understand the impact of SVs on biological functions and infer possible mechanisms that caused SVs, it is vital to develop computational methods that accurately detect existence of SVs and their positions in the genome by using genome sequences obtained by these sequencing technologies. Because of the importance of SVs, a number of methods have been already proposed for detecting SVs by using NGS sequences. Although detected SVs should be further used to analyze their impact on genomes, computational tools that annotate detected SVs have not been as intensively developed as detection tools. In particular, although SVs are only local information in genomes, computational methods that utilize detected SVs to infer global structure of chromosomes are not well established. This can hamper our understanding of the effect of SVs, e.g. structures of proteins or regulations of genes affected by SVs. In addition, accurate detection of SVs are still difficult problem even with many existing methods, because the length of NGS sequences are limited to only a few hundred bases and thus detection of SVs involves finding complex patterns hidden in an enormous number of alignments of NGS sequences with the reference genome. Determining positions of SVs is obviously more difficult than determining existence of SVs. Nonetheless, knowing accurate positions of SVs clarifies the range of genomic sequences affected by SVs, and is also necessary for inferring the biological significance of SVs or detecting mechanisms that had generated SVs. Most of existing methods take alignments of NGS sequences and the reference genome as input, and detect aberrant patterns of alignments as signatures that indicate existence and positions of SVs. Such signatures include aberrant distances and/or strands of alignments between paired reads obtained by NGS and the reference genome, aberrant number of aligned NGS sequences in a specific genomic region, and fragmented alignments. Because these signatures have already exploited, a new signature should be used to improve the existing methods. Because using NGS sequences to detect SVs include a computationally intensive step of mapping, i.e. aligning NGS sequences with the reference genome, accelerating this step is also crucial for exhaustive SV detection. In this thesis, we address these problems.

First, we address the problem of reconstructing global structures of chromosomes by using detected SVs. The problem had been previously formulated by Oesper et al. as an optimization problem on a graph constructed from SVs, which can be solved by calculating an optimal flow on the graph. However, they did not deeply analyzed computational complexity of the problem. In addition, the number and the length of chromosomes had not been considered. We formulate a new problem termed as the chromosome problem (ChrP) that takes into account the number and the length of chromosomes as well. In addition, we prove that ChrP is NP-complete by showing that there is an upper bound on the size of chromosomes in an optimal solution, and by reducing the Hamiltonian Cycle problem to ChrP. We also propose a biologically meaningful restriction on instances of the problem, termed as the weakly-connected constraint (WCC), and a variation of ChrP, termed as ChrW. ChrW imposes WCC on instances and removes limitation on the length of chromosomes. These modifications allow ChrW to be solvable in polynomial-time. Moreover, to show that removal of limitation of the length of chromosomes is necessary, another variation of ChrP that only imposes WCC on instances is defined and is proved to be NP-complete. Our result establishes a theoretical foundation of software tools emerging for the analysis of global structures of rearranged chromosomes. In computational experiments, our algorithm that solves ChrW was confirmed to be able to reduce noise in simulated SV data that include modified copy number variations (CNVs) or false positive translocations.

Second, we propose a method ChopSticks that accurately predicts positions of homozygous deletions, which is one of various types of SVs. ChopSticks mainly improves positions of homozygous deletions detected by finding alignments of paired reads with aberrant mapping distances. The paired reads with such alignments are called discordant reads. Positions of deletions calculated by using only discordant reads involve ambiguity. To reduce this ambiguity and to narrow down positions of boundaries of homozygous deletions, ChopSticks takes into account normally mapped sequences that have not been fully exploited by other methods, in addition to discordant reads. We theoretically prove that the expected distance between true positions and positions predicted by ChopSticks is close to distances of previous methods applied to NGS sequences with doubled depth of coverage. Experimental results also witnessed that our method is useful to predict accurate positions of homozygous deletions detected not only by using discordant reads but also by detecting drops of copy numbers.

Moreover, toward faster mapping of NGS sequences to the reference genome, we port widely used mapping programs to a many-core processor Xeon Phi. In a computational experiment, the performances of the ported programs increased as the number of threads increased up to at least 60. This result indicates that concurrent execution of tens of threads on a many-core is promising for future performance improvement.

In this thesis, we also address the problem of detecting SVs by comparing multiple genome sequences constructed by de novo assembly. Such a method will be useful when the new technology that can generate long sequences becomes widely available in the future. Assuming that the input sequences are concatenations of a hidden set of sequences, our method infers the hidden sequences from the concatenations. To this end, we define a class of strings, called disjoint common substrings (DCS's). DCS's are similar to hidden strings and are nonetheless efficiently identified from given concatenations. Our algorithm identifies all DCS's in time linear to the total length of given concatenations. The effectiveness of our method were confirmed by a computational experiment.

論文要旨

ゲノムの生物学的機能を理解するためには、ゲノムの変異を解析することが必須である。様々な変異のうち、構造多型 (structural variation, SV) と呼ばれる、通常 1kbp 以上の大型の変異が、注目を集めている。構造多型の例として、長い配列の欠失、挿入、反転、転座、コピー数多型が挙げられる。次世代シーケンシング (NGS) により、個人のゲノムに存在する構造多型 (structural variation, SV) を網羅的に検出することが可能となり、癌や重篤な先天性異常の患者をはじめとする、数千人規模の個人の SV の解析が可能となった。しかも、NGS よりもさらに新しい配列決定技術により、数千塩基が連続する配列を一度に読むことが可能となってきている。この技術により、ゲノム配列を de novo で決定することが容易になる。

こうした技術により得られる配列データを用いて、SV の生物の機能への影響を理解するとともに、SV を発生させる機構を推定するために、SV の存在および SV の位置を正確に検出する計算手法が不可欠である。SV は重要なため、NGS 配列に基づき SV を検出するための手法が、既に多数提案されている。しかし、検出された SV はゲノムへの影響を解析するために活用されなければならないが、見付かった SV を解析するための計算技術は検出技術ほどには開発が進んでいない。特に、SV がゲノムの局所的な情報にとどまるにも拘らず、検出された SV を用いて染色体の大域的構造を推定する計算手法は、確立されていない。このことは、蛋白質の構造や遺伝子の制御に対する SV の影響を調べる際の障害となり得る。さらに、NGS で得られる配列の長さが数百塩基にとどまることから、SV を検出するためには、NGS 配列の参照ゲノム配列の膨大な数のアラインメントに隠された複雑なパターンを探し出す必要があり、多数の手法が存在するにも拘らず、正確に SV を検出することはいまだに困難である。SV の存在だけでなく、位置も決めることはさらに困難である。にもかかわらず、SV の正確な位置を知ることは、SV によって影響を受けるゲノムの範囲を明確化し、SV の生物学的意味や SV の発生機構を推測するために必要である。殆どの既存手法は、NGS 配列と参照ゲノム配列とのアラインメントを入力とし、異常なアラインメントのパターンを、SV の存在および位置を示す特徴として利用する。こうした特徴には、距離またはストランドに異常があるペアエンド配列のアラインメント、ゲノム上の特定の領域におけるアラインメントの数の異常、断片化されたアラインメントが含まれる。既存手法がこれらを既に活用しているため、既存手法を改善にするには新しい特徴を用いるべきである。NGS 配列を用いて SV を検出する手法は、NGS 配列と参照ゲノムのアラインメントを行なうマッピング処理を含むが、これは計算負荷の大きな処理であるため、この高速化も SV の網羅的検出に重要である。本博士論文では、これらの課題を扱う。

第一に、検出された SV を用いて、染色体の大域構造を再構築する問題を扱う。この問題を Oesper らが SV に基づき構築したグラフ上の最適化問題として定式化しており、グラフ上で最適なフローを計算することで解を得ることができる。しかし Oesper らは、問題の計算論的な困難さについて深い考察は行なっていない。しかも、染色体の数や長さは考慮されていない。我々は、染色体の数や長さも考慮する chromosome problem (ChrP) を定式化した。さらに、ChrP が NP 完全であることを、解として得られる染色体の大きさに上限があり、またよく知られているハミルトニアン回路問題が ChrP に還元可能であることを示すことで、証明した。さらに我々は、問題の入力を制約する、生物学的な意味のある弱連結性制約 (weakly-connected constraint, WCC) および、WCC を入力に課すとともに

染色体長の上限を撤廃した新たな問題 ChrW を定義した。これらの修正により、ChrW は多項式時間で解くことができる。さらに、染色体長の制約を除去することが必要であることを示すために、ChrP に WCC を課しただけの問題が NP 完全であることを示した。これらの結果は、現在開発されつつある、変異した染色体の大域構造を解析するソフトウェアツールの、理論的基盤となる。計算機実験において、ChrW を解く提案アルゴリズムが、シミュレーションデータにおいてコピー数多型 (CNVs) に発生したノイズや、偽陽性の転座を低減できることを確認した。

第二に、SV の一種であるホモ接合性の欠失について、位置の正確な予測を行なう手法である ChopSticks を提案する。ChopSticks が主に解析対象とするホモ接合性の欠失は、ペアエンド配列の距離の異常を検出する方法により検出されるものである。このようなペアエンド配列は、discordant reads と呼ばれる。欠失の位置を discordant reads のみを用いて計算すると曖昧さが残るが、この曖昧さを除去し欠失の範囲を限定するために ChopSticks は正常にアラインメントされる concordant reads も併用する。理論的解析により、実際の欠失の位置と ChopSticks が予想した位置の差の期待値は、既存手法で配列量を 2 倍にした場合に近いことを示した。さらに実験により、ChopSticks は discordant reads を用いて検出された欠失のみならず、コピー数の減少により検出された欠失の位置を改善するためにも使用できることを実証した。

さらに、NGS 配列を参照ゲノム配列へマッピングする処理の高速化に向けて、広く使われているマッピングプログラムをメニーコアプロセッサ Xeon Phi へ移植した。計算機実験では、スレッド数を増加させたとき、少なくとも 60 スレッドまで性能が向上した。この結果から、メニーコアプロセッサ上で数十スレッドを並列実行することが、将来の高速化に寄与すると期待される。

本博士論文では、de novo アセンブリにより得られた複数のゲノム配列を相互比較することで、SV を検出する手法も考察する。このような手法は、長い配列を一度に決定できるシーケンシング技術が広く普及したときに、有用になると期待される。提案手法は、入力される文字列が未知の文字列集合の文字列を連結して得られたとの仮定のもとに、この未知の文字列集合を推測する。そのために、disjoint common substring (DCS) と呼ばれる文字列のクラスを定義した。DCS は、未知の各文字列に近い文字列であるにもかかわらず、連結された文字列から効率良く計算できる。本研究では、全ての DCS を入力文字列の長さの総和に対し線形の時間で計算可能なアルゴリズムを開発し、その有用性を計算機実験により確認した。

Acknowledgements

I am deeply grateful to my supervisor, Professor Satoru Miyano, for supervising my studies while I belong to the University of Tokyo, as well as his kind support while I work in Hitachi, Ltd. I would like to thank Associate Professor Seiya Imoto, for helpful advices and discussions. I would also like to thank Professor Masao Nagasaki and Shin Suzuki of Tohoku University for their support on my study. I am also grateful for Dr. Asako Koike of Hitachi, Ltd. who helped me study in university as an adult student.

This work is partially supported by the New Energy and Industrial Technology Development Organization (NEDO), Japan.

Contents

Chapter 1

Introduction

1.1 Background and Overview of Existing Works

Next-generation sequencing (NGS) technologies have drastically reduced the cost of genome sequencing[1]. Today, NGS technologies are essential tools in genome analysis, because they enable us to simultaneously obtain sequences of more than trillions of base pairs at maximum[2]. NGS technologies enable the characterization of not only small variations such as single-nucleotide polymorphisms (SNPs) but also large-scale mutations such as insertions, deletions, tandem duplications, and inversions. Mutations of these types are collectively called structural variations (SVs). In addition to NGS technologies, more recent technology [14] can produce sequences consisting of thousands of bases at once. Such a technology makes *de novo* assembly of genomes easier and will be an important tool to detect SVs in the future. As more genomic sequences have become available, it has become clear that genomes contain many SVs. In fact, SVs are frequently observed even in healthy individuals [55, 58, 79]. Because SVs affect much larger portions of genomes than small variations, e.g. SNPs, they have a great impact on biological functions [35]. SVs have already been associated with diverse diseases [82]. For example, the fusion genes BCR-ABL and EML4-ALK play key roles in the development of cancer, and it is believed that other recurrent rearrangements remain to be discovered [6].

To understand functions of genomes affected by SVs from molecular level, it is necessary to determine each SV as accurate as possible. However, just detecting SVs is insufficient because each SV is only a local information on a genome. Structures of chromosomes affected by SVs are still unknown even after individual SVs are detected. In this thesis, we address problems of detecting and analyzing SVs. First, we aim to infer global structure of chromosomes from detected SVs for the purpose of elucidating the organization of genomes. Second, we improve both accuracy and performance of detecting SVs from NGS sequences. Finally, we propose an algorithm that compares multiple genome sequences with each other to detect SVs.

Inferring global structures of chromosomes is difficult when genomes are affected by complex rearrangements. In cancer genomes, many SVs may be concentrated in a small region of the genome [8, 50, 78]. It has been suggested that a single catastrophic mutational event, known as *chromothripsis* [78], causes these concentrations. A study of prostate cancer also uncovered a distinct type of complex rearrangement termed *chromoplexy* [5, 73], wherein rearrangements are unclustered but involve multiple chromosomes. Complex genomic rearrangements

[1]`http://www.genome.gov/sequencingcosts/`
[2]`http://www.illumina.com/systems/sequencing.ilmn`

have even been observed in germline mutations, resulting in serious congenital diseases [29, 30]. Oesper et al. [66] proposed a problem of inferring the global structure of chromosomes from SVs, and developed an algorithm called *paired-end reconstruction of genome organization (PREGO)*. However, they did not deeply analyzed the computational complexity of the problem. Medvedev et al. [57] also considered a similar problem. However, they only considered a case where all fragments in the reference genome are included in the genome to be analyzed. Here, the *reference genome* is known and is a pre-existing sequenced genome of the same organism, such as the GRCh38 build of the human genome[3].

Because of the importance of SVs in functions of the genome, a lot of methods have been developed for finding SVs [1, 10, 25, 55, 58, 70, 87]. The purpose of these methods are to detect existence of SVs and to determine their positions as accurate as possible. Current NGS technologies can sequence paired reads, which are pairs of reads several hundred bases away from each other. This ability is useful for analyzing SVs. First, paired reads can be aligned with the reference genome more accurately than single reads. Second, we can analyze structures of genomes larger than the size of each read. SVs are detected by finding aberrant mapping patterns of paired reads to the reference genome, which are called *signatures* [58]. If paired reads have aberrant mapping distances and/or strands, they are likely caused by SVs. This signature is called *Read Pair (RP)* [10, 25, 40, 54, 70]. If the number of reads mapped to a region in the reference genome is extremely smaller or larger than expected, it is also likely caused by SVs. This signature is called *Read Depth (RD)* [1, 8]. If only a part of a read is mapped to some position in the reference genome and the rest is mapped to other position, again it is likely caused by SVs. This signature is called *Split Read (SR)* [87]. If the depth of coverage is high enough, the sequences around SVs is ultimately determined by assembling read sequences, a signature called *Sequence Assembly (AS)*. In spite of various methods that have been proposed, SV detection is still a difficult task. This is because it requires analysis of hidden complex structures involved in an enormous number of alignments of paired reads with the reference genome, and is because read sequences and alignments include unavoidable errors. Therefore, for example, a false detection rate (FDR) up to 10% had to be tolerated even when determining just the existence of each SV in the 1000 Genomes Project [79]. It is obviously more difficult to accurately detect the exact positions of SVs. Nevertheless, high-resolution SV calls are necessary to elucidate the functional impact of SVs and molecular mechanisms that generate SVs [35]. Moreover, to conduct a large-scale analysis, SV detection methods for data with a low depth of coverage (hereafter simply referred to as *coverage*) are desirable, because whole genome sequencing is expensive even with NGS technologies. Most of SV detection methods involve a step where paired reads obtained by NGS are mapped to the reference genome. A number of methods have been proposed for this task [24]. However, mapping still remains one of the most computationally intensive task in the analysis of SVs.

The third generation sequencing [14] is now available and is still being updated. In the future, completely different methods of sequencing, called *nanopore* technologies [3], might drastically change the field of sequencing. These new technologies can determine DNA sequences of thousands of contiguous bases at once. With this ability, it would be easy to determine genome sequences by *de novo* assembly. To exploit sequences obtained in this way for detecting SVs, we have to compare the obtained sequences with each other and exhaustively detect their

[3]`http://www.ncbi.nlm.nih.gov/projects/genome/assembly/grc/human/`

common segments and differences. However, it is computationally hard to solve even a simple problem where the smallest set of common substrings of which given two strings are concatenations is searched for [48].

1.2 Contribution of This Thesis

This thesis contributes to wide range of computational methods based on NGS sequences for the analysis of genomes affected by SVs, from inferring global structure of chromosomes using SVs to mapping NGS sequences to the reference genome. For the analysis of genome sequences obtained by *de novo* assembly, this thesis also proposes a method that compares multiple genome sequences and detects SVs by identifying shared regions as well as regions specific to each sequences.

1.2.1 Inferring Chromosome Structures from SVs Detected by NGS

The problem of inferring global structure of chromosomes from SVs was previously addressed [66], although the computational complexity of the problem was not deeply considered. Accordingly, it does not provide an theoretical basis on hardness of the problem and how to develop efficient algorithm. In this thesis, we propose the *chromosome problem (ChrP)* which searches for an optimal set of chromosomes that are the most consistent with given SVs as well as the estimated number and length of chromosomes. ChrP is formulated as an optimization problem on a bidirected graph constructed from SVs, which we term as a *chromosomes graph*. We also show that the problem is NP-complete by proving that the size of any solution is bounded by a polynomial expression of the number of vertices and edges of the chromosome graph, and that the well-known Hamiltonian Cycle problem can be reduced to ChrP. Further, a biologically meaningful restriction of the problem, which is termed as the *weakly connected constraint (WCC)*, is proposed. For each maximal subgraph in the chromosome graph that consists of edges corresponding to genomic segments whose copy number is at least one and edges corresponding to confirmed adjacencies of genomic segments, WCC requires that the subgraph include at least one end of a chromosome in the genome to be analyzed. We propose an algorithm that solves a variation of ChrP that can be solved in polynomial time. The algorithm was evaluated in computational experiments and it was confirmed that the algorithm reduced errors in copy numbers or false positive translocations in simulated SV data.

To analyze the effect of SVs on biological functions in detail and to infer the mechanism that caused detected SVs, the positions of SVs have to be determined as accurately as possible [35]. Mainly for methods that utilize the RP signature to detect SVs, we propose a method ChopSticks that improves the resolution of homozygous deletions. Methods based on the RP signature usually consider only *discordant pairs*, which are paired reads whose mapping distances and/or strands are not as expected. In order to obtain the improved estimates of positions of SVs, ChopSticks exploits *concordant pairs*, which are paired reads whose mapping distances and strands are as expected. A theoretical analysis revealed that by using concordant pairs the resolution can be improved to the extent similar to methods based on the RP signature with twice as many NGS sequences. In computational experiments using simulation data, ChopSticks performed well not only with methods base on the RP signature but also with a method base on the RD signature. In experiments using real NGS sequences obtained from inbred mice[4], ChopSticks also successfully improved the positions of SVs detected by methods based on the RP signature.

Detection of SVs involves a step that maps NGS sequences to the reference genome. Toward performance improvement in the future, we used a many-core processor Xeon Phi to conduct mapping by using 60 processing cores. To obtain mapping results that have already been well evaluated, we ported two famous mapping problem, Burrows-Wheeler Aligner (BWA) [43] and Bowtie2 [38], to Xeon Phi. Because both programs accelerate mapping by using vector operations implemented only in x86 processors, we ported tens of vector operations to Xeon Phi. In a computational experiment, the performance of both BWA and Bowtie2 was improved as the number of threads increased thanks to 60 cores implemented in Xeon Phi. Although mapping by Xeon Phi is still slower than that by CPU, our results is promising for performance improvement in the future.

1.2.2 Mutual Comparison of Genome Sequences

To detect SVs by comparing genome sequences with each other, we need an efficient method that identifies common and distinct regions in each genome. Because it is computationally hard to obtain an optimal solution if the problem is formulated as an optimization problem [65], we took the following approach. First, a class of substrings was defined as one that can be determined by comparing multiple strings with each other. These substrings were termed as *disjoint common substrings (DCS's).* Second, DCS's are detected by a linear-time algorithm based on suffix trees. In a computational experiment, the algorithm was shown to be completed in time linear to the total length of given strings and performs well for randomly generated DNA sequences.

1.3 Organization of This Thesis

The remaining part of this thesis is organized as follows. In Chapter 2, we formulate the problem of inferring global structures of chromosomes from SVs. We also prove that the problem is NP-complete and propose its restricted version solvable in polynomial time. In Chapter 3, we propose a new method ChopSticks that improves the resolution of SVs detected by other SV detection methods. In Chapter 4, we explain how genome mapping can be performed in a many-core processor Xeon Phi. In Chapter 5, we address the problem of comparing multiple strings and detect their common and distinct substrings. Finally, in Chapter 6, we conclude this thesis.

[4]`http://jaxmice.jax.org/type/inbred/index.html`

Chapter 2

Inferring Global Structures of Chromosomes

When genomic rearrangements are complex, enumerating only individual SVs is insufficient for elucidating the *global structure of chromosomes*, i.e., how the segments in a reference genome are extracted and ordered in an unknown target genome.

We address the problem of inferring the global structure of chromosomes based on *SV data*, which refer to aberrant adjacencies of genomic regions and copy number variations (CNVs) in this thesis. By solving this problem, we can determine the order of the genomic regions in the target genome. This order affects the structure of proteins if the genomic regions contain coding regions, and regulation of genes if the genomic regions include promoters or enhancers. In addition, raw SV data could be corrected by inferring the global structure of chromosomes because an optimal global structure would ignore false positive detection of aberrant adjacencies or correct wrongly estimated copy numbers. The task of inferring chromosomes is formulated as an optimization problem on a graph, which we term as a *chromosome graph.* Our contributions are summarized as follows:

- To infer the global structure of chromosomes, we formulate a computational problem that takes into account the number and length of chromosomes, as well as aberrant adjacencies and CNVs caused by genomic rearrangements. By taking SV data as the input, relatively low-depth NGS sequencing can be used.
- We prove that the problem is NP-complete.
- We propose a biologically meaningful restriction that makes the problem solvable in polynomial time. We also present an algorithm that solves the restricted problem.

Contents of this chapter are mainly from published work in [85].

2.1 Related Works

2.1.1 Copy Number and Adjacency Genome Reconstruction Problem

Oesper et al. [66] presented a pioneering work that aimed to infer the global structure of chromosomes from SV data. They formulated the *copy number and adjacency genome reconstruction problem.* Their formulation is based on graphs that they termed *interval-adjacency graphs.* These graphs are essentially the same as our chromosome graphs, except that we used bidirected graphs [16, 56] while they used alternating paths to exclude paths on the graph that do not correspond to chromosomes. They also implemented an efficient algorithm called *paired-end reconstruction of genome organization (PREGO)* to solve their problem and obtained promising results. Our work includes the following results that were not addressed by Oesper et al. First, we present a formulation that takes into account the number and length of chromosomes determined experimentally. Second, we prove that the problem is NP-complete. Finally, we propose a variation of the problem that can be solved in polynomial time.

2.1.2 *de novo* assembly

The aim of *de novo* sequence assembly is to reconstruct target genomes from raw NGS sequences [9, 19, 46, 56, 61, 67, 74, 88]. It includes a step to order fragments of genomes obtained by assembling NGS sequences. The step is usually implemented as an optimization problem, involving searching for paths on a graph that cover all vertices or all edges corresponding to substrings of genome sequences [56, 61]. By contrast, we allow some vertices and edges to be ignored because some portions of the reference genome might not appear in the target genome.

2.1.3 Reference Assisted Assembly

Reference-assisted assembly [28], also known as comparative assembly [68], aims at ordering segments of an unknown target genome by using known genomes of other organisms. By contrast, we order segments so that the chromosomes in the solution are most consistent with the SV data and the experimentally determined number and length of chromosomes.

2.1.4 Permutations of Integers

Methods based on permutations of integers [18] compare two genomes represented by two sequences of integers corresponding to genes or markers in the genome. Instead of using such sequences, we exploit SV data.

2.1.5 Cycle Optimization Problems

As explained in Section 2.5, we reduce the problem of inferring the global structure of chromosomes to the problem of finding an optimal cycle on a bidirected graph. We mention two problems that searches for an optimal cycle on a graph, although the cycle to be searched for in these problems is different from that in our problem.

The *Rural Postman Problem (RPP)* is a well-known classical problem on graphs. Let $G = (V, E)$ be a graph that is either undirected, directed, or bidirected, where V is a set of vertices and E is a set of edges. We assign a non-negative integer value $|e|$ to each edge $e \in E$. Let r be a cycle on G. We define a cost function $w(r)$ as follows:

$$w(r) \quad = \sum_{m(r,e)>0} |e| m(r, e).$$

Definition 2.1 (Rural Postman Problem (RPP))**.** *Suppose that we are given a graph $G = (V, E)$ and a subset E' of edges E. Then, find a cycle r with the smallest $w(r)$ from all cycles that pass each edge in E' at least once.*

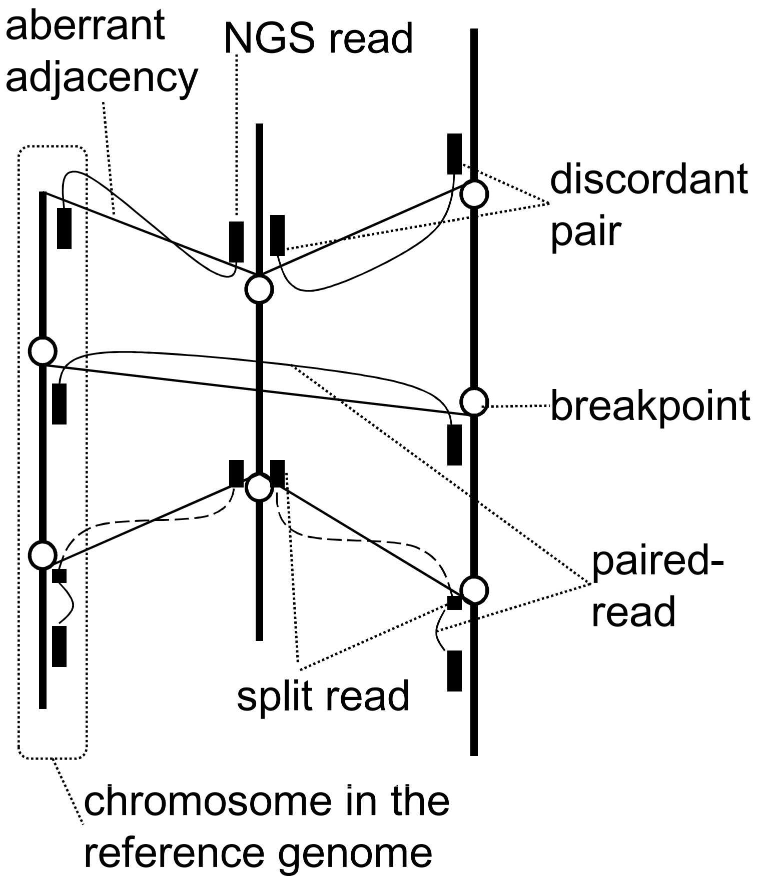


Figure 2.1: Thick vertical lines represent chromosomes in the reference genome, circles represent breakpoints, small black boxes represent NGS reads, solid curved lines represent paired reads, dashed curved lines represent split reads, and thin solid oblique lines represent aberrant adjacencies. Aberrant adjacencies are detected by using two types of NGS reads abnormally mapped to the reference genome: discordant pairs (three pairs from above), and split reads (two pairs from below).

Edges in E' are called *required edges*. RPP for undirected and directed graphs is NP-complete [41]. If a cycle finding problem includes RPP as a subproblem, the problem is NP-hard.

The Chinese Postman Problem (CPP) is also a well-known problem on graphs. CPP is a special case of RPP such that $E' = E$. Unlike RPP, CPP can be solved in polynomial time for directed and undirected graphs [2, 80]. Medvedev and Brudno [56] proposed an algorithm that solve CPP for bidirected graphs in $O(|E|^2 \log(|V|) \log(|E|))$ time, which is achieved by exploiting the Gabow's algorithm that solves the minimum cost bidirected flow problem [56, 16]. We borrow the idea of using bidirected graphs and Gabow's algorithm to solve our problem. Kundeti et al. [33] also proposed an algorithm that solves CPP for bidirected graphs. Their algorithm is efficient for typical instances. However, their algorithm is inefficient for general bidirected graphs.

2.2 Assumed Experimental Data

We assume the following experimental data as input.

2.2.1 Aberrant Adjacencies

In the target genome, distant segments in the reference genome may be adjacent because of rearrangements (Figure 2.1). Such aberrant adjacencies are detected by using NGS technologies as follows. First, NGS technologies can generate read pairs that are a few hundred bases apart from each other in the target genome. If two reads of a pair are not mapped to the reference genome with the expected orientations and mapped distance, the pair is called a *discordant pair* and is likely to be caused by SVs [10, 25, 70]. Second, if the alignment of a read and reference genome is split into more than one portion, such a split read also indicates a rearrangement [87]. A *breakpoint* is a position at a boundary of a rearrangement. Here, we ignore small differences between the real breakpoints and their estimations.

2.2.2 Copy Numbers

The number of occurrences of a subsequence in the reference genome may change because of rearrangements. This phenomenon results in *copy number variations (CNVs)*. Traditionally, CNVs have been analyzed by using DNA microarrays [55]. Several recent methods detect CNVs by finding changes in the depth of coverage of NGS sequences [1, 8]. Although tumor samples are usually a mixture of normal cells and various tumor cells, the copy numbers of a cancer cell can still be estimated by single-cell analysis [63]. In this thesis, for the sake of conciseness, the boundaries of CNVs are also called *breakpoints*.

2.2.3 Number of Chromosomes and Truncations

Identifying chromosomes and finding aberrant chromosomes by microscopy is an important part of clinical diagnostics [26]. The number of chromosomes, denoted by n_N in this thesis, is available after inspection. Throughout this thesis, we assume that $n_N \geq 1$. In addition, we also take into account the number of chromosomal truncations, which we denote as n_T. Chromosomal truncations are detected as a decrease in copy numbers without aberrant adjacencies. We consider n_N and n_T to improve the inference of the global structure of chromosomes from SV data.

2.2.4 Chromosome Length

The length of chromosomes can be estimated experimentally from flow karyotyping, and, approximately, from microscopic images [27]. Here, the estimated length is denoted by λ_i for $1 \leq i \leq N_L$, where $N_L(\geq n_N)$ is the maximum possible number of chromosomes.

2.3 Problem Definition

Any instance of our problem is modeled as a graph that we term a *chromosome graph*. The graph contains elements derived from the reference genome and experimental data. Each vertex corresponds to a location in the reference genome. In addition, each edge corresponds to either a segment in the reference genome, an adjacency of flanking segments in the reference genome, or an aberrant adjacency in the target genome caused by rearrangements.

We assume that the target genome is a set of chromosomes, each of which is a concatenation of segments in the reference genome. Each chromosome in the

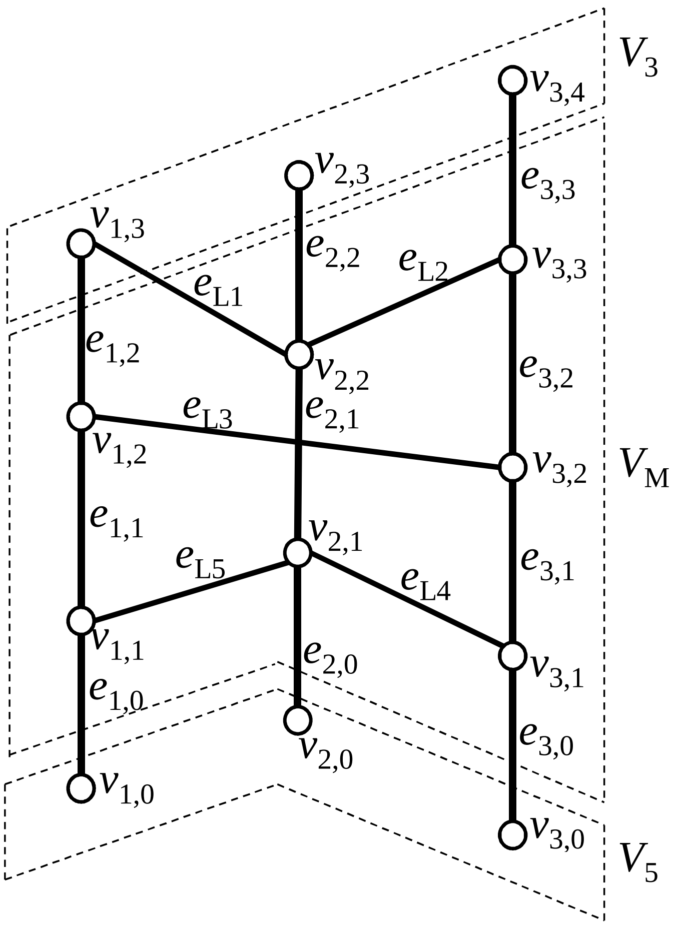


Figure 2.2: Thick vertical edges represent edges in E_S that correspond to segments in the reference genome, oblique edges represent edges in E_L that correspond to aberrant adjacencies. Vertices surrounded by dashed lines belong to V_5, V_M, and V_3, read from the bottom of the graph to top.

target genome is represented as a path on the graph, and these paths explain how segments in the reference genome are incorporated into the target genome. The goodness of the estimated target genome is measured by a cost function, and we search for an optimal set of chromosomes that minimizes this cost function.

We first define a graph that contains some of elements described above. Then, we extend the graph to a chromosome graph. Finally, we present the formal definition of the problem.

2.3.1 Prototype chromosome graph

We first construct an undirected graph called a *prototype chromosome graph*, $G = (V, E)$ (Figure 2.2). Let N_C be the number of chromosomes of the reference genome and n_i be the number of breakpoints in the i-th chromosome of the reference genome. Then, V contains the following vertices.

- Vertices corresponding to breakpoints:

$$V_M = \{v_{i,j} | 1 \le i \le N_C, 1 \le j \le n_i\}.$$

- Vertices corresponding to the beginning of chromosomes in the reference

genome:
$$V_5 = \{v_{i,0} | 1 \le i \le N_C\}.$$

- Vertices corresponding to the end of chromosomes in the reference genome:
$$V_3 = \{v_{i,n_{i+1}} | 1 \le i \le N_C\}.$$

Then, we define $V = V_5 \cup V_3 \cup V_M$.

Next, we define a set of edges, E. We make the following two types of edges.

- Edges corresponding to segments between two breakpoints that are next to each other in the reference genome. For each $1 \le i \le N_C$ and $0 \le j \le n_i$, we make an edge $e_{i,j} = (v_{i,j}, v_{i,j+1})$.
- Edges corresponding to aberrant adjacency of two segments in the reference genome. Let N_A be the number of detected aberrant adjacencies. For the k-th aberrant adjacency ($1 \le k \le N_A$) that links positions corresponding to v_{i_1,j_1} and v_{i_2,j_2}, we make an edge $e_{Lk} = (v_{i_1,j_1}, v_{i_2,j_2})$.

Then, we define
$$\begin{aligned} E_S &= \{e_{i,j} | 1 \le i \le N_C, 0 \le j \le n_i\}, \\ E_L &= \{e_{Lk} | 1 \le k \le N_A\}, \\ E &= E_S \cup E_L. \end{aligned}$$

2.3.2 Chromosome graph

In a prototype chromosome graph, a path might visit two edges in E_L contiguously. Such a path does not correspond to a real chromosome. To exclude such a path we use a technique similar to that of Oesper et al. [66]. Although Oesper et al. [66] used alternating paths, their formulation can be represented by using a bidirected graph whose edges have directions at both ends [56, 60]. We directly define our graph by using a bidirected graph (Figure 2.3). Let $d(e, v) \in \{+, -\}$ be the direction of an edge e at a vertex v, and $-d(e, v)$ be the opposite direction of $d(e, v)$.

- Each vertex $v_{i,j} \in V_M$ is split into two vertices $v^+_{i,j}$ and $v^-_{i,j}$. The set V_M is redefined as
$$V_M = \{v^-_{i,j}, v^+_{i,j} | 1 \le i \le N_C, 1 \le j \le n_i\}.$$
Vertices in V_5 and V_3 are renamed so that
$$\begin{aligned} V_5 &= \{v^-_{i,0} | 1 \le i \le N_C\}, \\ V_3 &= \{v^+_{i,n_{i+1}} | 1 \le i \le N_C\}. \end{aligned}$$
- An edge $e_{i,j} = (v_{i,j}, v_{i,j+1}) \in E_S$ is reconnected to $v^-_{i,j}$ and $v^+_{i,j+1}$. In addition, $d(e_{i,j}, v^-_{i,j}) = -$ and $d(e_{i,j}, v^+_{i,j+1}) = +$.
- Let $e \in E_L$ be an edge connected to $v_{i,j}$ in the prototype chromosome graph. If e corresponds to an aberrant adjacency involving the segment that stretches toward $v_{i,j+1}$, e is reconnected to $v^-_{i,j}$ and $d(e, v^-_{i,j})$ is set to '+'. Otherwise, e is reconnected to $v^+_{i,j}$ and $d(e, v^+_{i,j})$ is set to '−'.
- We add the following set of new edges:
$$E_R = \{\hat{e}_{i,j} = (v^+_{i,j}, v^-_{i,j}) | 1 \le i \le N_C, 1 \le i \le n_i\}.$$
Directions are set so that $d(\hat{e}_{i,j}, v^+_{i,j}) = -$ and $d(\hat{e}_{i,j}, v^-_{i,j}) = +$.

The modified graph represents a *chromosome graph*.

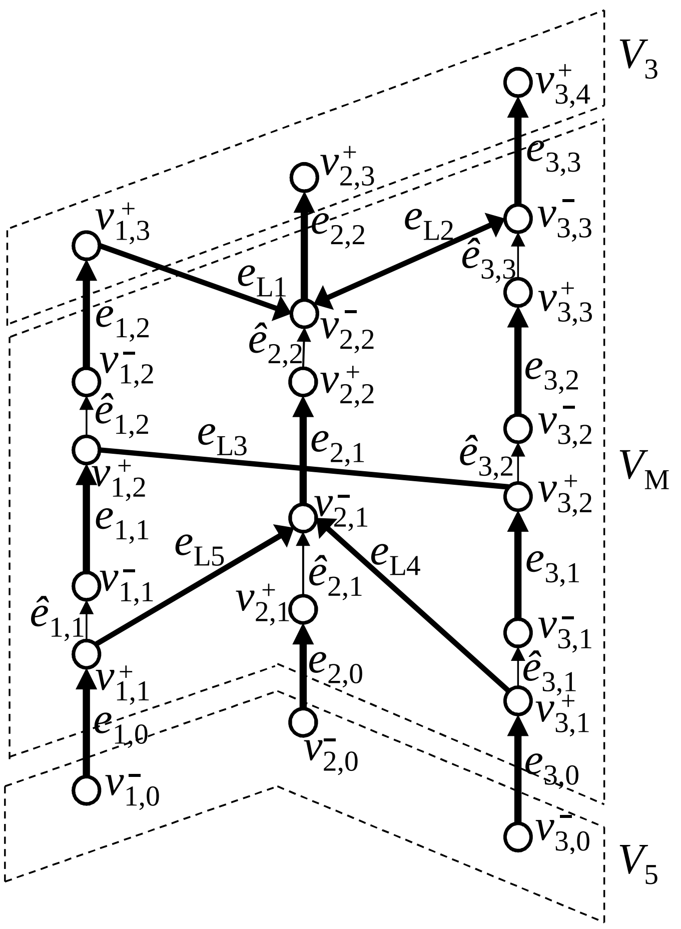


Figure 2.3: Thin vertical edges represent edges in E_R. Arrowheads represent the '+'-direction, whereas ends of edges without arrowheads represent '$-$'-direction.

2.3.3 Paths and Chromosomes

A *path* $c = v_1e_1v_2e_2v_3 \dots e_lv_{l+1}$ on a chromosome graph G is an alternating sequence of vertices and edges, which has the following properties:

- The first and the last of c are vertices.
- Any subsequence of the form $e_kv_ke_{k+1}$ $(1 \leq k \leq l)$ means that $d(e_k, v_k) = -d(e_{k+1}, v_k)$.

A path c is said to *visit* an edge e if c contains e. Similarly, c is said to *visit* a vertex v if c contains v. When a path is written as a sequence of vertices and edges, for simplicity, we omit the notation of the vertices if they are clear. Let $C = \{c_1, c_2, \dots, c_{|C|}\}$ be a multi-set of paths on G. We define C as a multi-set so that more than one identical path can exist. In addition, let $m(c, e)$ be the number of times c visits an edge e, and $m(C, e) = \sum_{c_i \in C} m(c_i, e)$. A *cycle* is a path whose first and last vertices are identical and the directions of the first and the last edges at the vertex are opposite. A *chromosome* on G is a path whose first and last edges are both in E_S.

2.3.4 Copy Numbers and Lengths

Two integers are assigned to each $e \in E$. First, $n(e)$ for $e \in E_S$ represents an experimentally estimated copy number of the corresponding segment in the reference genome. Second, $|e|$ for $e \in E_S$ represents the length of the corresponding segment in the reference genome. For $e \in E_L \cup E_R$, we set $n(e)$ and $|e|$ to 0. The length of a path c is defined as $|c| = \sum_{e \in E} |e| m(c, e)$. To simply describe all properties of e together, we use the following notation:

$$e = \langle d(e, v_1)v_1, d(e, v_2)v_2, n(e), |e| \rangle.$$

2.3.5 Upper Bound on Parameters

Campbell et al. [8] presented examples of amplified regions in cancer cells. The copy numbers were less than 100 in these regions. Therefore, we assume that the copy numbers are in at most hundreds. We also assume that short repeat elements are masked in advance in order to exclude segments that appear spuriously. Based on the details given above, we assume that n_N, n_T, and $n(e)$ for $e \in E_S$ are all less than a fixed constant U. The value of U does not have to be determined because U is only used in the analysis of computational complexity.

2.3.6 Formulation of the Problem

To find an optimal set of chromosomes, we define an optimization problem over a chromosome graph. We define a cost function to be used as a target function of the optimization problem. This function imposes costs on the number of chromosomes, the number of chromosomal truncations, and the number of visits to edges, penalizing for deviations from those that are experimentally expected.

Let $C = \{c_1, c_2, \dots, c_{|C|}\}$ be a multi-set of chromosomes on G, and $w_N(C)$ be the cost of the difference between n_N and $|C|$. Also let $\mathrm{Tr}(C)$ be the number of ends of chromosomes in V_M, and $w_T(C)$ be the cost of the difference between n_T and $\mathrm{Tr}(C)$. In addition, $w(e, x)$ for $e \in E_S$ is defined as the cost when e is visited x-times. For $e \in E_L \cup E_R$, $w(e, x)$ is set to 0.

We assume that $w_N(C)$, $w_T(C)$, and $w(e, x)$ for $e \in E_S$ monotonically increase as $||C| - n_N|$, $|\mathrm{Tr}(C) - n_N|$, and $|x - n(e)|$ increase, respectively. Then, we define the cost function $W(C)$ as follows:

$$W(C) = w_N(C) + w_T(C) + \sum_{e \in E} w\left(e, m(C, e)\right). \tag{2.1}$$

We assume that each term is 0 if and only if

$$\left.\begin{array}{rcl} |C| & = & n_N, \\ \mathrm{Tr}(C) & = & n_T, \\ m(C, e) & = & n(e) \text{ for } e \in E_S. \end{array}\right\} \tag{2.2}$$

With these notations, we formulate the problem of inferring the global structure of chromosomes as follows:

Definition 2.2 (Chromosome problem (ChrP))**.** *Suppose that we are given a chromosome graph $G = (V, E)$, a cost function $W(C)$, and parameters λ_i ($1 \leq i \leq N_L$), where N_L is the maximum possible number of chromosomes. Then, find a multi-set of chromosomes C on G that minimizes $W(C)$ under the constraint that $|c_i| \leq \lambda_i$ for $c_i \in C$.*

2.4 NP-completeness of ChrP

Although a similar problem was proposed previously [66], its computational complexity was not analyzed. We show that ChrP is computationally hard.

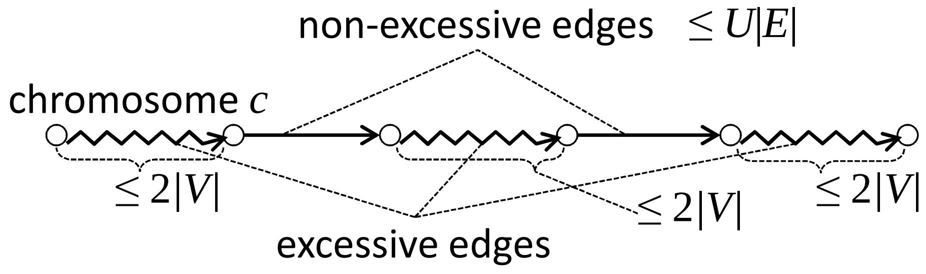


Figure 2.4: Straight arrows represent non-excessive edges, while jagged lines represent sequences of excessive edges.

Theorem 2.1. *ChrP is NP-complete.*

To show that Theorem 2.1 holds, we first present an upper bound on the size of an optimal solution of ChrP to show that ChrP is in NP. Then, we prove that ChrP is NP-hard.

Lemma 2.1. *Let $G = (V, E)$ be a chromosome graph. Also, let C be a multi-set of chromosomes on G that minimizes $W(C)$ such that $|c_i| \leq \lambda_i$ for $c_i \in C$. Then, C has at most $U(4|V| + 1)(|E| + 1)$ edges.*

Proof. Let $c \in C$ be a chromosome in C. We define an edge e in c as *non-excessive* if $e \in E_S$ and $m(C, e) \leq n(e)$, and *excessive* otherwise. Let t_c be the number of non-excessive edges visited by c. If $t_c > 0$, c can be written as $c = p_1e_1p_2e_2 \ldots e_{t_c}p_{t_c+1}$, where e_k $(1 \leq k \leq t_c)$ is a non-excessive edge and p_k $(1 \leq k \leq t_c{+}1)$ is a possibly empty path that contains only excessive edges (Figure 2.4). If p_k contains a cycle as its subpath, the cycle can be removed to decrease $W(C)$, a contradiction. Accordingly, p_k does not contain a cycle. This implies that p_k visits at most $2|V|$ vertices and, thus, $2|V|$ edges. Therefore, at most, $4|V|$ excessive edges are visited for each non-excessive edge. Note that a non-excessive edge e can be visited, at most, $n(e)$-times. Therefore, $\sum_{c \in C} t_c \leq \sum_{e \in E_S} n(e)$.

Chromosomes such that $t_c = 0$ can exist only if they contribute to the decrease of the first or the second term of $W(C)$ defined by (2.1). Accordingly, the number of such chromosomes is, at most, $n_N + n_T$. In addition, a chromosome c, such that $t_c = 0$, does not contain any cycles because such a cycle can be removed to decrease $W(C)$. Therefore, at most, c visits $2|V|$ vertices and, thus, $2|V|$ edges.

Consequently, C contains, at most, $2|V|(n_N + n_T) + (4|V| + 1)\sum_{e \in E_S} n(e) \leq U(4|V| + 1)(|E| + 1)$ edges. □

Lemma 2.2. *The problem ChrP is in NP.*

Proof. Once an optimal solution C is given, whether or not $W(C)$ is greater than a given constant can be determined in $O(|V||E|)$ time by Lemma 2.1. □

Lemma 2.3. *The problem ChrP is NP-hard.*

Proof. The *Hamiltonian Cycle problem (HC)* is a problem of finding a cycle that visits each vertex of a graph exactly once, and is a well-known NP-complete problem [17]. Here, we reduce HC to ChrP. Consider HC on a directed graph

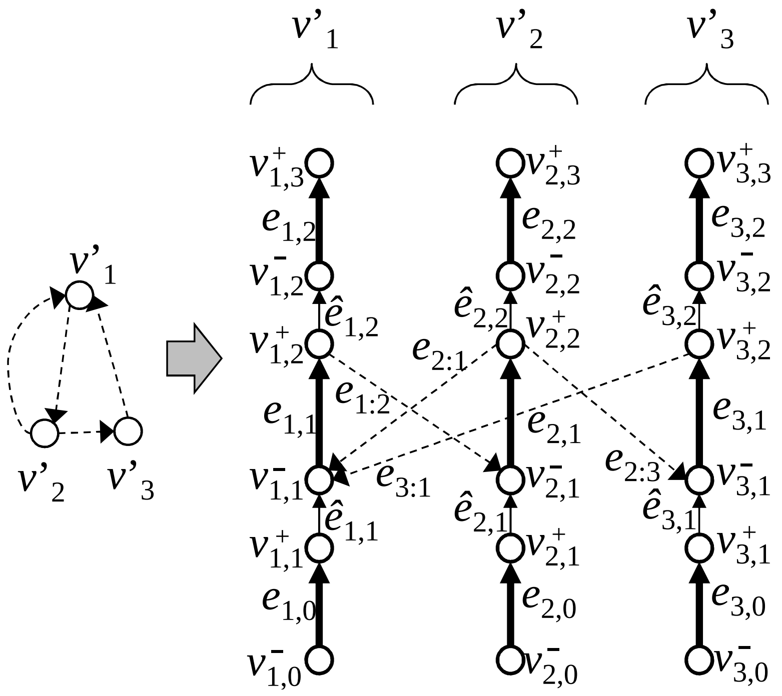


Figure 2.5: In this graph, solid edges are constructed for each vertex in a graph H of HC, whereas dashed edges correspond to edges in H.

$H = (V', E')$, where $V' = \{v'_1, v'_2, \ldots, v'_{|V'|}\}$ is a set of vertices and E' is a set of edges. We construct a chromosome graph $G = (V, E)$ from H (Figure 2.5), where

$$V = \bigcup_{1 \le i \le |V'|} \{v^-_{i,0}, v^+_{i,1}, v^-_{i,1}, v^+_{i,2}, v^-_{i,2}, v^+_{i,3}\}$$

is a set of vertices, and $E = E_S \cup E_L \cup E_R$ is a set of edges. Here, E_S consists of

$$\begin{aligned}
e_{1,0} &= \langle -v^-_{1,0}, +v^+_{1,1}, 1, 1\rangle, \\
e_{1,1} &= \langle -v^-_{1,1}, +v^+_{1,2}, 2, 1\rangle, \\
e_{1,2} &= \langle -v^-_{1,2}, +v^+_{1,3}, 1, 1\rangle, \\
e_{i,0} &= \langle -v^-_{i,0}, +v^+_{i,1}, 0, 1\rangle \quad (2 \le i \le |V'|), \\
e_{i,1} &= \langle -v^-_{i,1}, +v^+_{i,2}, 1, 1\rangle \quad (2 \le i \le |V'|), \\
e_{i,2} &= \langle -v^-_{i,2}, +v^+_{i,3}, 0, 1\rangle \quad (2 \le i \le |V'|).
\end{aligned}$$

E_R consists of

$$\begin{aligned}
\hat{e}_{i,1} &= \langle -v^+_{i,1}, +v^-_{i,1}, 0, 0\rangle \quad (1 \le i \le |V'|), \\
\hat{e}_{i,2} &= \langle -v^+_{i,2}, +v^-_{i,2}, 0, 0\rangle \quad (1 \le i \le |V'|).
\end{aligned}$$

E_L consists of

$$e_{i':i} = \langle -v^+_{i',2}, +v^-_{i,1}, 0, 0\rangle \quad ((v'_{i'}, v'_i) \in E').$$

In addition, we set $n_N = 1$, $n_T = 0$, and $\lambda_i = |V'| + 3$ for any i. Then, we prove that H has a Hamiltonian cycle if, and only if, ChrP on G has a solution C such that $W(C) = 0$. Suppose that h is a Hamiltonian cycle on H. Let c be a chromosome that begins with $e_{1,0}\hat{e}_{1,1}e_{1,1}$ and then visits $e_{i':i}e_{i,1}$ in the order that edges $(v_{i'}, v_i)$ appear in h from $i' = 1$, and finally ends with $e_{1,1}\hat{e}_{1,2}e_{1,2}$. Then, a set of a single chromosome $C = \{c\}$ satisfies $W(C) = 0$ and $|c| = |V'| + 3 \le \lambda_1$.

Conversely, let C be a solution of ChrP that satisfies $W(C) = 0$. Because (2.2) holds, $|C| = 1$, $\mathrm{Tr}(C) = 0$, and $m(C, e) = n(e)$. Let c be the only chromosome in C. Because $n(e_{1,1}) = 2$ and $n(e_{i,1}) = 1$ for $2 \leq i \leq |V'|$, a path that visits vertices $v'_i \in V'$ in the order that $e_{i,1}$ appears in c is a Hamiltonian cycle on H. □

Theorem 2.1 directly follows Lemma 2.2 and 2.3.

2.5 Polynomial-time Solvable Variation

We propose a variation of ChrP that is solvable in polynomial time. For $e \in E$, it is highly likely that $m(C, e) \geq 1$ if e is supported by a large number of paired reads. Therefore, it is worth considering a variation in which some edges must appear in the target genome. We refer to the edges as *required edges*. In addition, because chromosomal truncations can be detected, it is also worth considering a variation in which we know where the ends of the chromosomes of the target genome exist in the reference genome. Because the definition of $W(C)$ is abstract, we focus on a cost function such that

$$\left.\begin{array}{rcl} w_N(C) &=& Q_N||C| - n_N|, \\ w_T(C) &=& Q_T|\mathrm{Tr}(C) - n_T|, \\ w(e, x) &=& |e||x - n(e)|, \end{array}\right\} \tag{2.3}$$

where Q_N and Q_T are constants given as parameters. The values of Q_N and Q_T are tuned in advance so that known global structures of genomes are well reconstructed.

2.5.1 Weakly Connected Constraint

Let $G = (V, E)$ be a general bidirected graph. A subgraph g of G is a *weakly connected component* if g is a connected component when all directions are removed [77]. In addition, g is *maximal* if g is not a subgraph of a larger weakly connected component. For a subset E' of E, we define $\mathrm{CC}(G, E')$ as a set of maximal weakly connected components of a graph induced from G by removing the edges not in E'.

Definition 2.3 (Weakly connected constraint (WCC))**.** *Let $G = (V, E)$ be a chromosome graph. Also let V_W and E_W be subsets of V and E, respectively. Each $g \in CC(G, E_W)$ is* good *if g contains at least one vertex in V_W. Then, G satisfies the* weakly connected constraint (WCC) *if all $g \in CC(G, E_W)$ are good.*

We use WCC by setting V_W to a set of vertices that correspond to ends of chromosomes in the target genome, $E_W = \{e \in E | e \text{ is required}\}$. See Figure 2.6 for an example. An instance that satisfies WCC can be obtained as follows. First, V_W is obtained by finding the positions of chromosomal truncations, as well as the ends of the chromosomes of the reference genome that remain in the target genome. Because a chromosome that does not include detected ends can be in a solution, V_W does not need to contain all ends of chromosomes in the target genome. We assume that $n_T \geq |V_W|$. Next, if $g \in \mathrm{CC}(G, E_W)$ is not good, edges $e \in E$ on some path connecting g and good $g' \in \mathrm{CC}(G, E_W)$ are added to E_W. To do this, if possible, we experimentally confirm that e is required if $e \in E$. Finally, if some $g \in \mathrm{CC}(G, E_W)$ that are not good still remain, edges in g are forcibly removed from E_W by setting $n(e)$ to 0 if $e \in E_S$ or by changing e not required if $e \in E_L \cup E_R$.

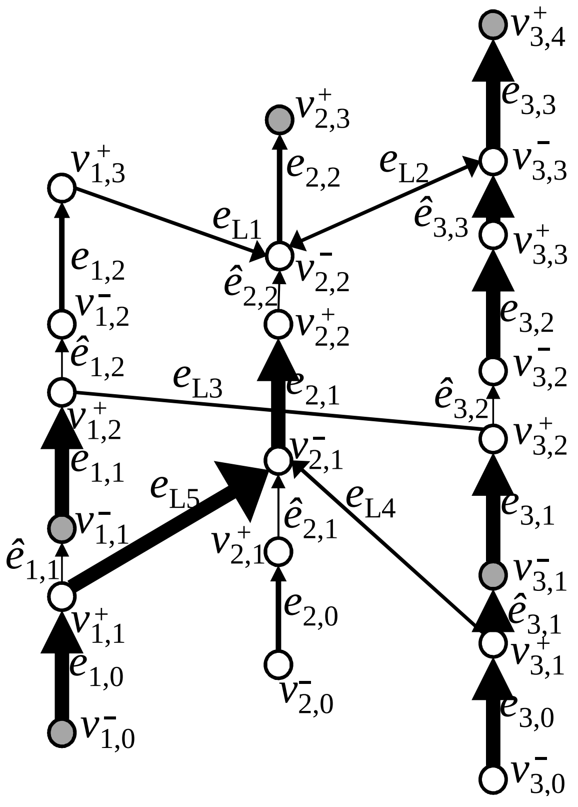


Figure 2.6: Gray circles are vertices in V_W and thick arrows are edges in E_W.

Definition 2.4 (Chromosome problem with WCC (ChrW))**.** *Let $G = (V, E)$ be a chromosome graph that satisfies WCC with respect to some $V_W \subset V$ and $E_W \subset E$. Then, find a set C of chromosomes on G such that each vertex in V_W is an end of some $c \in C$, $m(C, e) > 0$ for $e \in E_W$, and C minimizes $W(C)$ when (2.3) is satisfied.*

Theorem 2.2. *The problem ChrW can be solved in $O(|E|^2 \log |V| \log |E|)$ time.*

We show how ChrW can be solved in polynomial time in the rest of this section.

2.5.2 Circular Chromosome Graph

RPP and CPP are useful for the purpose of analyzing computational complexity of problems related to graphs and developing efficient algorithms. However, in ChrW we have to cope with multiple chromosomes on a chromosome graph and a cost imposed on the number of chromosomes, which are out of the scope of RPP and CPP. To bridge them, we construct graphs called *circular chromosome graphs.*

Definition 2.5 (Circular chromosome graph)**.** *Let $G = (V, E)$ be a chromosome graph, and let v_N and v_T be new vertices. In addition, let E_N be a set of the following edges: for $1 \le i \le N_C$,*

$$\begin{aligned}
e_t(v_{i,0}^-) &= \langle -v_N, +v_{i,0}^-, 0, 0\rangle, \\
e_t(v_{i,n_i}^+) &= \langle -v_N, -v_{i,n_i}^+, 0, 0\rangle, \\
e_t(v_{i,j}^+) &= \langle -v_T, -v_{i,j}^+, 0, 0\rangle \quad (1 \le j \le n_i), \\
e_t(v_{i,j}^-) &= \langle -v_T, +v_{i,j}^-, 0, 0\rangle \quad (1 \le j \le n_i),
\end{aligned}$$

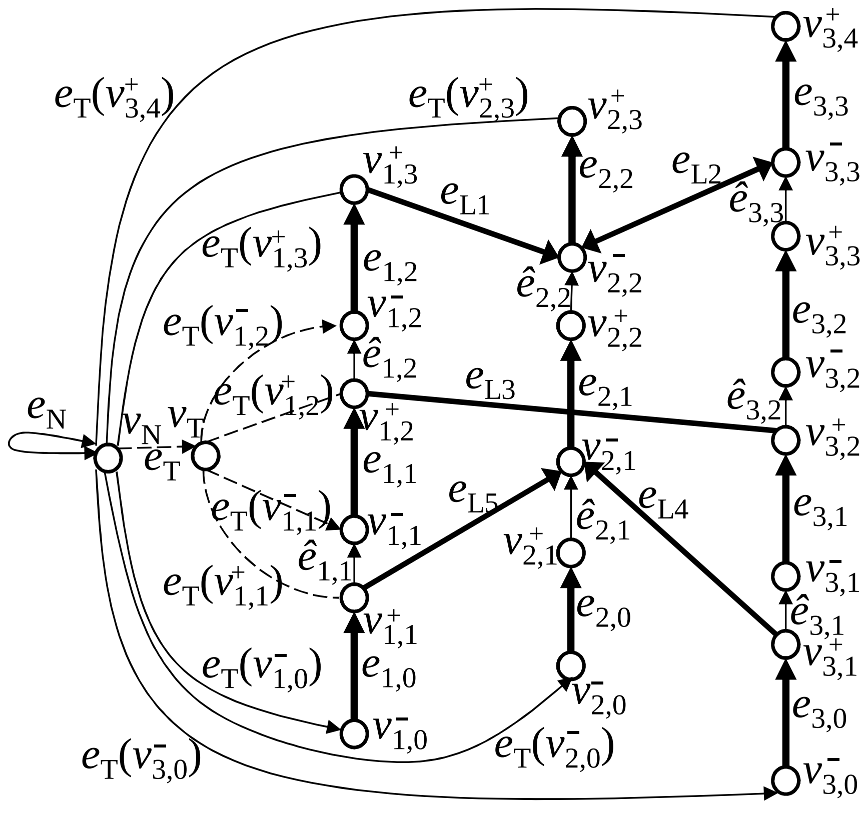


Figure 2.7: The problem of optimizing multiple chromosomes is converted to the problem of finding a cycle on this graph. For simplicity, we omitted $e_t(\cdot)$, except for the leftmost chromosome in the reference genome.

and

$$\begin{aligned} e_T &= \langle -v_N, +v_T, n_T, Q_T \rangle, \\ e_N &= \langle +v_N, +v_N, n_N, Q_N \rangle. \end{aligned}$$

The graph $\tilde{G} = (V \cup \{v_N, v_T\}, E \cup E_N)$ *is called a* circular chromosome graph.

See Figure 2.7 for an example.

2.5.3 Circulation on a Bidirected Graph

Let $G = (V, E)$ be a bidirected graph, and $a_{v,e}$ for $v \in V$ and $e \in E$ be an integer such that

$$a_{v,e} = \begin{cases} 2 & \text{if } e \text{ has two '+'-ends at } v, \\ 1 & \text{if } e \text{ has only one '+'-end at } v, \\ -1 & \text{if } e \text{ has only one '$-$'-end at } v, \\ -2 & \text{if } e \text{ has two '$-$'-ends at } v, \\ 0 & \text{if } e \text{ is not connected to } v. \end{cases}$$

Also let b_v be an integer defined for each $v \in V$, Z be the set of non-negative integers, and $l(e)$ and $u(e)$ be two non-negative integers assigned to each edge $e \in E$ called a *lower bound* and an *upper bound*, respectively. Unless otherwise specified, in this study $l(e) = 0$ and $u(e) = \infty$.

Definition 2.6. *A* bidirected flow (biflow) *[16, 56] is a mapping* $f : E \to Z$ *such that*

$$l(e) \le f(e) \le u(e) \qquad \textit{for each } e \in E, \tag{2.4}$$

$$\sum_{e \in E} a_{v,e} f(e) = b_v \qquad \textit{for each } v \in V. \tag{2.5}$$

The cost of f is defined as $w_G(f) = \sum_{e \in E} w(f, e)$, where $w(f, e)$ is a cost of f on $e \in E$. A biflow f is optimal *if f minimizes $w_G(f)$. A* circulation *is a biflow such that $b_v = 0$ for any $v \in V$.*

When $w(e, f) = |e|f(e)$, Gabow's algorithm [16] calculates an optimal biflow in $O(|E|^2 \log |V| \log(\max_{e \in E}\{u(e)\}))$ time. In order to relate a circulation to a cycle on a bidirected graph, we propose the following lemma.

Lemma 2.4. *Let f be a circulation of a bidirected graph $G = (V, E)$.*

1. *There exists a multi-set R of cycles on G such that*

$$m(R, e) = f(e) \textit{ for each } e \in E. \tag{2.6}$$

   Conversely, there exists a circulation f that satisfies (2.6) for any multi-set R of cycles on G.

2. *Let $h(G)$ be the number of connected components of G when all edges e such that $l(e) = 0$ are removed from G. Then, cycles in R can be merged into, at most, $h(G)$ cycles.*

3. *The merged cycles can be obtained in $O(\sum_{e \in E} f(e))$ time if f is given.*

Proof. First, we prove that R exists. Although a similar result is known as the *flow decomposition theorem* [2], we prove this for completeness and for illustrating the algorithm that calculates R. Consider a path r formed by the algorithm FindCycle shown in Figure 2.8. During the algorithm, (2.5) holds for all vertices except for u_0, u. Therefore, a new edge e' in Step 7 always exists until u_0 is reached. Because $|V|$ is finite, r eventually reaches u_0 and forms a cycle. If r reaches u_0 with an edge with '+'-end at u_0, there remains an edge e with '$-$'-end at u_0 such that $f(e) > 0$ because of (2.5). Accordingly, the FindCycle algorithm can continue. If r never reaches u_0 with an edge with '$-$'-end, it contradicts with (2.5).

By repeating the FindCycle algorithm until $f(e) = 0$ for all $e \in E$, we obtain a multi-set R of cycles such that $m(R, e) = f(e)$ for each $e \in E$. In addition, R can be obtained in $O(\sum_{e \in E} f(e))$ time. Conversely, let R be a set of cycles on G. Then, $f(e) = m(R, e)$ is clearly a circulation.

Next, let g be one of connected components generated by removing all edges e such that $l(e) = 0$ from G, and $R_g \subset R$ be a multi-set of cycles such that each $r \in R_g$ shares at least one edge with g. We show that the cycles in R_g can be merged into a single cycle when $|R_g| \geq 2$. Suppose that all vertices in some $r_1 \in R_g$ are never visited by other cycles in R_g. Then, g is not weakly connected because each edge in g must be in some cycle in R and therefore in R_g, a contradiction. Therefore, some vertex v in r_1 is also in another cycle $r_2 \in R_g$. Then, r_1 and r_2 can be merged into a single cycle $v r_1 v r_2 v$. By repeating the merging, all cycles in R_g can be merged into a single cycle.

All cycles in R sharing vertices can be merged in $O\left(\sum_{r \in R} \sum_{e \in E} m(r, e)\right) = O(\sum_{e \in E} f(e))$ time. Then, the number of cycles is at most $h(G)$ because there are only $h(G)$ connected components. □

```
1   $e$ := an edge that satisfies $f(e) > 0$
        and has at least one '+'-end.
2   $u_0$ := a vertex at a '+'-end of $e$.
3   $u$ := a vertex at another end of $e$.
4   $r := u_0 e u$.
5   repeat
6       $f(e) := f(e) - 1$.
7       $e'$ := an edge that satisfies $f(e') > 0$ and
            has a direction different from $e$ at $u$.
8       $u$ := another end of $e'$.
9       $r := r e' u$.
10      $e := e'$.
11  until $u = u_0$ and $e$ have '$-$'-end at $u$.
```

Figure 2.8: Algorithm FindCycle. This algorithm finds a cycle r that consists of edges e such that $f(e) > 0$, and changes $f(e)$ to $f(e) - m(r, e)$.

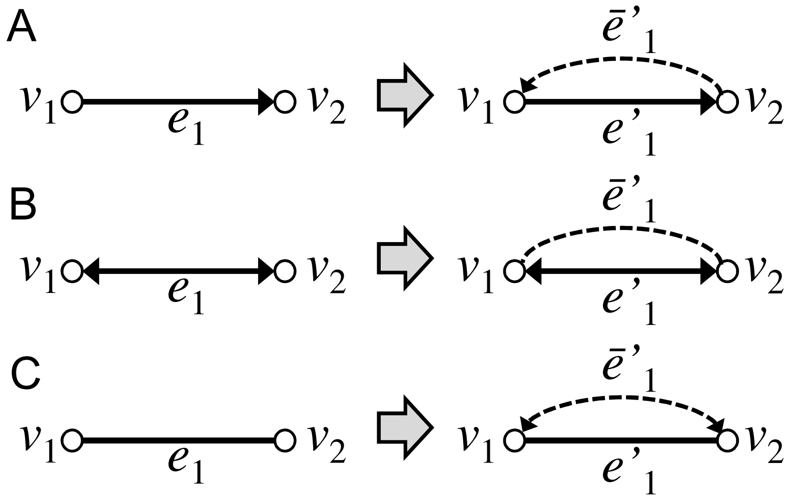


Figure 2.9: Schematic illustrations of dualized edges. A: A case where $d(e_1, v_1) = +$ and $d(e_1, v_2) = -$. B: $d(e_1, v_1) = d(e_1, v_2) = +$. C: $d(e_1, v_1) = d(e_1, v_2) = -$.

2.5.4 Cost Optimization for Each Edge

To optimize the cost defined by (2.3) using a flow on a circular chromosome graph, we use a method similar to one that calculates an optimal flow when each edge has convex cost functions [2]. Although the method [2] uses edges with negative lengths, our method presented here only uses edges with non-negative lengths. Let $G = (V, E)$ be a bidirected graph, $|e|$ and $n(e)$ be non-negative integers defined for $e \in E$, and e_1 be an edge in E. Also, let v_1 and v_2 be vertices at the ends of e_1. We define that to *dualize* e_1 is to replace e_1 with the following two edges

$$\begin{aligned} e'_1 &= \langle d(e_1, v_1)v_1, d(e_1, v_2)v_2, 0, |e_1|\rangle, \\ \bar{e}'_1 &= \langle -d(e_1, v_1)v_1, -d(e_1, v_2)v_2, 0, |e_1|\rangle, \end{aligned}$$

and to set $l(e'_1) = \max\{l(e_1), n(e_1)\}$ and $u(\bar{e}'_1) = \max\{0, n(e_1) - l(e_1)\}$. See Figure 2.9 for examples. The edge e'_1 is used to represent the flow of the same direction as e_1, while $\bar{e}_1'$ is used to penalize the decrease of the flow of e_1 from $n(e_1)$.

Lemma 2.5. *Let G be a bidirected graph, and f be a biflow on G whose cost is defined as follows:*

$$w_G(f) = \sum_{e \in E} |e||f(e) - n(e)|.$$

Also, let G' be the bidirected graph generated by dualizing some of edges $e \in E$, and f' be a biflow on G' whose cost is defined as follows:

$$w_{G'}(f') = \sum_{e \in E'} |e|f'(e), \tag{2.7}$$

where E' is a set of edges in G'. Then, an optimal biflow f on G that minimizes $w_G(f)$ can be calculated in $O(|E|)$ time, provided that an optimal biflow f' on G' that minimizes $w_{G'}(f')$ is given.

To prove Lemma 2.5, we use the following lemma.

Lemma 2.6. *Let $G = (V, E)$ be a bidirected graph, $e_1 \in E$ be an edge, $|e_1|$ and $n(e_1)$ be non-negative integers, and f be a biflow on G whose cost function is*

$$w_G(f) = \sum_{e \in E - \{e_1\}} w(f, e) + |e_1||f(e_1) - n(e_1)|, \tag{2.8}$$

where $w(f, e)$ is a cost of f on e. Also let $G' = (V, E')$ be the bidirected graph generated by dualizing an edge e_1 in G, where $E' = E \cup \{e_1', \bar{e_1}'\} - e_1$. In addition, let f' be a biflow on G' whose cost function is

$$w_{G'}(f') = \sum_{e \in E - \{e_1\}} w(f', e) + |e_1|f(e_1') + |e_1|f(e_1''). \tag{2.9}$$

Then, an optimal biflow f on G that minimizes $w_G(f)$ can be calculated in $O(1)$ time, provided that an optimal biflow f' on G' that minimizes $w_{G'}(f')$ is given.

Proof. First, we show that for any f', there exists a biflow f such that

$$w_G(f) = w_{G'}(f') - |e_1|n(e_1) \tag{2.10}$$

and let Z be a set of non-negative integers. Consider the following mapping $f : E \to Z$:

$$\begin{aligned} f(e) &= f'(e) \quad \text{for } e \neq e_1, & (2.11)\\ f(e_1) &= f'(e_1') - f'(\bar{e}_1'). & (2.12)\end{aligned}$$

We prove that f is a biflow on G because (2.4) and (2.5) holds. If v is not a vertex at an end of e_1, (2.5) clearly holds. If v is a vertex at an end of e_1,

$$\begin{aligned} a_{v,e_1} f(e_1) &= a_{v,e_1} f'(e_1') - a_{v,e_1} f'(\bar{e}_1') \\ &= a_{v,e_1'} f'(e_1') + a_{v,\bar{e}_1'} f'(\bar{e}_1'). \end{aligned}$$

The first equality holds because of (2.12), and the second one holds because $\bar{e}_1'$ has directions opposite to those of e_1. Therefore, because f' is a biflow on G',

$$\sum_{e \in E} a_{v,e} f(e) = \sum_{e \in E'} a_{v,e} f'(e_1) = b_v. \tag{2.13}$$

In addition, we prove (2.10). If $f'(e'_1) \geq n(e_1) + x$ and $f'(\bar{e}'_1) \geq x$ for some $x > 0$, $w_{G'}(f')$ can be decreased by subtracting x from $f'(e'_1)$ and $f'(\bar{e}'_1)$, which represents a contradiction. Therefore,

$$f'(e'_1) = n(e_1) \text{ or } f'(\bar{e}'_1) = 0. \tag{2.14}$$

In both cases,

$$\begin{aligned} |f(e_1) - n(e_1)| &= |f'(e'_1) - f'(\bar{e}'_1) - n(e_1)| \\ &= f'(e'_1) + f'(\bar{e}'_1) - n(e_1). \end{aligned} \tag{2.15}$$

From (2.8), (2.9), and (2.15), (2.10) holds. On the other hand, from definition of e'_1 and $\bar{e}'_1$, $f(e_1) = f'(e'_1) - f'(\bar{e}'_1) \geq \max\{l(e_1), n(e_1)\} - \max\{0, n(e_1) - l(e_1)\} \geq l(e_1)$. Therefore, f' satisfies (2.4).

Second, suppose that f on G is given. We show that there exists a biflow f' on G' that satisfies (2.10). Consider a mapping $f' : E' \to Z$ defined by (2.11) and

$$\begin{aligned} f'(e'_1) &= \max\{f(e_1), n(e_1)\}, \\ f'(\bar{e}'_1) &= \max\{0, n(e_1) - f(e_1)\}. \end{aligned}$$

Note that $f'(e'_1) \geq \max\{l(e_1), n(e_1)\} = l(e'_1)$ and $f'(\bar{e}'_1) \leq \max\{0, n(e_1) - l(e_1)\} = u(\bar{e}'_1)$. Therefore, f' satisfies (2.4).

If $f(e_1) \geq n(e_1)$, $f'(e'_1) = f(e_1)$ and $f'(\bar{e}'_1) = 0$. If $f(e_1) \leq n(e_1)$, $f'(e'_1) = n(e_1)$ and $f'(\bar{e}'_1) = n(e_1) - f(e_1)$. In both cases, (2.12) and (2.14) hold. Because (2.12) holds and f is a biflow on G, (2.13) holds, and thus, f' is a biflow on G'. In addition, (2.12) and (2.14) imply (2.15). Therefore, (2.10) holds.

Because f and f' can be calculated from each other in $O(1)$ time so that (2.10) holds, f that minimizes $w_G(f)$ can be calculated in $O(1)$ time if f that minimizes $w_{G'}(f')$ is given. □

Now we prove Lemma 2.5.

Proof. Let E^* be a subset of E, and G_{E^*} be the bidirected graph generated by dualizing edges in E^*. Also, let f_{E^*} be an optimal biflow on G_{E^*} that minimizes the following cost function:

$$w(f_{E^*}) = \sum_{e \in E^{**}} |e| f_{E^*}(e) + \sum_{e \in E - E^*} |e||f_{E^*}(e) - n(e)|,$$

where E^{**} is a set of edges generated by dualizing edges in E^*.

In addition, let $f_{E^* - \{e_1\}}$ be an optimal biflow on the graph in which edges in $E^* - \{e_1\}$ are dualized for some $e_1 \in E^*$. From Lemma 2.6, $f_{E^* - \{e_1\}}$ can be calculated from f_{E^*} in $O(1)$ time. Therefore, f can be calculated from f' by the algorithm in Figure 2.10 in $O(|E|)$ time. □

2.5.5 Circulation Corresponding to an Optimal Solution of ChrW

Let $n(e_N) = n_N$ and $n(e_T) = n_T$. Also, let $\tilde{G}'$ be the bidirected graph generated from $\tilde{G}$ by dualizing edges in $E_S \cup \{e_N, e_T\}$, and f be a circulation on $\tilde{G}$ whose cost is defined as follows:

$$w_{\tilde{G}}(f) = \sum_{e \in E \cup E_N} |e||f(e) - n(e)|. \tag{2.16}$$

We set $l(e_t(v))$ for $v \in V_W$ and $l(e)$ for $e \in E_W$ to 1 because these edges have to be visited in the solution.

```
1  $E^* := \{e \in E | e$ is dualized$\}$.
2  $f_{E^*} = f'$.
3  **repeat**
4      Arbitrarily choose $e_1 \in E^*$.
5      Calculate $f_{E^* - \{e_1\}}$ from $f_{E^*}$.
6      $E^* := E^* - \{e_1\}$
7      $f_{E^*} := f_{E^* - \{e_1\}}$.
8  **until** $E^*$ is empty.
9  Output $f_{E^*}$.
```

Figure 2.10: An algorithm that calculates an optimal biflow f on G.

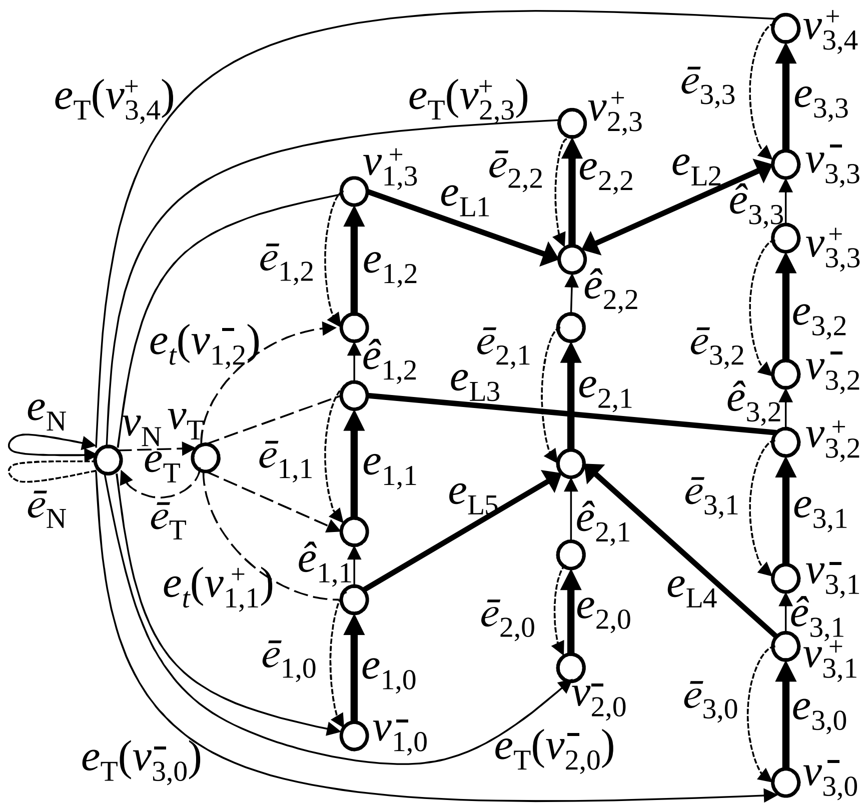


Figure 2.11: An example of a circular chromosome graph in which edges are dualized.

Lemma 2.7. *For any multi-set C of chromosomes on G, there is a circulation f on $\tilde{G}$ such that*

$$w_{\tilde{G}}(f) = W(C). \tag{2.17}$$

Conversely, for any circulation f on $\tilde{G}$ that minimizes $w_{\tilde{G}}(f)$, there is a multi-set C of chromosomes on G that satisfies (2.17). In addition, C can be calculated in $O(\sum_{e \in E \cup E_N} f(e))$ time.

Proof. First, we show that for any multi-set C of chromosomes on G, there exists a circulation f on $\tilde{G}$ that satisfies (2.17). Let $\mathrm{End}(v)$ be the number of chromosomes that begin or end with v. Consider the following f:

$$\begin{aligned} f(e) &= m(C,e) && \text{for } e \in E, \\ f(e_t(v)) &= \mathrm{End}(v) && \text{for } v \in V, \\ f(e_N) &= |C|, \\ f(e_T) &= \mathrm{Tr}(C). \end{aligned}$$

Then, f is a circulation on $\tilde{G}$ because f satisfies (2.4) and (2.5). Because $|e| = 0$ for $e \in E_L \cup E_R \cup \{e_t(v)|v \in V\}$,

$$\begin{aligned} w_{\tilde{G}}(f) &= \sum_{e \in E_S} |e||f(e) - n(e)| + |e_N||f(e_N) - n(e_N)| + |e_T||f(e_T) - n(e_T)| \\ &= \sum_{e \in E_S} |e||m(C,e) - n(e)| + |Q_N||C - n_N| + |Q_T||\mathrm{Tr}(C) - n_T| \\ &= W(C). \end{aligned}$$

Therefore, f satisfies (2.17).

Conversely, let f be a circulation on $\tilde{G}$ that minimizes $w_G(f)$. We show how to construct a multi-set C of chromosomes on G that satisfies (2.17).

First, we construct a set R of cycles such that $m(R,e) = f(e)$ for any edge e in $\tilde{G}$. Because of Lemma 2.4, This can be done in $O(\sum_{e \in E \cup E_N} f(e))$ time.

Second, we merge cycles in R. Let $E_+ = \{e \in E \cup E_N | l(e) \geq 1 \text{ or } n(e) \geq 1\}$. Note that $\mathrm{CC}(\tilde{G}, E_+)$ has only one weakly connected component because of WCC. This implies that all cycles that contain edges in E_+ can be merged into a single cycle. Note that any $r \in R$ contains at least one edge in E_+, because otherwise r can be removed to decrease $w_G(f)$. Therefore, all cycles in R can be merged into a single cycle $\tilde{r}$.

Finally, let C be a multi-set of paths generated by removal of v_N, v_T, and edges in E_N from $\tilde{r}$. Because $c \in C$ is connected to edges in E_N in $\tilde{r}$, the first and last edge of c is in E_S due to the directions of these edges. Accordingly, c is a chromosome. Therefore, C is a multi-set of chromosomes on G.

All of these steps can be completed in $O(\sum_{e \in E \cup E_N} f(e))$ time. In addition, we observe that the following equations hold:

$$\begin{aligned} |C| &= f(e_N), \\ \mathrm{Tr}(C) &= f(e_T), \\ m(C,e) &= f(e) \quad \text{for } e \in E_S. \end{aligned}$$

Therefore, C satisfies (2.17). □

2.5.6 Polynomial-time Algorithm

Based on the discussion so far, an optimal set of chromosomes can be calculated in $O(|E|^2 \log |V| \log |E|)$ by the following algorithm.

1. Construct a circular chromosome graph $\tilde{G}$ from G. This requires $O(|V|)$ time.

2. Construct a dualized circular chromosome graph $\tilde{G}'$ by dualizing e_N, e_T, and the edges of $\tilde{G}$ in E_S. This requires $O(|E|)$ time.

3. Calculate an optimal circulation f' on $\tilde{G}'$ where $w(e, f') = |e|f'(e)$ for each edge e in $\tilde{G}'$ by Gabow's algorithm [16]. By Lemma 2.1, setting $u(e) = U(4|V|+1)(|E|+1)$ does not affect the solution. Therefore, f' can be calculated in $O(|E|^2 \log |V| \log(\max_{e \in E}\{u(e)\})) = O(|E|^2 \log |V| \log |E|)$ time.

4. Calculate an optimal circulation f on $\tilde{G}$ from f'. This requires $O(|E|)$ time by Lemma 2.5.

5. Construct a set of chromosomes C that satisfies $W(C) = w_{\tilde{G}}(f)$, which exists by Lemma 2.7. This requires $O(\sum_{e \in E} f(e)) = O(|V||E|)$ time by Lemma 2.1.

6. Output C.

2.6 Computational Experiment

Our algorithm for ChrW was evaluated in computational experiments with two data sets. The first data set is simulated data of which all translocations, CNVs, adjacencies in the target genomes, and ends of chromosomes of the target genomes are known. Therefore, we could use them so that instances of the ChrW satisfied WCC and the inferred genomes were evaluated by comparing them with the true target genome. The second data set is a data set of real SV data converted from the data of the International Cancer Genome Consortium (ICGC) [13].

2.6.1 Implementation

For implementation, we require an algorithm that can calculate an optimal circulation on the bidirected graph. Although Gabow's algorithm can calculate an optimal biflow in polynomial time, no efficient implementation is currently known. Therefore, we used the GNU Linear Programming Kit (GLPK)[1] to calculate an optimal circulation.

The rest of the algorithm were implemented with C++ language. The parameters Q_N, Q_T were set to 1×10^7, 5×10^7, respectively. Edges $e \in E_S$ was assumed to be required if $n(e) > 0$.

2.6.2 Experiment with Simulated Data

Simulation

To evaluate our algorithm by using SV data whose target genome is known, a set of simulated SV data was generated as follows by using random numbers.

1. Determine three parameters n, P_T and P_L. The first parameter n is the number of chromosomes to be generated. The second parameter P_T affects the number of truncations, while the third parameter P_L affects the number of translocations.

2. The reference genome of the human genome (version GRCh38) was split into 205 segments.

3. A set of n chromosomes was generated. Each chromosome is generated as follows.

   (a) A segment generated in Step 2 was randomly chosen.

   (b) A direction, which is one of toward the upstream segment and the downstream segment, was also randomly chosen. Then, go though the reference genome from the chosen segment.

[1]`http://www.gnu.org/software/glpk/glpk.html`

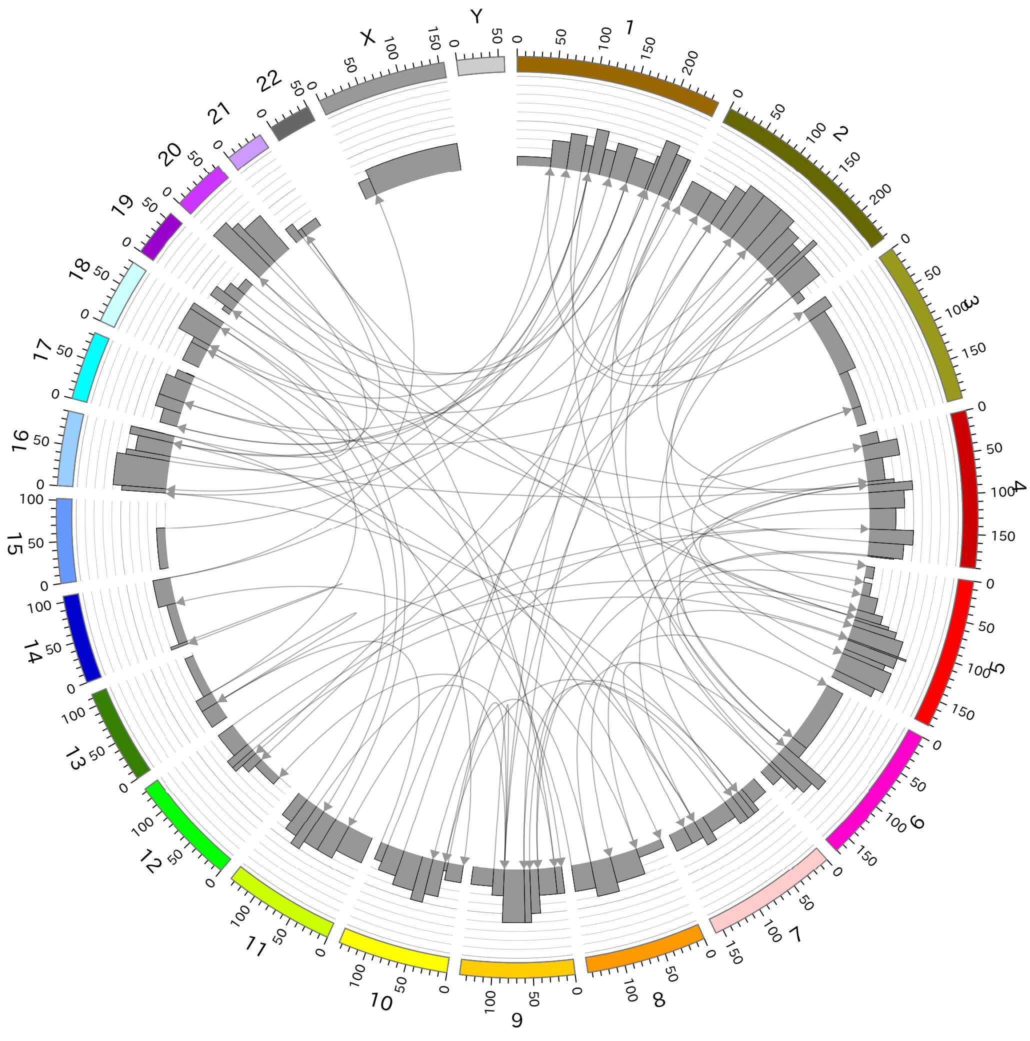


Figure 2.12: The translocations and copy number variations in a virtual genome generated by simulation. The outermost numbers 1, 2, . . ., 22 and symbols X, Y are the names of chromosomes in the reference human genome GRCh38. The positions along with chromosomes are in mega-bases. The histogram filled in gray represents copy numbers. Each link connecting two positions on the genome represents a translocation. For each translocation, an arrowhead is shown at an end if the translocation connects to the downstream segment at the end. No data were not shown for chromosomes 21 and Y because no segments in these chromosomes were chonsen in our simulation which depended on random numbers. Note that if multiple translocations share the same endpoint, an arrow head is shown whenever at least one of the translocations connects to the downstream segment. This figure was generated by using Circos [32].

   (c) When a segment was gone through, stop with the probability P_T. Otherwise, jump to randomly chosen segment with the probability P_L. The direction after the jump is also determined randomly.

4. From the generated chromosomes, a set of SV data was generated with data to satisfy WCC as follows.

   (a) Translocations: Each jump at Step 3c was extracted as a translocation.

   (b) CNVs: The copy number of each segment was calculated by counting the number of visits to the segment.

   (c) Ends of chromosomes: Their positions in the reference genome were collected.

   (d) Required Edges: Each edge in the chromosome graph was considered as a required edge if it corresponded to segments with copy numbers greater than zero or to adjacencies of generated chromosomes.

In our experiment, we set $n = 50$, $P_T = 0.05$, and $P_L = 0.2$. The result of simulation is shown in Figure 2.12. The generated data set consisted of 50 chromosomes, 86 translocations, 29 truncations. Segments with non-zero copy numbers spanned 2,913,132,812 bp on GRCh38 in total.

Results for Perfect Data

First, we provided our algorithm with the generated SV data as they were. Because of ambiguity in conversion of a circulation to chromosomes [56, 66], it is impossible in general to reconstruct the generated chromosomes. However, CNVs can be optimized because it does not have such ambiguity. Because the generated SV data was provided as it was, our algorithm can find an optimal solution whose copy numbers are exactly the same as that of the original simulated genome.

As shown in Figure 2.13, the inferred set of chromosomes had copy numbers exactly the same as those generated by simulation for all 2,913,132,812 bp. For 81 translocations out of all 86 translocations, the number of usage in the inferred genome was identical to that of the translocation in the generated data set.

Results for Data with Errors in Copy Numbers

To evaluate the robustness of our algorithm against errors in copy numbers, we add noises to the generated SV data. We randomly chose n segments with non-zero copy numbers in the SV data where $n = 1, 2, \ldots, 20$, and modified each of the copy numbers by one.

As shown in Figures 2.14 and 2.15, the number of segments with incorrectly inferred copy numbers was almost always less than the number of segments with noise. From this result, it was confirmed that our algorithm can correct the errors in copy numbers.

Results for Data with False Positive Translocations

To evaluate the robustness of our algorithm against false positive translocations, we added n randomly generated translocations from the generated SV data, where $n = 1, 2, \ldots, 20$. Each additional translocation was generated by randomly choosing two segments with a non-zero copy number in the generated SV data and linking them. The additional translocations were considered as non-required edges in the chromosome graph.

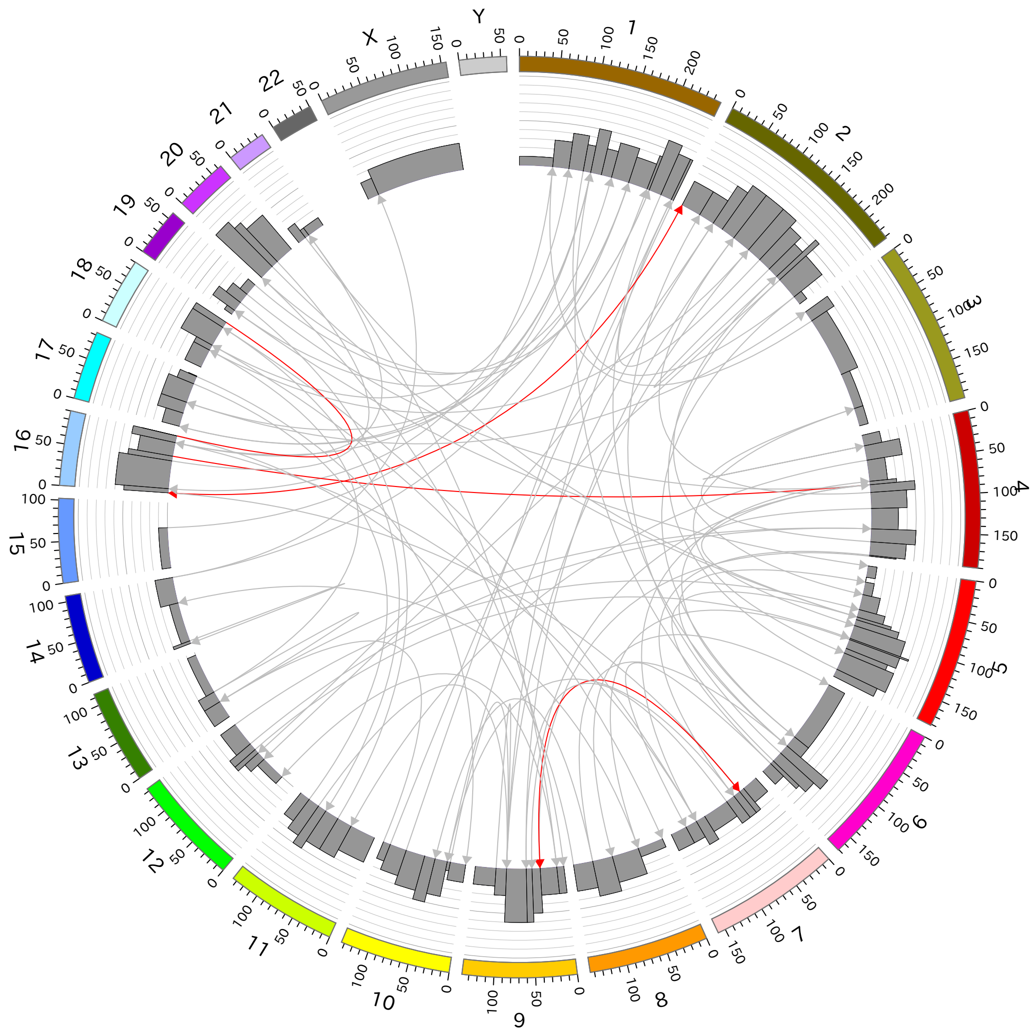


Figure 2.13: The translocations and copy number variations in an inferred genome by using the perfect data without noise. Red links represent translocations used more in the inferred genome than in the original simulated genome. All copy numbers were consistent with the original simulated genome. In addition, most of translocations except were used in the inferred genome exactly the same number of time as the original simulated data. The inconsistent translocations were inferred to be used more.

As shown in Figures 2.16 and 2.17, many of the additional translocations were correctly ignored. This is a promising result, although it would be harder to infer false positive translocations in real situations because additional translocations were considered as non-required edges in the chromosome graph while those in the original simulated genome were considered as required edges. All copy numbers were inferred correctly. This was because an optimal solution that ignores all additional translocations exists.

Results for Data with Missing Translocations

To evaluate the robustness of our algorithm against missing translocations, we removed randomly chosen n translocations from the generated SV data, where $n = 1, 2, \ldots, 20$.

As shown in Figures 2.18, the number of segments with incorrectly inferred

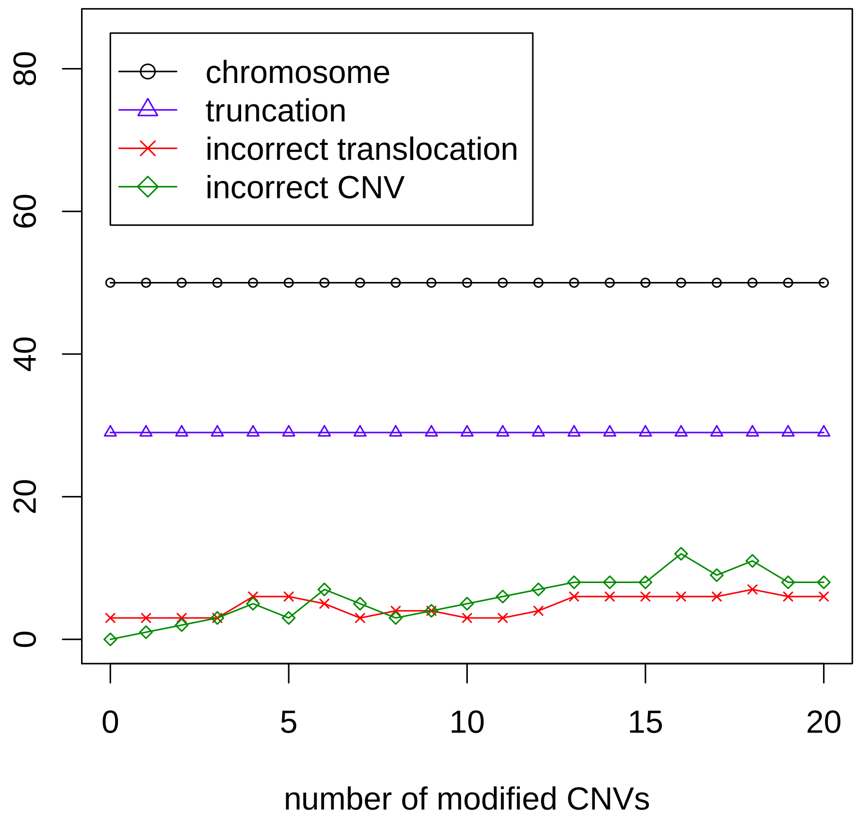


Figure 2.14: The accuracy of our algorithm against errors in copy numbers. The number of segments with different copy numbers from the original simulated genome is less than the number of modified copy numbers, which indicates the ability of our method to recover correct copy numbers. The number of incorrectly inferred translocations increased as the number of incorrect copy numbers increased.

copy numbers increased as n increased. In addition, the number of incorrectly inferred translocations increased faster. This result indicates that inferring the target genome is difficult when missing translocations exist. Figure 2.19 show an example of the inferred genomes.

Effect of Considering the Number of Chromosomes and Truncations

To demonstrate the advantage of our method which can take into account the number of chromosomes and truncations, we conducted two experiments. In the first experiment, we set $Q_N = Q_T = 0$ to remove the first and second terms of (2.1). In the second experiment, we set $Q_N = 1 \times 10^7$ and $Q_T = 2 \times 10^7$ as in other experiments. In both experiments, we added noise to the SV data of the original simulated genome by modifying copy numbers of 20 segments and by adding 20 additional translocations.

As shown in Table 2.1, by taking into account the number of chromosomes

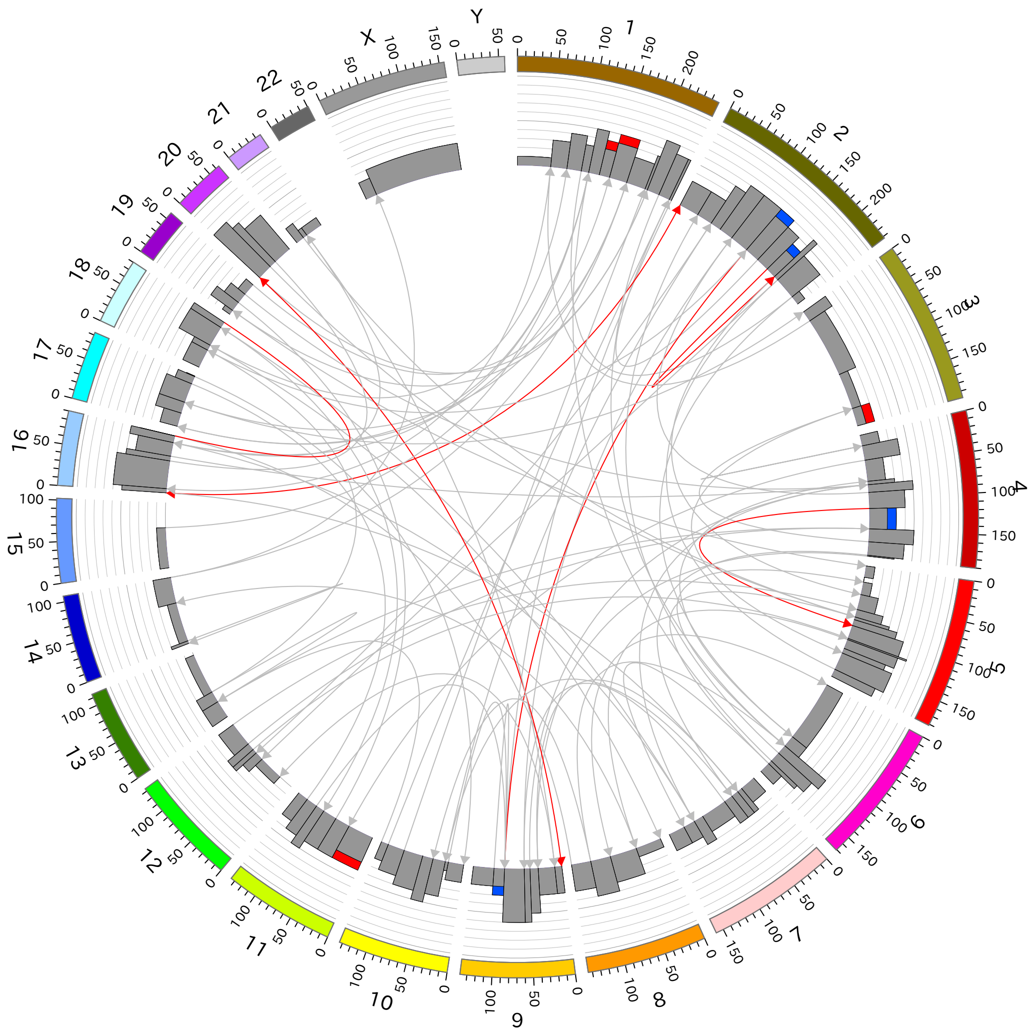


Figure 2.15: The inferred translocations and copy number variations when copy numbers were modified for 20 segments. The number of segments with inconsistent copy numbers were less than 20, which indicates that our method recovered modified copy numbers. However, several translocations were used more in the inferred genome than in the original simulated genome.

and truncations with the first and second terms of the cost function (2.1), our method successfully estimated these numbers. In addition, the number of incorrect copy numbers and translocations also decreased. Therefore, the effectiveness of considering the number of chromosomes and truncations was confirmed.

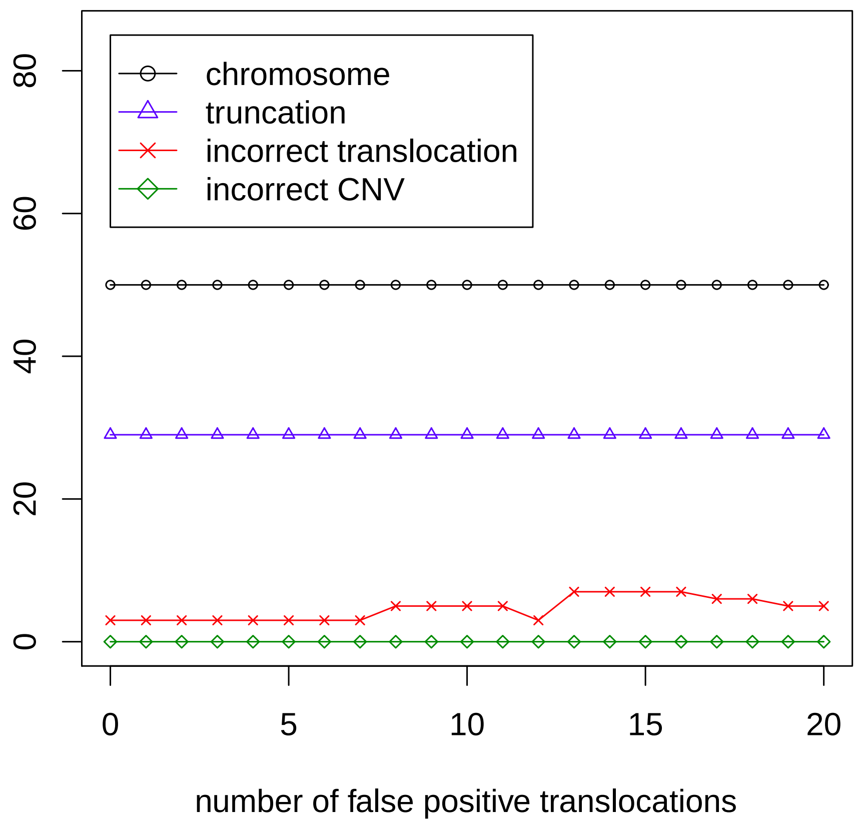


Figure 2.16: The accuracy of our algorithm against false positive translocations. Our method correctly ignored many false positive translocations. However, because all additional translocations were converted to non-required edges in the chromosome graph, it would be harder to infer false positive translocations in real situations than in this experiment. The copy number was always exactly inferred because they were not modified and no translocation was removed.

Table 2.1: The number of inferred chromosomes and truncations when the number of the first and the second terms of the cost function (2.1) were considered or were not considered. By setting $Q_N = 1 \times 10^7$ and $Q_T = 2 \times 10^7$ to consider the first and second terms in the cost function (2.1), our method could infer a set of chromosomes with more accurately estimated number of chromosomes and truncations.

	simulation	$Q_N = 1 \times 10^7$, $Q_T = 2 \times 10^7$	$Q_N = Q_T = 0$
chromosomes	50	60	50
truncations	29	53	29
incorrect copy numbers	–	20	14
incorrect translocations	–	17	13

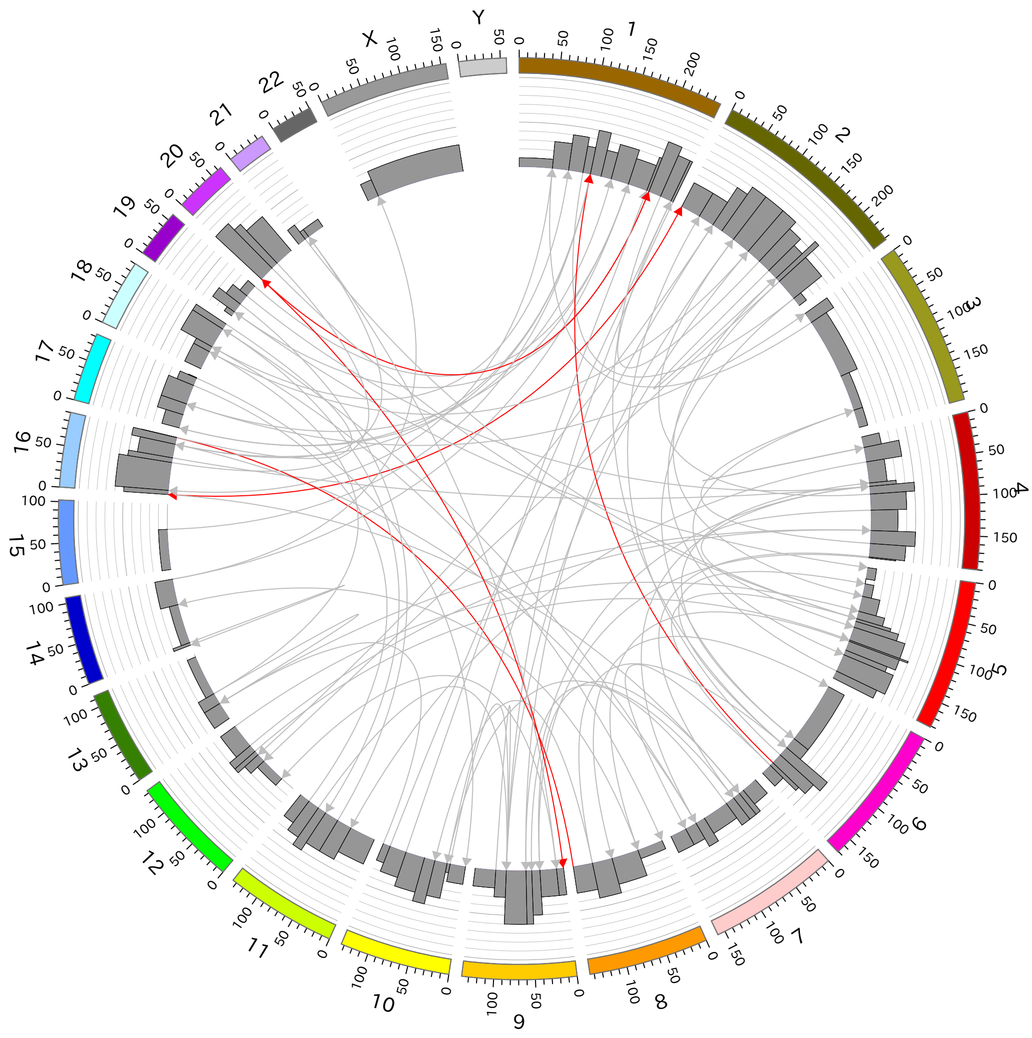


Figure 2.17: The inferred translocations and copy number variations when 20 random translocations were added. All of the additional translocations were correctly ignored. However, five translocations were incorrectly used more in the inferred genome than in the original simulated genome. As in Figure 2.16, it would be harder to infer false positive translocations in real situations than in this experiment.

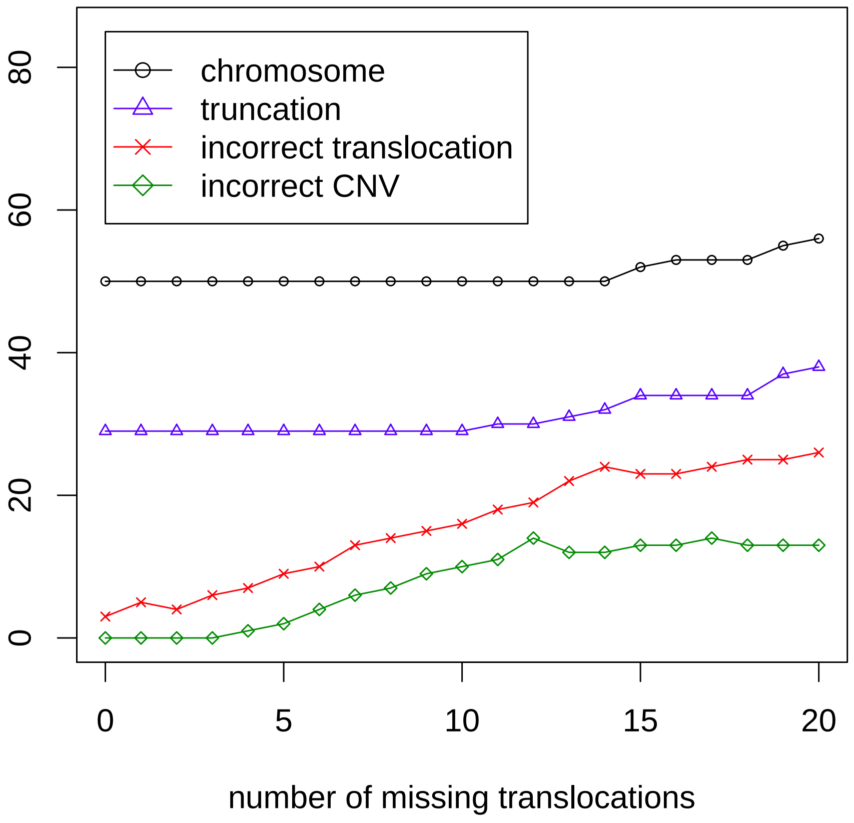


Figure 2.18: The accuracy of our algorithm against missing translocations. Both of incorrectly inferred copy numbers and translocations increased as the number of missing translocations increased. This result indicates that it is difficult to infer correct copy numbers and translocations when a lot of translocations are missing.

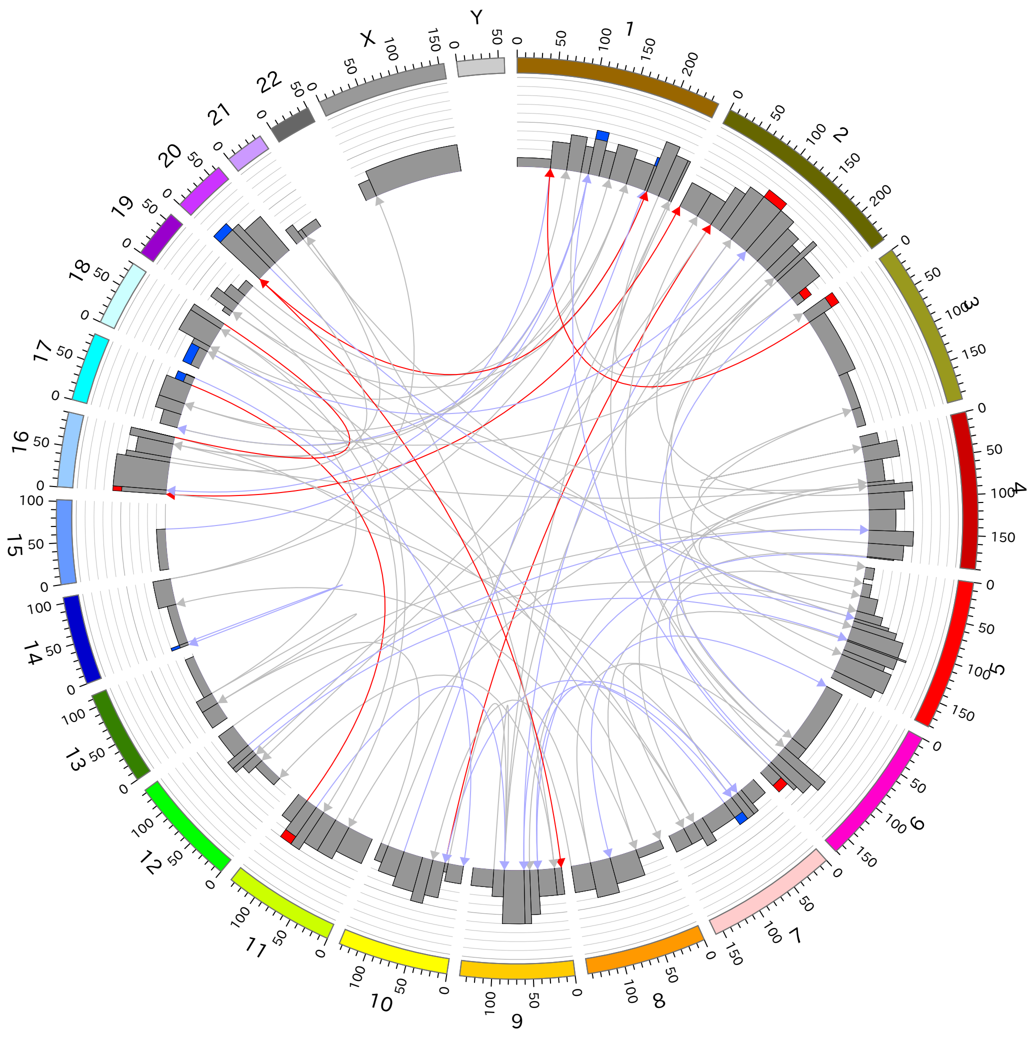
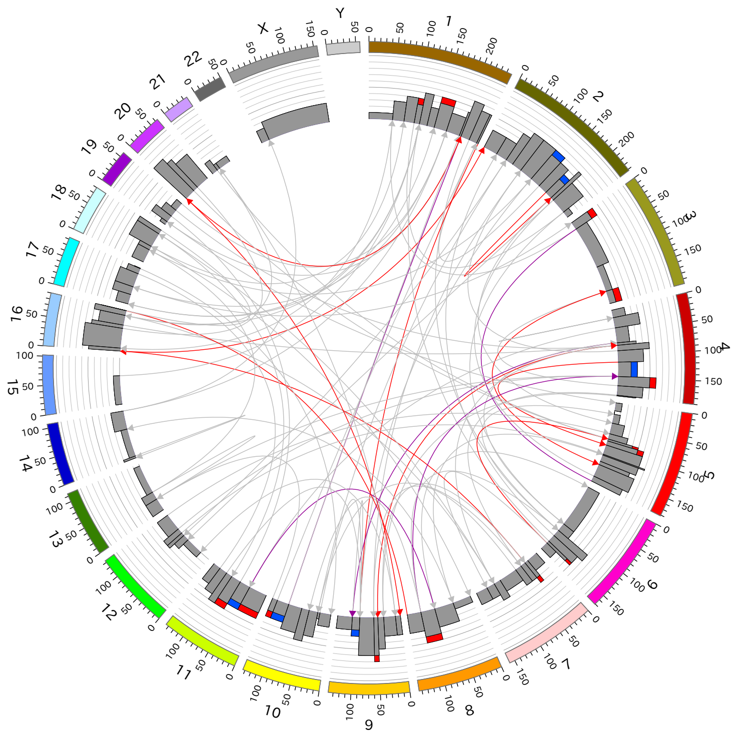


Figure 2.19: The inferred translocations and copy number variations when 20 translocations were removed. Light blue translocations were vanished in the inferred genome than in the original simulated genome. Both of the inferred translocations and CNVs included several errors.

A



B

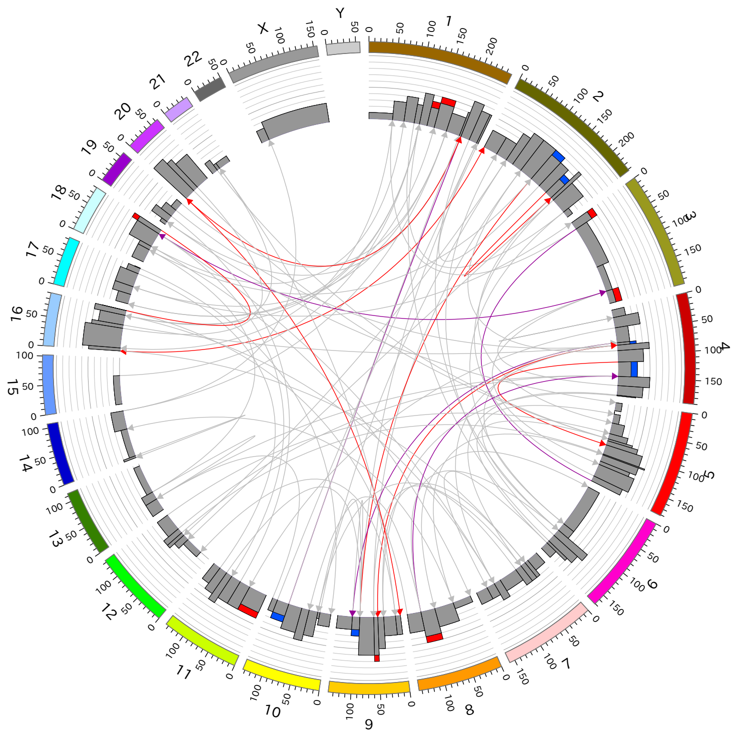


Figure 2.20: The inferred translocations and copy number variations when copy numbers were modified for 20 segments and 20 translocations were added. Purple translocations were false positive ones incorrectly inferred to be used. A: The results of inference when parameters Q_N and Q_T were both set to 0. B: The results of inference when parameters Q_N and Q_T were set to 1×10^7 and 2×10^7, respectively. The results shown in B had less incorrectly inferred translocations and CNVs than those shown in A.

Table 2.2: SV data obtained from ICGC (release 17).

sample ID	number of involved chromosomes	truncation	CNV
SA130868	10	34	8
SA130876	11	39	25
SA130901	13	24	19
SA130903	14	25	6
SA130905	11	15	5
SA130909	10	35	19
SA130911	18	47	21
SA130913	16	26	12
SA130915	14	17	6
SA514938	17	29	13
SA514940	13	43	6
SA514942	13	23	6
SA514946	13	22	7
SA514948	17	28	6
SA514958	12	16	2
SA514959	9	22	6
SA514961	15	54	13
SA514962	14	48	10

2.6.3 Experiment with Real SV Data

Our algorithm was also applied to real SV data. The real SV data were obtained from the open repository of International Cancer Genome Consortium (ICGC) [13] (release 17). There were 50 project of sequencing cancer genomes conducted by various countries for various types of cancers. Among them, only eight projects (BOCA-UK, BRCA-UK, EOPC-DE, OV-AU, PACA-AU, PAEN-AU, PBCA-DE,PRAD-UK) provided data of both translocations and CNVs. Each of the eight projects provided SV data of many samples, which amounted to 653 samples in total. Among them, we chose 24 samples that satisfied the following conditions.

- Data of both translocations and CNVs were provided.
- Segments with copy numbers did not overlap each other.
- Translocations had valid strand information.

Because the obtained data did not include positions of terminals of chromosomes to be inferred and highly likely adjacencies, WCC could not be satisfied. In addition, copy numbers were provided only limited portions of the reference genome. To apply our method to these data, we added the following data under an assumption that the target genomes to be inferred were similar to the reference genome unless contradicting SV data were provided.

- For each chromosome with any copy number in SV data, copy numbers of segments without data were set to two.
- All edges in E_R were considered to be required.
- All ends of chromosomes in the reference genome were also ends of some chromosome in the target genome.

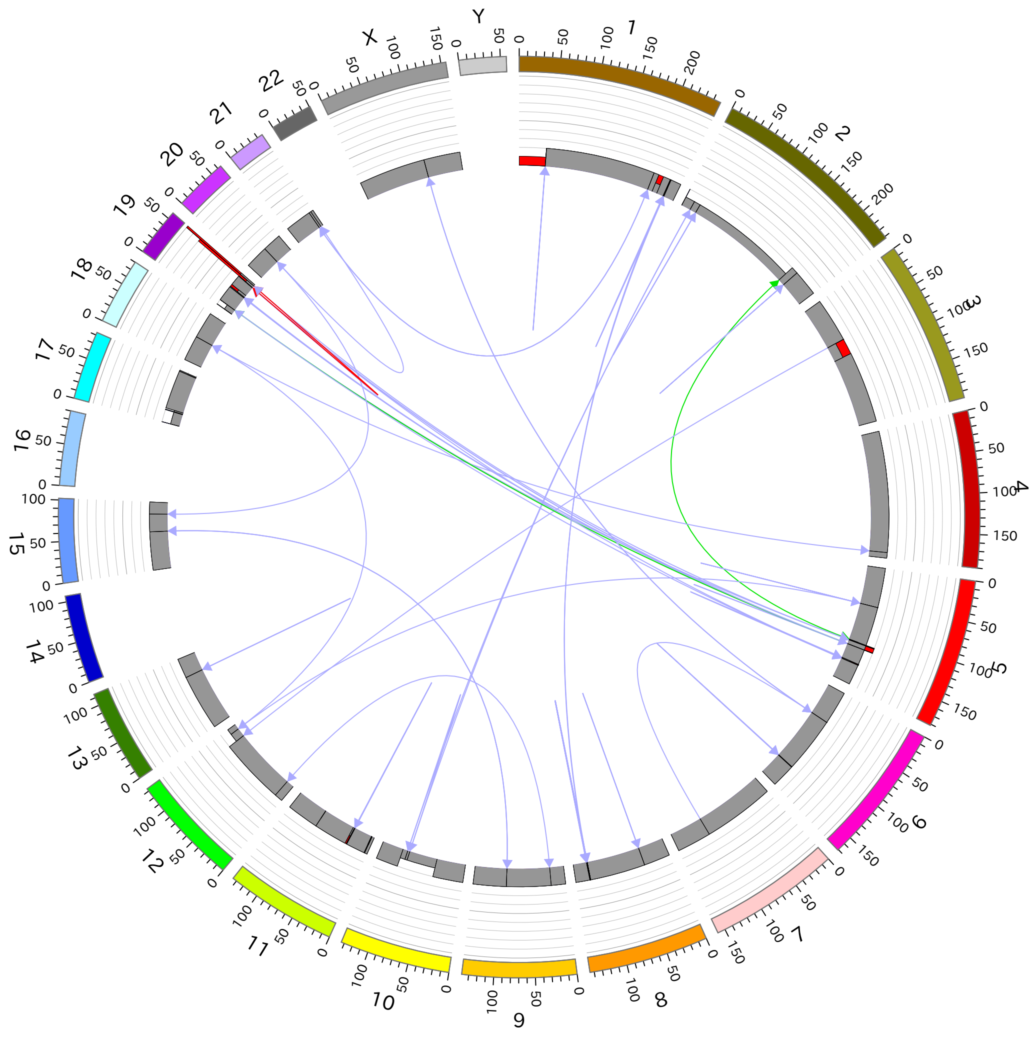


Figure 2.21: An example of inferred genomes for the SV data obtained from ICGC. This figure shows an inferred genome for sample SA130911. Inferred copy numbers were equal to the given copy numbers in segments described in the SV data, or assumed copy numbers in segments not described in the SV data. However, only a few translocations in the SV data were used and the rest were removed.

Due to lack of structures of chromosomes for these samples, it is difficult to appropriately evaluate the results. However, we observed that most of translocations in SV data were removed in the inferred genomes in this experiment. Figure 2.21 and 2.22 show examples of the inferred genomes. Possible reasons of this result include (i) the SV data contained a lot of false positive translocations, (ii) translocations and CNVs in the SV data were inconsistent, (iii) the assumptions above were not appropriate, and (iv) our method did not work well because of noise or an unexpected structure of chromosomes that cannot be optimized by the cost function (2.1). Further analysis would be required.

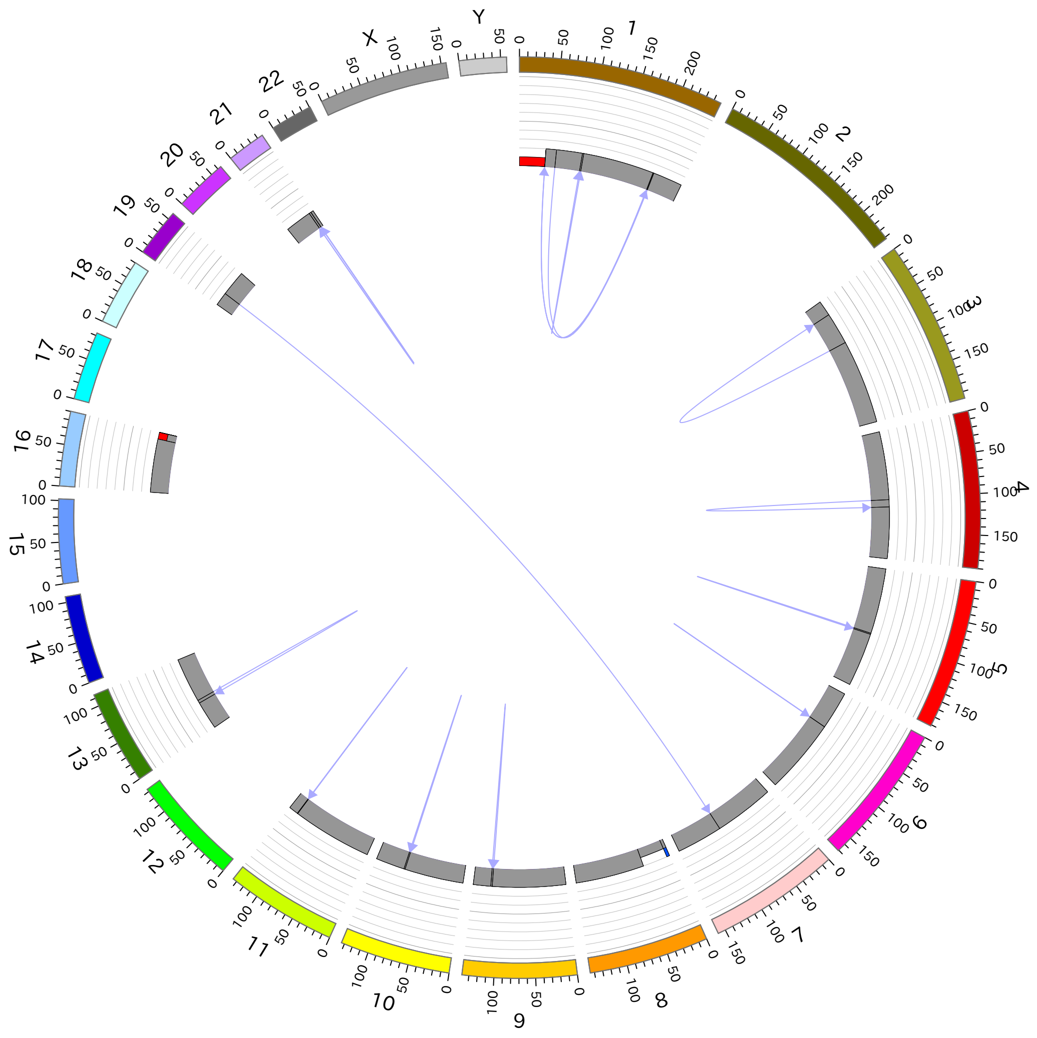


Figure 2.22: Another example of inferred genomes for the SV data obtained from ICGC. This figure shows an inferred genome for sample SA514958. For the SV data of this sample, no translocation in the SV data was used in the inferred genome.

2.7 Necessity of Restriction on the Length of Chromosomes

In ChrW, we removed restrictions on the length of chromosomes. This relaxation is necessary to make the problem solvable in polynomial time.

Definition 2.7 (ChrW with restriction on length (ChrL))**.** *ChrW with restriction on length (ChrL) is the same problem as ChrW, except that the length of each chromosome c_i is bounded by a parameter λ_i ($1 \le i \le N_L$), where N_L is the maximum possible number of chromosomes.*

Theorem 2.3. *The problem ChrL is NP-complete.*

Proof. ChrL is in NP because of Lemma 2.1.

Here, we show that the well-known PARTITION problem [17] can be reduced to ChrL. Let n be a positive integer and $S = \{i \in Z | 1 \le i \le n\}$. Also, let $s(i)$ be an integer function defined for $i \in S$ such that $s(i) > 0$, and $S_\Sigma = \sum_{i \in S} s(i)$. The problem of finding a subset $S' \subset S$ such that

$$\sum_{i \in S'} s(i) = \sum_{i \in S - S'} s(i) = S_\Sigma / 2$$

is called the *partition problem* (hereafter referred to as *PARTITION*) [17]. It is well known that PARTITION is NP-complete. We reduce PARTITION to ChrL by constructing a chromosome graph whose solution for ChrL contains two chromosomes that correspond to two subsets of a solution of PARTITION.

Let $G = (V, E)$ be a chromosome graph, where

$$V = \bigcup_{1 \le i \le n+1} \{v_{i,0}^-, v_{i,1}^+, v_{i,1}^-, v_{i,2}^+, v_{i,2}^-, v_{i,3}^+\}$$

is a set of vertices, and $E = E_S \cup E_L \cup E_R$ be a set of edges. Here, E_S consists of

$$\begin{aligned}
e_{i,0} &= \langle -v_{i,0}^-, +v_{i,1}^+, 1, 9S_\Sigma \rangle \quad (1 \le i \le n), \\
e_{i,1} &= \langle -v_{i,1}^-, +v_{i,2}^+, 2, s(i) \rangle \quad (1 \le i \le n), \\
e_{i,2} &= \langle -v_{i,2}^-, +v_{i,3}^+, 1, S_\Sigma - s(i) \rangle \ (1 \le i \le n), \\
e_{n+1,0} &= \langle -v_{n+1,0}^-, +v_{n+1,1}^+, 2, 9S_\Sigma/2 \rangle, \\
e_{n+1,1} &= \langle -v_{n+1,1}^-, +v_{n+1,2}^+, n+2, 0 \rangle, \\
e_{n+1,2} &= \langle -v_{n+1,2}^-, +v_{n+1,3}^+, 2, 5S_\Sigma \rangle.
\end{aligned}$$

In addition, E_R consists of

$$\begin{aligned}
\hat{e}_{i,1} &= \langle -v_{i,1}^+, +v_{i,1}^-, 0, 0 \rangle \quad (1 \le i \le n{+}1), \\
\hat{e}_{i,2} &= \langle -v_{i,2}^+, +v_{i,2}^-, 0, 0 \rangle \quad (1 \le i \le n{+}1),
\end{aligned}$$

and E_L consists of

$$\begin{aligned}
e_{Li} &= \langle +v_{i,1}^-, -v_{n+1,2}^+, 0, 0 \rangle \quad (1 \le i \le n), \\
e'_{Li} &= \langle -v_{i,2}^+, +v_{n+1,1}^-, 0, 0 \rangle \quad (1 \le i \le n).
\end{aligned}$$

We set $\lambda_i = 10S_\Sigma$ for any $i \ge 1$, $Q_N = Q_T = 100S_\Sigma$, $n_N = n + 2$, and $n_T = 0$. See Figure 2.23 for an example. In addition, we set V_W to $V_5 \cup V_3$, and E_W to E by making all edges in $E_L \cup E_R$ required so that G satisfies WCC.

We show that PARTITION for S has a solution $S' \subset S$ if, and only if, there exists a solution C of ChrL such that $W(C) = 0$. First, suppose that PARTITION has a solution S'. Let $r_{S'}$ be a cycle generated by merging cycles

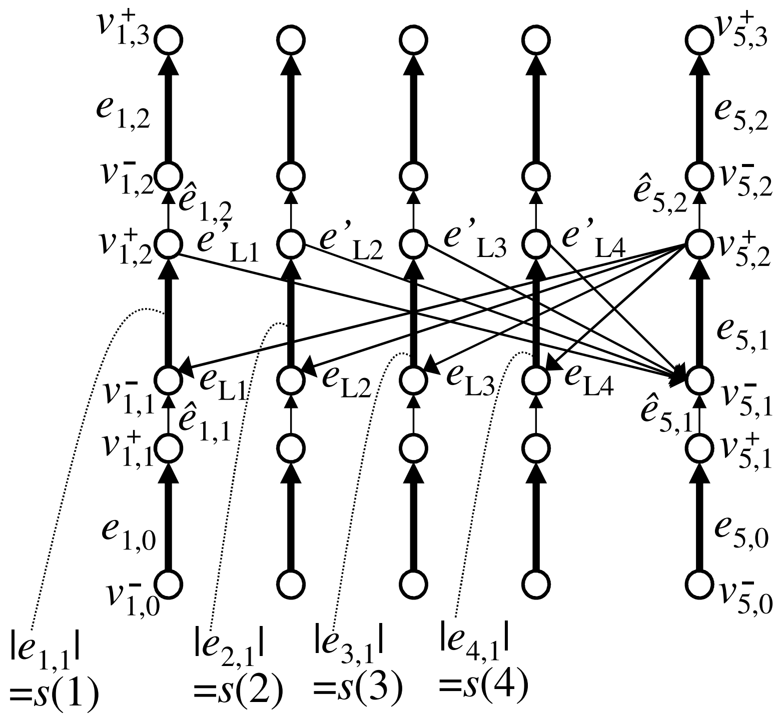


Figure 2.23: An example of a chromosome graph for solving the partition problem (PARTITION). In this example, $n = 4$.

$e_{n+1,1}e_{Li}e_{i,1}e'_{Li}$ for $i \in S'$. We define $r_{S-S'}$ in the same way. Consider a multi-set $C = \{c_1, \ldots, c_{n+2}\}$, where $c_i \in C$ is a chromosome on G such that

$$\begin{aligned} c_i &= e_{i,0}\hat{e}_{i,1}e_{i,1}\hat{e}_{i,2}e_{i,2} \ (1 \le i \le n), \\ c_{n+1} &= e_{n+1,0}\hat{e}_{n+1,1}r_{S'}e_{n+1,1}\hat{e}_{n+1,2}e_{n+1,2}, \\ c_{n+2} &= e_{n+1,0}\hat{e}_{n+1,1}r_{S-S'}e_{n+1,1}\hat{e}_{n+1,2}e_{n+1,2}. \end{aligned}$$

Then, $W(C) = 0$ because $|C| = n + 2$, $\mathrm{Tr}(C) = 0$, and $m(C, e) = n(e)$ for $e \in E_S$. In addition, C visits all required edges. Furthermore, $|c_i| = 10\Sigma \le \lambda_i$ for $1 \le i \le n + 2$.

Conversely, suppose that ChrL for G has an optimal solution C that satisfies $W(C) = 0$. Because $W(C) = 0$, we obtain $|C| = n+2$, $\mathrm{Tr}(C) = 0$, and $m(C, e) = n(e)$ for $e \in E$. Because $\sum_{e \in E} |e|n(e) = 10(n + 2)S_\Sigma$, $|c| = 10\Sigma$ for each $c \in C$. Let c_i be a chromosome that begins with $e_{i,0}$ for $1 \le i \le n$. The other two chromosomes are denoted by c_{n+1} and c_{n+2}. Then, c_1 begins with $e_{1,0}\hat{e}_{1,1}e_{1,1}$. Suppose that c_1 does not visit $\hat{e}_{1,2}e_{1,2}$. Then, there is a chromosome c_i that visits $\hat{e}_{1,2}e_{1,2}$, whose previous edge has to be $e_{1,1}$ in c_i. Therefore, for some paths p_1 and p_2,

$$\begin{aligned} c_1 &= e_{1,0}\hat{e}_{1,1}e_{1,1}p_1 \\ c_i &= p_2e_{1,1}\hat{e}_{1,2}e_{1,2}. \end{aligned} \tag{2.18}$$

Because of (2.18), $|c_1| = |e_{1,0}| + |\hat{e}_{1,1}| + |e_{1,1}| + |p_1| = 10S_\Sigma = |e_{1,0}| + |\hat{e}_{1,1}| + |e_{1,1}| + |\hat{e}_{1,2}| + |e_{1,2}|$. Therefore, $|p_1| = |\hat{e}_{1,2}| + |e_{1,2}|$. We modify C so that

$$\begin{aligned} c_1 &= e_{1,0}\hat{e}_{1,1}e_{1,1}\hat{e}_{1,2}e_{1,2}, \\ c_i &= p_2e_{1,1}p_1. \end{aligned}$$

The modified C still satisfies the required conditions. After this modification is repeated for $2 \leq i \leq n$ until no more modifications can be applied, C satisfies $c_i = e_{i,0}\hat{e}_{i,1}e_{i,1}\hat{e}_{i,2}e_{i,2}$ for $1 \leq i \leq n$. Another chromosome exists that visits $e_{i,1}$ for each $1 \leq i \leq n$, which is one of c_{n+1} and c_{n+2}. Let $S' = \{i | m(c_{n+1}, e_{i,1}) > 0\}$. Then, $\sum_{i \in S'} s(i) = 10S_\Sigma - (9/2 + 5)S_\Sigma = 1/2 S_\Sigma$. Therefore, S' is a solution of PARTITION. □

2.8 Discussion

Handling Practical Situations

In addition to segments in the reference genome, our method can handle newly inserted fragments not in the reference genome. Such a fragment is incorporated into a chromosome graph as a new chromosome. In particular, an edge e, where $|e|$ is equal to the length of the fragment, is added to E_S, and edges that connect vertices in a chromosome graph to e are added to E_L. If any breakpoints are contained within the new fragment, vertices and edges are added to V_M and E_R, respectively. If a breakpoint corresponds to any aberrant adjacency, edges are also added to E_L.

If a gene duplication has occurred in the target genome, it causes an increased copy number and aberrant adjacencies flanking the gene. If it is a tandem duplication, an aberrant adjacency connecting the upstream and downstream segments of the gene should exist. If these SVs exist in given SV data, any solution to our problem has to take into account gene duplication.

Limitations

A mixture of many cells cannot be handled because it is difficult to correctly estimate copy numbers. However, our method may generate meaningful results for data obtained from multiple cells if the sum of copy numbers is correctly estimated. In this case, the solution is a mixture of chromosomes of all cells in the sample, although some of the chromosomes might be fused.

Note that many optimal solutions may exist depending on how an optimal circulation is converted into chromosomes (Figure 2.24). Choosing the right solution requires additional information such as the mate-pairs of long genomic fragments, or the result of experiments involving such techniques as fluorescence *in situ* hybridization (FISH) that indicate whether or not distant genomic segments are in the same chromosome.

2.9 Summary

We formulated the problem of inferring chromosomes from the aberrant adjacencies of genomic regions, copy number variations (CNVs), and the number and length of chromosomes. The problem, which we term as the *chromosome problem (ChrP)*, was proved to be NP-complete. However, if an instance of ChrP satisfies a constraint, which we call a *weakly connected constraint (WCC)*, and if the length of chromosomes is ignored, the problem can be solved in $O(|E|^2 \log |V| \log |E|)$ time. We also explained that ignoring upper bounds on the length of chromosomes is necessary because another variation of ChrP that bounds the length of inferred chromosomes is NP-complete.

In computational experiments, our algorithm that solves ChrW could infer CNVs and false positive translocations. However, it is difficult to cope with

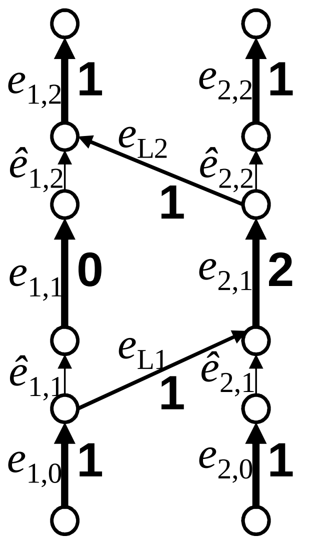


Figure 2.24: Bold digits represent an optimal circulation on this graph. The chromosome graph in this figure has two optimal solutions $\{e_{1,0}e_{L1}e_{2,1}e_{L2}e_{1,2},\ e_{2,0}\hat{e}_{2,1}e_{2,1}\hat{e}_{2,2}e_{2,2}\}$ and $\{e_{1,0}e_{L1}e_{2,1}\hat{e}_{2,2}e_{2,2},\ e_{2,0}\hat{e}_{2,1}e_{2,1}e_{L2}e_{1,2}\}$. Edges in $E_N \cup E_D$ are omitted, and the flow on each edge in E_D has been subtracted from the flow of a corresponding edge in E_S.

missing translocations. It is also necessary to analyze the behavior of our method for real SV data.

Chapter 3

Resolution Improvement for Detection of Structural Variations

In this chapter, we propose a new method called *ChopSticks* that improves the resolution of the positions of homozygous deletions detected mainly by methods based on paired reads with aberrant mapping distances and/or strands. ChopSticks exploits normally mapped paired reads in addition to aberrantly mapped ones. By using this new independent information, ChopSticks improves the positions of SVs detected by other methods. We theoretically analyze the improvement of the resolution. In addition, we demonstrate the effectiveness of ChopSticks in computational experiments.

Contents of this chapter are mainly from published work in [86].

3.1 Related Works and Our Contribution

Current computational methods for SV detection based on NGS search for *signatures* that indicate SVs hidden in NGS sequences and their alignments with the reference genome. The following are basic signatures used for SV detection [55, 58, 79].

3.1.1 Read Pair (RP)

If paired reads have aberrant strands or distances, they are likely to be caused by SVs [10, 25, 70]. Such pairs are called *discordant* pairs, and normally mapped ones are called *concordant* pairs. If strands of a discordant pair are as expected, a larger distance than expected indicates a deletion, whereas a smaller distance indicates an insertion. There are several categories of methods that detect discordant pairs by using mapping distances.

- Threshold-based: A pair with a mapped distance larger or smaller than a predefined threshold is defined as a discordant pair. The threshold is $\mu \pm 3\sigma$ or $\mu \pm 4\sigma$ for BreakDancer [10] and VariationHunter [25] where μ and σ are mean and standard deviation of mapped distances, or median fragment size $\pm$ 10 median absolute deviations for HYDRA [70].

- Distribution-based: Although the mapped distance of a single pair might vary by tens or hundreds bases even without SVs, greater (smaller) mapped distances of many pairs in the same region indicate deletions (insertions). Such reads can be detected by statistical tests on the distribution of mapped distances [10, 40]. Paired reads that support SVs detected in this way might have mapping distances more similar to the expected distance than those of other methods. Nonetheless, we still call them *discordant* pairs in this thesis to unify the word used to refer pairs that support SVs.

- Graph-based: Marshall et al. [54] proposed a new method CLEVER based on the graph theory. CLEVER constructs a graph where a node represents an alignment of a read pair and the reference genome, while an edge means that connected alignments potentially support the same allele. In this graph, a clique corresponds to a set of pairs supporting the same allele. CLEVER detects SVs by finding maximal cliques (max-cliques). CLEVER has an ability to find more than one max-clique overlaping each other, each of which supports a different allele. Therefore CLEVER can distinguish more than one SV located at the same locus, for example, two deletions of different sizes in a diploid genome.

3.1.2 Read Depth (RD)

If coverage changes at some position in the genome, this indicates a copy number variation [1, 8].

3.1.3 Split Read (SR)

If an alignment of a read and the genome includes only a part of the read, this indicates existence of a SV and a position of a breakpoint [87]. Here, a *breakpoint* is the boundary between a region affected by some SV and its unaffected flanking region.

3.1.4 Sequence Assembly (AS)

If the coverage is sufficient, assembling NGS sequences around an SV reveals the exact sequence around the SV and the positions of breakpoints [46, 70].

3.1.5 Our Contribution

The most popular signature used to detect SVs is threshold-based RP. Methods based on this signature can detect SVs from a small number of discordant read pairs; therefore threshold-based RP methods can be applied to low-coverage data. However, threshold-based RP methods localize SVs only to regions surrounded by discordant read pairs, thus causing some ambiguity. For RD methods, the problem of resolution is much bigger. Because RD methods involve calculation of coverage in windows of a fixed size, its resolution cannot be finer than the window size. Methods based on the SR signature can determine positions of breakpoints up to base-pair-level (bp-level) resolution if there are reads covering the breakpoints. However, such reads might not exist, in particular when coverage is low, because of unevenness of coverage or repeat elements to which reads cannot be aligned uniquely. Moreover, because such a split alignment is shorter than a read itself, careful analysis is required to avoid spurious matches. If coverage is sufficiently high, AS methods would ultimately reveal the exact positions of SVs at bp-level resolution. Although extremely deep sequencing can be conducted by targeted sequencing[51], it is still expensive to obtain paired reads of high coverage over the entire genome so that assembly can be performed. In fact, a previous study has indicated that the sensitivity of AS methods is rather low (see the supplementary table 6B of Mills et al. [58]).

Because these signatures have their own advantages and disadvantages, it is desirable to combine more than one method [55]. In fact, several methods that use more than one signature have also been proposed [23, 89]. In combined approaches, we should integrate SV signatures that are independent of each other. ChopSticks exploits concordant pairs, which have not been fully exploited so far, as an additional independent signature for the purpose of improving the resolution of the positions of detected SVs.

3.2 Strategy for Resolution Improvement

3.2.1 Expectation of Resolution

We define a *discordant read* as a read of a discordant pair and a *concordant read* as that of a concordant pair. Among the two reads of a pair, the one mapped upstream is called an *upstream read* and the other is called a *downstream read* in this thesis. Let c be the depth of coverage. Assume that the positions of read pairs are uniformly random over the genome, and that the length r of each read is a fixed constant. Let $q(c)$ be the probability that there is no read pair whose upstream read begins at a given base in the genome. Suppose that there are N read pairs uniquely mapped to a genomic sequence of length G. According to a classical analysis [36],

$$q(c) \quad = \quad \left(1 - \frac{1}{G}\right)^N \approx e^{-N/G} = e^{-c/2r}. \tag{3.1}$$

Hereafter, we just write q instead of $q(c)$ for simplicity. In threshold-based RP approaches, the position of an upstream end of a deletion is determined by the upstream discordant read that is the closest to the breakpoint. Let b be the position of an upstream end of a deletion, Δ_b be the distance between b and the closest upstream discordant read, and d be the distance between paired reads. We assume that d is a constant.

Lemma 3.1. *Let $E[\Delta_b|b,c]$ be the expectation of Δ_b given that b is detected and the coverage is c. Then,*

$$E[\Delta_b|b,c] \quad = \quad \frac{1-q}{1-q^{d+1}} S(q,d), \tag{3.2}$$

where

$$S(q,d) = \sum_{j=0}^{d} j q^j = \frac{q - (d+1)q^{d+1} + dq^{d+2}}{(1-q)^2}.$$

Proof. Because the resolution at downstream ends of deletions can be estimated symmetrically, we only analyze the resolution at upstream ends. Let P_b be the probability that a breakpoint b is successfully included in a deletion call by a threshold-based RP method. If b is detected, there exists an upstream discordant read within d bases from b. Therefore,

$$P_b = 1 - q^{d+1}.$$

We derive the expected distance between the true ends of deletions and the predicted ones in a manner similar to Bashir's analysis[6]. For $0 \leq j \leq d$, Bashir et al. defined A_j as an event in which b is detected and an upstream read of a discordant pair is exactly j bases upstream of b. The probability that A_j occurs is

$$\Pr(A_j) \quad = \quad (1-q)q^j.$$

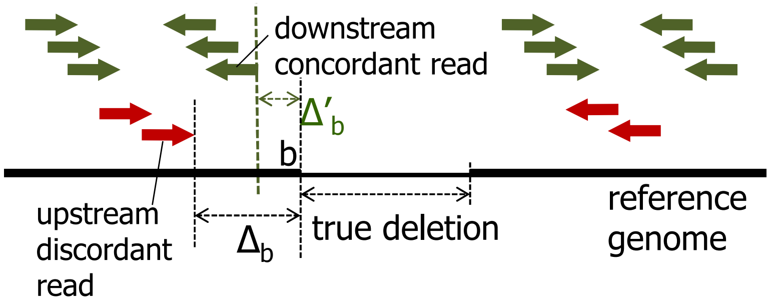


Figure 3.1: Resolution improvement by exploiting concordant read pairs. Schematic illustration of the key idea of our method ChopSticks. Unlike conventional SV detection methods based only on discordant pairs whose mapped distances were not close to the expectation, ChopSticks uses concordant read pairs as well. There is a chance that there is a concordant read closer to the boundary of the deleted region (breakpoint) than any discordant reads. Such a concordant read localizes the predicted position of the breakpoint, and therefore it contributes to achieving a high resolution. In this figure, b is the upstream end of a true deletion, Δ_b is the distance between the upstream end of a true deletion and that of a deletion call by threshold-based read-pair (RP) methods. Similarly, Δ'_b is defined for our method. The expected values of Δ_b and Δ'_b are given by Equations (3.2) and (3.4), respectively.

Consequently,

$$E[\Delta_b|b,c] = \frac{1}{P_b} \sum_{0 \le j \le d} j \Pr(A_j) = \frac{1-q}{1-q^{d+1}} S(q,d). \tag{3.3}$$

□

3.2.2 Expectation of Improved Resolution

We can obtain better resolution by using concordant reads in addition to discordant reads, because there is a chance that there exists a concordant read closer to b than any upstream discordant read (Figure 3.1). Such a read can contribute to the localization of the position where b can exist.

Lemma 3.2. *Let Δ'_b be the distance between b and the closest read in the upstream of b, and let $E[\Delta'_b|b,c]$ be the expectation of Δ'_b given that b is detected and the coverage is c. Then,*

$$E[\Delta'_b|b,c] \quad = \quad \frac{1}{1-q^{d+1}} \Big((1-q^2) S(q^2,d) - q^{d+1}(1-q) S(q,d) \Big). \tag{3.4}$$

Proof. Let A'_j be an event wherein b is detected and the closest read upstream of b is exactly j bases apart. When A'_j occurs, there are two mutually exclusive cases: (i) at least one of the closest reads is an upstream discordant read or (ii) all the closest reads are concordant reads. In the latter case, we have to consider the joint probability of the following events.

- A concordant read exists at j bases upstream of b, the probability of which is $1-q$.
- No read nearer than the closest concordant read exists, the probability of which is q^{2j}.

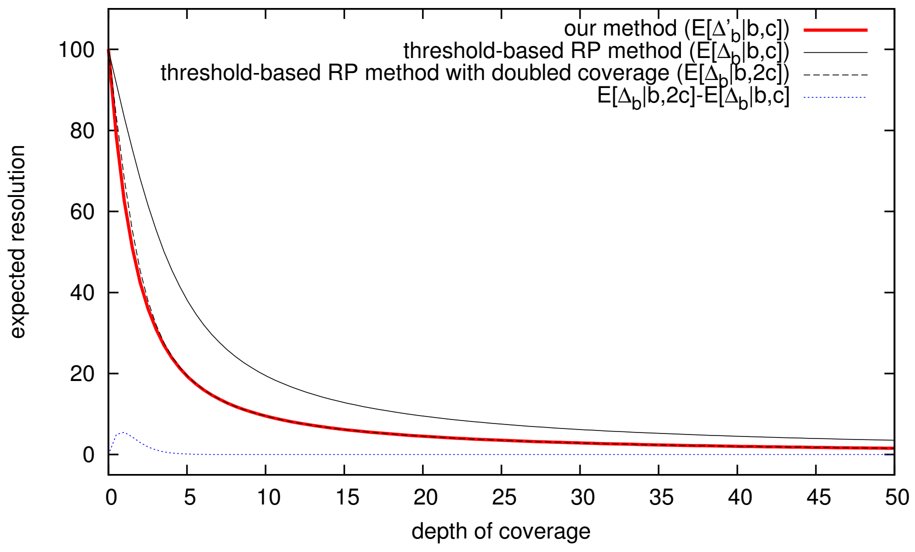


Figure 3.2: Expected resolutions of ChopSticks and threshold-based RP methods. The expected resolution of our method ($E[\Delta'_b|b,c]$) is shown by a thick red line, that of threshold-based RP methods ($E[\Delta_b|b,c]$) is shown by a thin solid black line, and that of threshold-based RP methods with double coverage ($E[\Delta_b|b,2c]$) is shown by a dashed black line. The difference between $E[\Delta'_b|b,c]$ and $E[\Delta_b|b,2c]$ is also shown by a dotted blue line. As the coverage goes away from zero, the resolution obtained by our method quickly outperforms that of normal RP methods. It is also clear that the resolution of our method is very close to that of threshold-based RP methods with double coverage. The difference approaches zero when coverage approaches zero or infinity, as indicated by the blue dotted line. $E[\Delta_b|b,c]$, $E[\Delta'_b|b,c]$, and $E[\Delta_b|b,2c]$ are given by Equations (3.2), (3.4), and (3.7), respectively. In this figure, $d = 200$ and $r = 100$.

- No discordant read exists at j bases upstream of b, the probability of which is q.
- There must exist an upstream read of discordant pairs whose alignment ends in a region that is $j+1$ to d bases upstream of b so that b is successfully included in a deletion call, the probability of which is $1 - q^{d-j}$.

Therefore,

$$\begin{aligned}\Pr(A'_j) &= (1-q)q^{2j} + (1-q)q^{2j}q(1-q^{d-j}) \\ &= (1-q^2)q^{2j} - q^{d+1}(1-q)q^j.\end{aligned}$$

Consequently,

$$\begin{aligned}E[\Delta'_b|b,c] &= \frac{1}{P_b}\sum_{0\le j\le d} j\Pr(A'_j) \\ &= \frac{1}{1-q^{d+1}}\Big((1-q^2)S(q^2,d) - q^{d+1}(1-q)S(q,d)\Big). \qquad (3.5)\end{aligned}$$

□

As shown in Figure 3.2, the expected resolution of our method is significantly superior to that of threshold-based RP methods, which only use discordant pairs.

The achieved resolution is quite close to that of threshold-based RP methods but with double coverage, which we confirmed theoretically.

Theorem 3.1. *The expectation $E[\Delta_b'|b,c]$ is a weighted sum of $E[\Delta_b|b,2c]$ and $E[\Delta_b|b,c]$. To be more precise, the following equation holds:*

$$E[\Delta_b'|b,c] \;=\; (1+q^{d+1})E[\Delta_b|b,2c] - q^{d+1}E[\Delta_b|b,c]. \tag{3.6}$$

Proof. From Equation (3.1), $E[\Delta_b|b,2c]$ can be obtained by replacing q with q^2 in Equation (3.2):

$$E[\Delta_b|b,2c] \;=\; \frac{1-q^2}{1-q^{2(d+1)}}S(q^2,d). \tag{3.7}$$

From Equations (3.2), (3.4), and (3.7), Equation (3.6) can be obtained. □

When $c \to 0$, $E[\Delta_b|b,2c]$ and $E[\Delta_b|b,c]$ approach $d/2$, which is the expected resolution when a deletion is detected with only one read pair. Therefore $E[\Delta_b'|b,c]$ also approaches $d/2$ when $c \to 0$. On the other hand, when c approaches infinity, $E[\Delta_b'|b,2c]$ approaches $E[\Delta_b|b,2c]$ because $q^{d+1} \to 0$. In summary,

Theorem 3.2. *$E[\Delta_b'|b,c]$ is asymptotically equal to $E[\Delta_b|b,2c]$ when $c \to 0$ or $c \to \infty$.*

Proof. First, we consider a case where $c \to 0$. Because $q \to 1$ by Equation (3.1),

$$S(q,d) \to \sum_{j=0}^{d} j = \frac{d(d+1)}{2}.$$

Besides,

$$\frac{1-q}{1-q^{d+1}} = \frac{1}{1+q+q^2+\cdots+q^d} \to \frac{1}{d+1}.$$

Therefore, all of $E[\Delta_b'|b,c]$, $E[\Delta_b|b,2c]$, and $E[\Delta_b|b,c]$ approach $d/2$ by Equation (3.6). On the other hand, when $c \to \infty$, q^{d+1} approaches 0. In consequence, the right hand side of Equation (3.6) approaches $E[\Delta_b|b,2c]$ when $c \to 0$ or $c \to \infty$. □

3.2.3 Trimming of Deletion Calls to Improve Resolution

If all regions existing in the reference genome were covered by at least one read and there were absolutely no reads mapped to regions of homozygous deletions, the resolution of deletion calls could be quite easily improved by just trimming the ends of deletion calls that are covered by alignments of reads. Obviously, such a simple assumption does not hold in practical situations. First, coverage might be zero even in regions that actually exist in the genome, because no reads are obtained therein owing to the unevenness of the coverage or because reads cannot be uniquely mapped owing to repeat elements. Second, there might exist erroneous alignments in deleted regions because of incidental sequence similarity. Therefore, we developed the algorithm ChopSticks to carefully trim the ends of deletion calls (Figure 3.3). ChopSticks recognizes high-coverage regions close to the ends of deletion calls even if they are fragmented, and it repeatedly excludes the high-coverage regions from deletion calls. ChopSticks uses two parameters, k and f. The k parameter is a threshold used to distinguish high-coverage regions from low-coverage ones, and f determines the threshold of joint coverage of regions excluded from a deletion call.

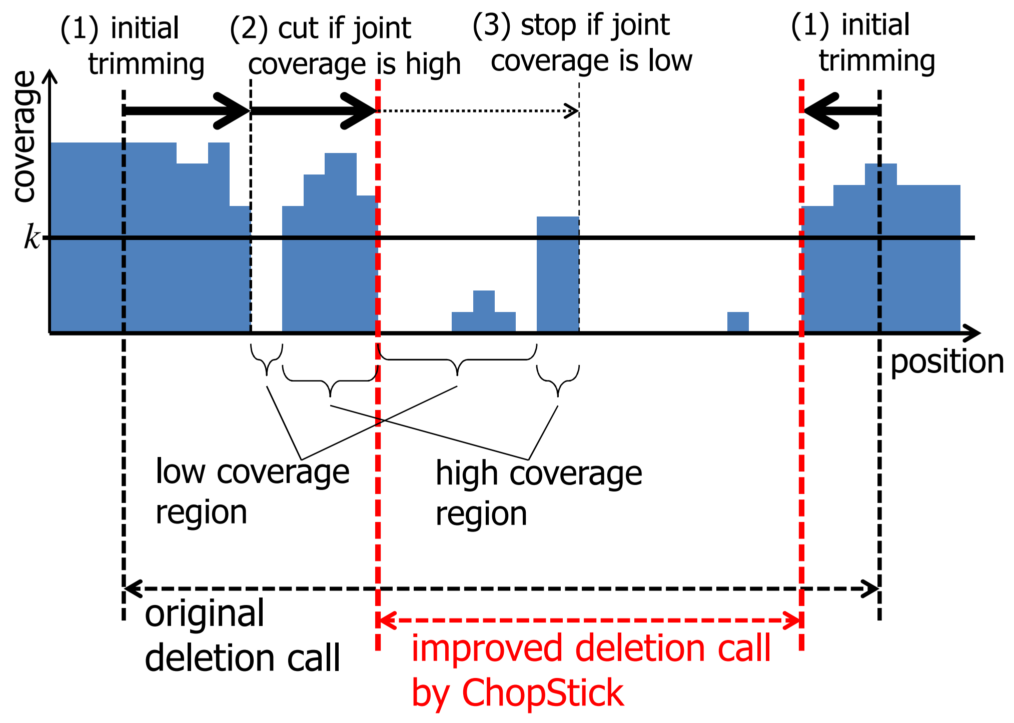


Figure 3.3: Overview of trimming algorithm of ChopSticks. Schematic illustration of the trimming algorithm of ChopSticks. ChopSticks trims ends of deletion calls that are not likely to be parts of deletions, according to their coverage. First, it trims high-coverage regions at the ends of deletion calls. Here, a *high-coverage region* is a region whose coverage is greater than a given parameter k. Second, it recognizes a high-coverage region separated by a low-coverage region and trims these regions if their joint coverage is deeper than kf, where f is another parameter. The second step is repeatedly conducted until the joint coverage becomes less than kf.

ChopSticks repeatedly recognizes a high-coverage region in a deletion call that is likely a continuation of a high-coverage region outside the deletion. We show in Figure 3.4 the trimming algorithm executed by ChopSticks for upstream ends. Here is a brief description of the algorithm:

Line 2: Skip a high-coverage region at the end of the deletion call.

Lines 6–9: Go through a low-coverage region.

Lines 10–13: Go through a high-coverage region.

Line 14: If the joint coverage is low, exit the loop.

Line 17: Trim regions which the algorithm has gone through.

Trimming of the downstream ends is conducted symmetrically.

Our implementation of ChopSticks is available on the Internet[1].

[1] `https://github.com/toyasuda/ChopSticks`

```
1   $x := 0$
2   **while**($x < L$ and $c[x] \geq k$) { $x := x + 1$ }
3   $y := x$
4   **while**($x < L$) {
5       $s := 0$
6       **while**($c[x] < k$ and $x < L$){
7           $s := s + c[x]$
8           $x := x + 1$
9       }
10      **while**($c[x] \geq k$ and $x < L$){
11          $s := s + c[x]$
12          $x := x + 1$
13      }
14      **if**($s/(x - y) < kf$) **goto** Line 17
15      $y := x$
16  }
17  Trim the first $y$ bases of the deletion call
```

Figure 3.4: Pseudo code of trimming algorithm. Pseudocode of the trimming algorithm of ChopSticks. Here, L is the length of the deletion call being processed, k is a threshold used to discriminate high-coverage regions from low-coverage ones, and f is a parameter that determines the threshold of the coverage of regions to be trimmed. The variable x represents the position of the base being examined, and the variable y represents the length of a region to be trimmed. The value $c[x]$ is the coverage at the x-th base in the deletion call, while s keeps the sum of $c[x]$ values.

3.3 Computational Experiment

ChopSticks is especially valuable when target SVs are expected to be homozygous as those of inbred mice[2] whose genomes are homozygous at virtually all loci.

To evaluate the power of ChopSticks in improving the resolution of deletion calls, we conducted computational experiments. Let the *upstream difference* of a deletion call be $x - y$, where x is the position of the upstream end of the true deletion and y be that of the deletion call. Similarly, let the *downstream difference* of a deletion call be $y' - x'$, where x' is the position of the downstream end of the true deletion and y' is that of the deletion call. By definition, the closer to zero a difference is, the better. A positive difference value indicates that the called breakpoint is outside the true deletion, whereas a negative value indicates that it is inside the true deletion.

3.3.1 Data for Computational Experiments

To evaluate ChopSticks, the results of ChopSticks have to be compared with the positions of true deletions. Therefore we need NGS reads of a genome whose SVs against the reference genome are known up to bp-level resolution. We evaluated ChopSticks using the following two data sets.

Simulated NGS Reads

We artificially introduced deletions and insertions into the mm9 reference genome and then generated simulated NGS reads using the modified genome. To obtain

[2] `http://jaxmice.jax.org/type/inbred/index.html`

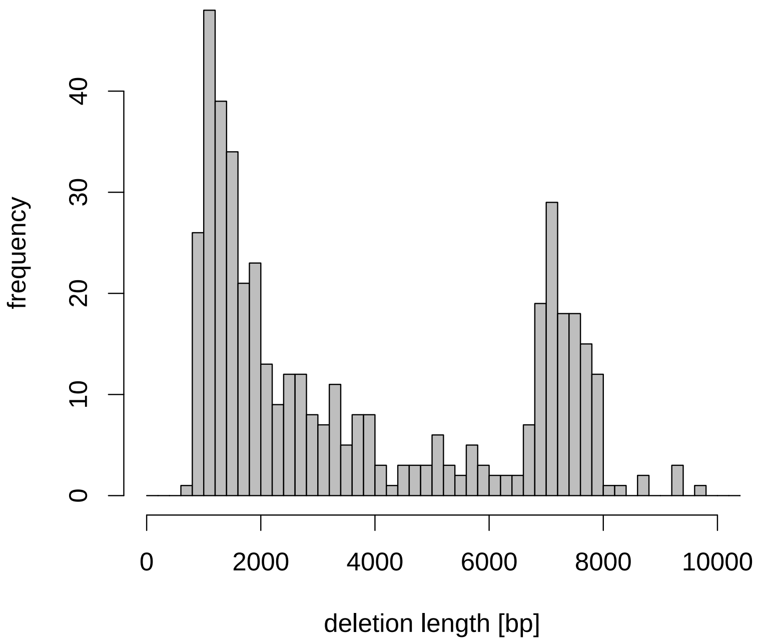


Figure 3.5: Distribution of deletion lengths in our simulation.

data as realistic as possible, we built a simulated genome sequence using SV annotations generated by Quinlan et al. [70], which are available from the dbVar database under accession no. nstd19. First, we deleted regions annotated as deletions in nstd19 from the mm9 reference genome sequence of chromosome 1. We show the distribution of lengths of deletions in Figure 3.5. Second, we inserted fragments consisting of randomly chosen bases so that the number and the distribution of lengths of inserted fragments were the same as those of deletions, assuming that the genome to be analyzed and the reference genome are affected symmetrically by deletions and insertions. Third, we introduced random single nucleotide substitutions with a probability of 1.0×10^{-4} at each base. Finally, we generated paired reads from the modified genome sequence so that the read length was 100 bp and the average and the standard deviation of distances of paired reads were 200 bp and 50 bp, respectively. We generated four sets of simulated NGS reads whose depth of coverage were 2, 5, 10, 15, and 20, respectively.

Real Illumina Reads of DBA/2J

We generated our own bp-level deletion calls by using publicly available Sanger reads of the DBA/2J strain. From the NCBI trace archive, we retrieved all 7,998,826 Sanger reads of whole-genome shotgun sequencing for the DBA/2J strain. We mapped these Sanger reads to chromosome 1 of mm9 by using MegaBLAST [90], and we searched for Sanger reads that were split into two parts and aligned uniquely on the same strand and in the right order. There were 763 reads that indicated deletions whose lengths were at least 50 bp. By

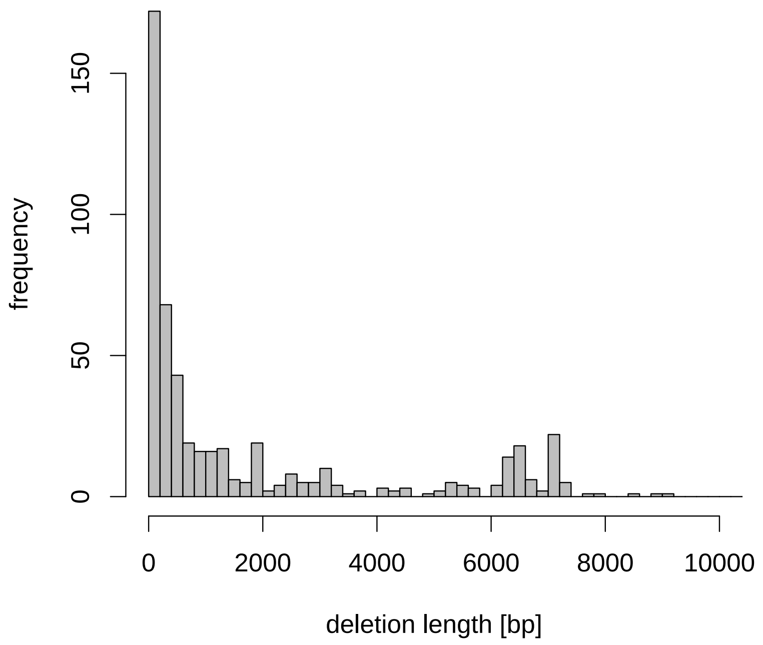


Figure 3.6: Distribution of deletion lengths detected with Sanger reads.

merging redundant ones, we obtained 525 deletion calls. We show the distribution of lengths of these deletions in Figure 3.6. Among these, 83 deletions overlapped with single BreakDancer deletion calls, each of which in turn overlapped with our deletion calls. We evaluated ChopSticks by using these 83 deletion calls. Our deletion calls based on Sanger reads are in the dbVar database under accession no. nstd70. NGS sequences of the DBA/2J strain generated by Quinlan et al. are available in the SRA database [72] under accession no. SRA010027.

3.3.2 Mapping to the Genome

We mapped paired reads to the mm9 reference genome sequences of *Mus musculus* using BWA version 0.5.9[45] with default parameters. The target genome sequences involved in our experiment included all chromosomes of mm9 except chromosome Y, assuming cases where a female mouse was analyzed [59, 70].

Simulated NGS Sequences

To focus on uniquely mapped reads for BreakDancer, MoDIL, CLEVER, and ChopSticks, we removed paired reads if the mapping quality (MAPQ) score was zero for at least one of the two reads of a pair. For CNVnator and Pindel, we used the result of BWA without filtering. We show the total length of reads and the number of aligned reads in Table 3.1.

Table 3.1: On the third row, we only counted read pairs whose reads were both mapped uniquely.

		Number of reads	
Depth of coverage	Total number of bases	all	mapped
2	394,391,200	3,943,912	3,677,398
5	985,978,000	9,859,780	9,194,942
10	1,971,956,000	19,719,560	18,391,288
15	2,957,934,000	29,579,340	27,587,970
20	3,943,912,000	39,439,120	36,783,348

Table 3.2: Summarized statistics of NGS reads of the DBA/2J strain [70] and their alignments to mm9.

Total number of bases	13,050,980,662
Number of reads	330,462,408
Reads of uniquely mapped pairs	149,021,716
Reads of uniquely mapped pairs (chromosome 1)	10,316,525

Real DBA/2J Sequences

We split the data set of NGS reads into 275 parts, and we mapped each of them with an independent BWA process and merged the results. Then we removed reads whose MAPQ score was zero for at least one of the two reads of a pair. We show the total length of reads and the number of aligned reads in Table 3.2.

3.3.3 Simulated Reads

In the first experiment, we evaluated ChopSticks with simulated NGS reads for which all SVs were known up to bp-level resolution. SV analysis was conducted by using SV detection tools from each of categories described at the beginning of this chapter: BreakDancer[10] of threshold-based RP methods, MoDIL[40] of distribution-based RP methods, CLEVER[54] of graph-based RP methods, CNVnator[1] of RD methods, and Pindel[87] of SR methods. After that, we applied ChopSticks to their results.

Before applying ChopSticks, we examined the ability of SV detection tools to detect 460 deletions in chromosome 1 of the simulated mouse genome. We say that a deletion call is *correct* if it overlaps exactly one true deletion while the true deletion in turn overlaps exactly one deletion call. We show the number of called and correct SV calls in Table 3.3. We also show their *recall* (the number of correct deletion calls divided by the number of true deletions) and *precision* (the number of correct deletion calls divided by the number of all deletion calls) in Figure 3.7. The recall of BreakDancer and CLEVERwas relatively good for all of tried coverage values, whereas the recall of Pindel was satisfactory only when coverage was high. The recall of MoDIL was low for all coverage values tried. Although almost all deletions called by these methods were correct, CNVnator generated numerous false positives (Table 3.3). Because ChopSticks is developed to correct breakpoints outside true deletions, we counted the number of deletion calls that cover the whole of true deletions. As shown in Figure 3.9, most of

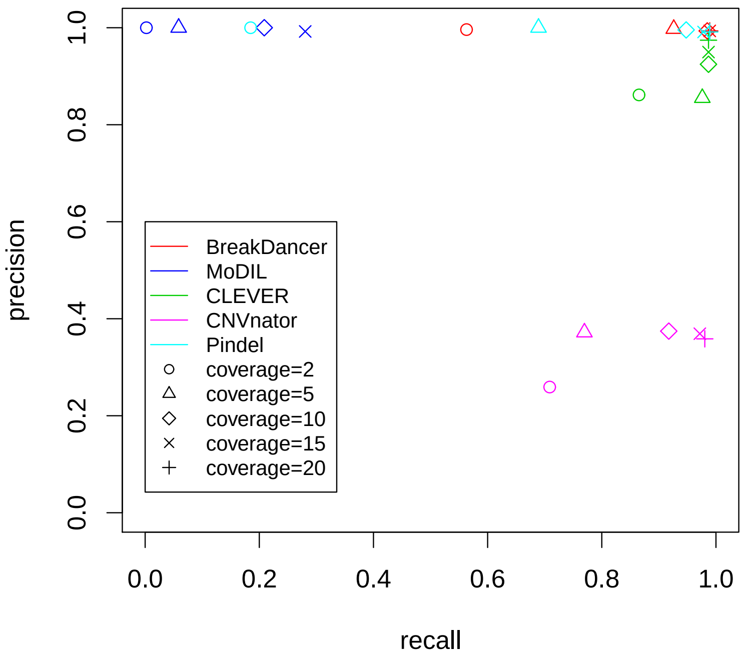


Figure 3.7: Recall and precision of results of SV detection tools. BreakDancer and CLEVER achieved relatively good recall for all coverage, while recall of MoDIL was low. Although recall of CNVnator was not bad, its precision was low. The recall of an SR method Pindel was good when coverage was high, but it was insufficient when coverage was low.

the deletion calls by MoDIL, CNVnator, and Pindel covered the whole of true deletions. However, a significant portion of BreakDancer and CLEVER results did not cover the whole of true deletions. Note that ChopSticks is harmless to these deletion calls because ChopSticks does not trim them when there are no alignments in true deletions.

Next, we applied ChopSticks to the results of SV detection tools. After that, we examined how well the resolution of deletion calls was improved. We tested ChopSticks for $k = 1, 2, \dots, 5$ and $f = 0.1, 0.2, \dots, 1.0$. We evaluated differences at both the upstream and downstream ends of deletions and found that the results were similar. Therefore we only present the results at upstream ends.

Resolution Improvements for BreakDancer Deletion Calls

As shown in Figure 3.10, the resolution of deletion calls was clearly improved by using ChopSticks. The original BreakDancer results was successfully corrected, which is also clear in Figure 3.11. When coverage was low, the resolution was well improved for small k values. When coverage was high, the resolution was also improved for large k values. Therefore, when the coverage is high, we recommend using large k values to ignore erroneous alignments. As shown in Figure 3.12, ChopSticks worked well regardless of deletions length.

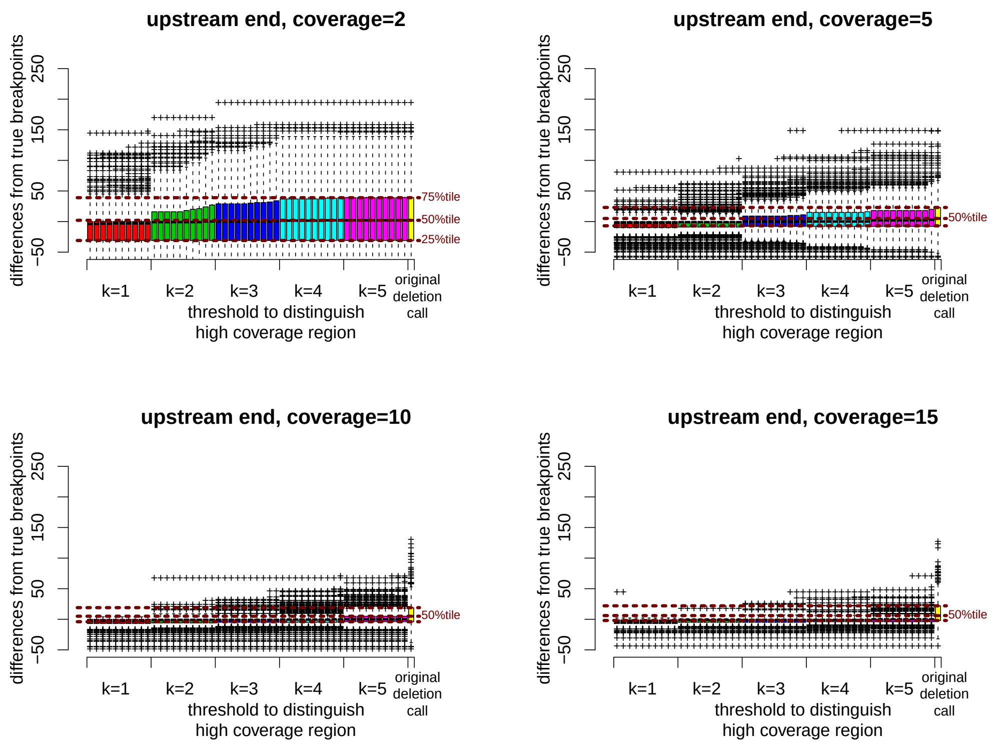


Figure 3.8: CLEVER results improved by ChopSticks. Box-and-whisker plots of upstream differences of deletion calls obtained by CLEVER and those improved by ChopSticks. The differences were successfully corrected. Note that a significant portion of breakpoints predicted by CLEVER were inside the true deletion. Nonetheless, ChopSticks selectively trimmed predicted breakpoints outside true deletions, and left those inside untouched.

Resolution Improvements for MoDIL Deletion Calls

As shown in Figure 3.13, the resolution of deletion calls was also improved by using ChopSticks. We omitted evaluation of MoDIL for coverage=20 because MoDIL was very slow. (See Section 3.2).

Resolution Improvements for CLEVER Deletion Calls

The resolution of deletion calls was also improved by using ChopSticks. As mentioned above, deletion calls of CLEVER do not always cover the whole of true deletions. Nonetheless, as shown in Figure 3.8 and 3.14, ChopSticks successfully improved resolution of CLEVER results by selectively correcting predicted breakpoints outside true deletions.

Resolution Improvements for CNVnator Deletion Calls

Because RD methods call SVs by examining coverages in windows of a fixed size, the positions of breakpoints predicted by the RD methods have unavoidable ambiguity and they might be either inside or outside true deletions. Because ChopSticks assumes that predicted breakpoints are outside true deletions, we applied ChopSticks after we expanded deletion calls of CNVnator at both ends by the window size. As shown in Figure 3.15, the results of CNVnator were successfully improved. This result indicates that ChopSticks is also available for

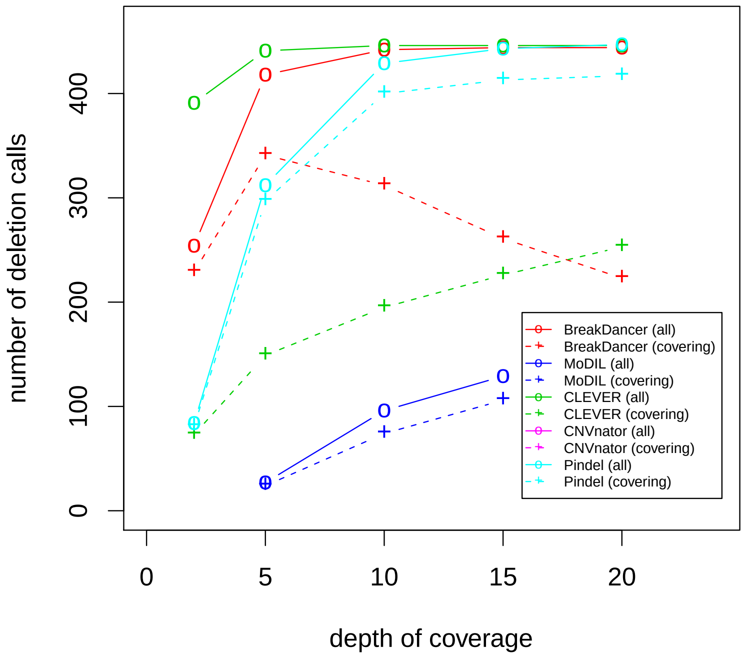


Figure 3.9: Number of deletion calls covering the whole of true deletions. Solid lines and circles show the number of all deletion calls generated by each tool, whereas dashed lines and '+' symbol show the number of deletion calls covering the whole of true deletions. Most of deletion called of MoDIL, CNVnator (expanded by the window size), and Pindel covered the whole of true deletions. Many CLEVER results did not always contain the whole of true deletions, while median of the distribution of predicted breakpoints was close to the true breakpoints as shown in Figure 3.8. BreakDancer results for high coverage data did not always contained true deletions either. Predicted breakpoints of BreakDancer approach true breakpoints as the depth of coverage increases, and sometimes intruded into true deletions when coverage was high.

RD methods in addition to RP methods.

Results of ChopSticks applied to Pindel deletion calls

Owing to the SR signature that allows Pindel to detect SVs at bp-level resolution, the positions of breakpoints obtained with Pindel were quite accurate. When ChopSticks was applied to the results of Pindel, the results became slightly worse than the original Pindel results, as shown in Figure 3.16, although differences remained close to zero in most cases. Note that the recall of Pindel was not satisfactory when coverage is low, as shown in Figure 3.7. ChopSticks is useful in cases where deletions missed by Pindel are analyzed.

3.3.4 Real Illumina Reads of DBA/2J

In the second experiment, we evaluated ChopSticks using the real NGS sequences of Quinlan et al. [70]. The sample was taken from a female mouse of the DBA/2J

Table 3.3: The values to the left of "/" are the numbers of *correct* deletion calls, where a *correct* deletion call is the one that overlaps with exactly one true deletion, which, in turn, only overlaps with the deletion call; the values to the right of "/" are the numbers of all deletion calls. BreakDancer and CLEVER results were good in both sensitivity and specificity. CNVnator generated numerous false positives, while Pindel suffered from low coverage. MoDIL missed lots of deletions.

	Depth of coverage				
SV caller	2	5	10	15	20
BreakDancer	259/260	426/427	453/456	455/458	455/458
MoDIL	1/1	27/27	96/96	129/130	–/–
CLEVER	398/462	449/525	454/491	454/478	454/466
CNVnator	326/1,258	354/952	422/1,127	447/1,211	451/1,258
Pindel	85/85	317/317	436/438	450/454	456/456

strain, whose genome contains SVs against the reference genome of the C57BL/6J strain [59]. The read sequences were available from the NCBI Sequence Read Archive (SRA) database [72]. The accession number of the read sequences is SRA010027. To evaluate the results of ChopSticks, we need bp-level SV annotations of DBA/2J as well. Therefore we generated deletion calls at bp-level resolution using Sanger reads in a manner similar to that of Quinlan et al. See Methods for details. Our deletion calls will soon be available at the dbVar database under accession no. nstd70.

We tried the five SV detection tools used in the previous experiment, and found that MoDIL, CNVnator and Pindel missed the most of deletions detected with Sanger reads. These methods seemed to suffer from the low depth of coverage and short read lengths. Therefore, we hereafter only describe results of ChopSticks applied to BreakDancer and CLEVER results.

Resolution Improvements for BreakDancer Deletion Calls

Figure 3.17 shows the differences between BreakDancer results and those improved by using ChopSticks. As the previous experiment where simulated NGS reads were used, the differences obtained with real NGS reads were reduced. The median and differences less than the median clearly shifted toward zero, which is also clear in Figure 3.18. Although ChopSticks trimmed some deletion calls into those based on Sanger reads when $k = 1$ or $k = 2$ and f was small, this problem quickly disappeared as k or f became larger. No correlation between deletion lengths and the performance of ChopSticks were observed ($r^2 = 0.021$). Although we generated 525 deletion calls by using Sanger reads, only 83 of them were found by BreakDancer. There were at least two reasons for this difference. First, it is difficult to find small deletions because read pairs spanning small deletions might not be recognized as discordant pairs. Second, a lot of deletion calls based on Sanger reads had fewer than two NGS-read pairs spanning them. Such deletion calls would be missed because BreakDancer deletion calls must be supported by at least two pairs when the default parameters are used, in order to reduce false positives. For this data set, 82 of all 83 deletion calls generated by BreakDancer contained deletions predicted with Sanger reads.

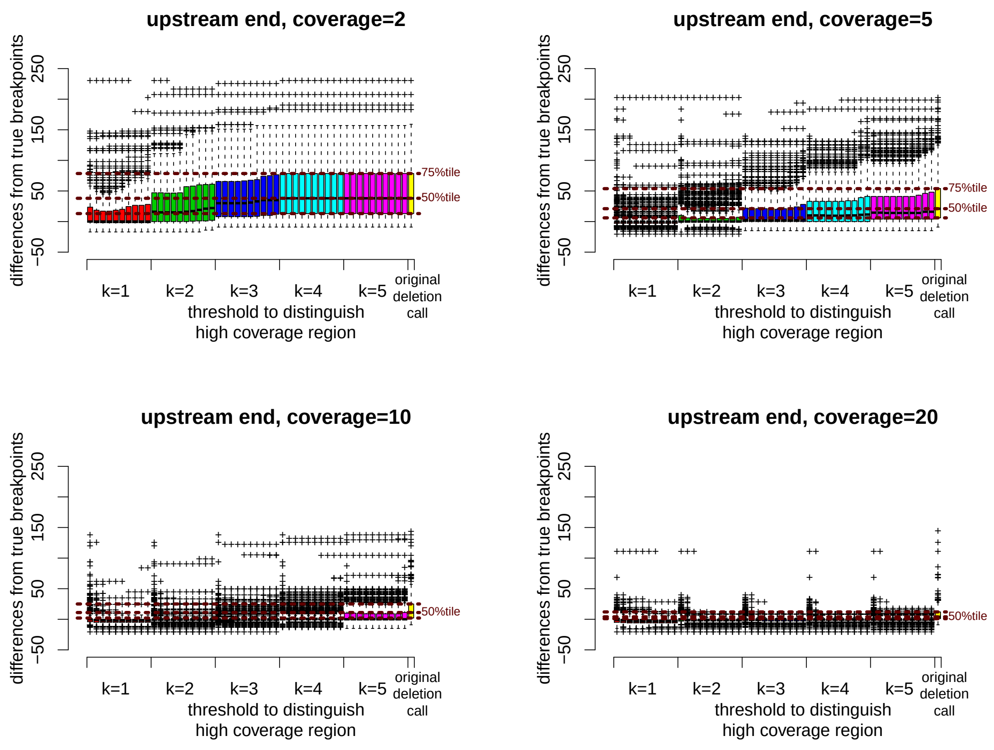


Figure 3.10: BreakDancer results improved by ChopSticks. Box-and-whisker plots of upstream differences of deletion calls obtained by BreakDancer and those improved by ChopSticks. The red, green, blue, light blue, and magenta boxes correspond to k values of 1, 2, 3, 4, and 5, respectively, and the rightmost yellow box corresponds to the original results of BreakDancer. Among boxes of the same color, from left to right, $f = 0.1, 0.2, \ldots, 1.0$. Brown horizontal dashed lines indicate the values of 25%, 50%, and 75% tiles of differences of original deletion calls from below to above, respectively. The results in this figure indicate that ChopSticks clearly improved the resolution of the original BreakDancer results. When the coverage was low, small k values were effective in improving the resolution. When coverage was high, the resolution was also improved for large k values. Therefore, when the coverage is high, we recommend using large k values to avoid erroneous alignments of NGS reads and the genome. We omitted the results for coverage=15 because they were similar to those for coverage=20.

Resolution Improvements for CLEVER Deletion Calls

CLEVER detected much more (347) deletions than BreakDancer. The results of CLEVER were also improved by ChopSticks as shown in Figure 3.19, where the peak around zero became stronger. However, it was difficult for ChopSticks to correct positions of breakpoints when they were away from those predicted with Sanger reads by hundreds of bases.

3.3.5 Parameters for SV Detection Methods and Evaluation of Their Results

We executed BreakDancer with default parameters, and Pindel with an expected template size of 432 bp because the median fragment size was 432 bp according to Quinlan et al. [70]. For CNVnator, we tested three window sizes: 50 bp, 100 bp, and 200 bp. Because the recall of window size 50 bp outperformed those of

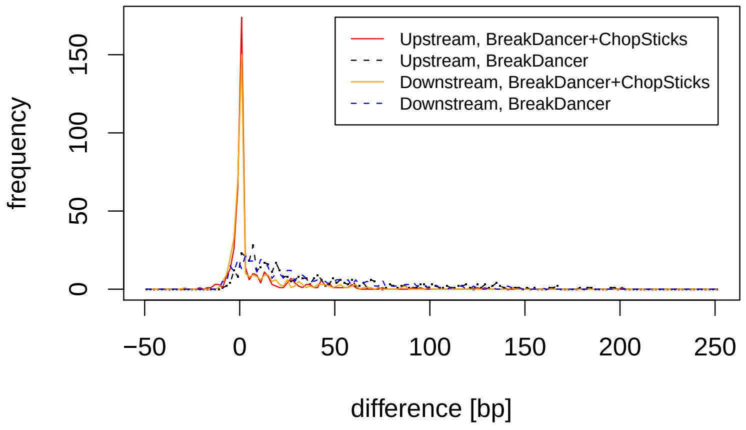


Figure 3.11: Distribution of differences of BreakDancer results and those improved by ChopSticks. The distribution of differences of ChopSticks results concentrated around zero, whereas that of BreakDancer results had long tail in 0–50 bp. Here, $k = 2$, $f = 0.5$, and coverage=5. Each frequency corresponds to the number of differences in bins of 2 bp.

window sizes 100 bp and 200 bp for our simulated data when coverage was 2, we used results of window size 50 bp for evaluation. Because CLEVER tends to generate deletion calls duplicatedly with slightly different positions, we chose the best one for those overlapping with true deletions in order to estimate the upper limit of the accuracy of CLEVER. We divided the chromosome 1 of mm9 into 5.1 Mbp fragments where franking fragments share 0.1Mbp and applied MoDIL to each fragments, because MoDIL was quite slow as reported previously [54]. We omitted evaluation of MoDIL for coverage=20. To compare the positions of true and predicted deletions, BEDTools [71] were used.

3.4 Summary

We have presented a new method called ChopSticks to improve the resolution of predicted positions of deletions. The key idea is to exploit both concordant read pairs and discordant ones. According to our theoretical analysis, the resolution of our method is quite similar to that of threshold-based RP methods but with double coverage. In an experiment on simulated NGS reads, ChopSticks clearly improved the results of BreakDancer, MoDIL, CLEVER, and CNVnator. Although the resolution of Pindel results is quite high, ChopSticks works well even for low-coverage data where recall of Pindel is not sufficient. The effectiveness of ChopSticks was also confirmed by performing an experiment on real Illumina reads. Despite a number of methods proposed for detecting SVs [55, 58, 79], there is no one-stop method that simultaneously achieves high sensitivity, high specificity, high resolution, and robustness for low-coverage data. Therefore a combination of SV detection methods is required, and ChopSticks can play an important role because it uses new independent information ignored in other methods.

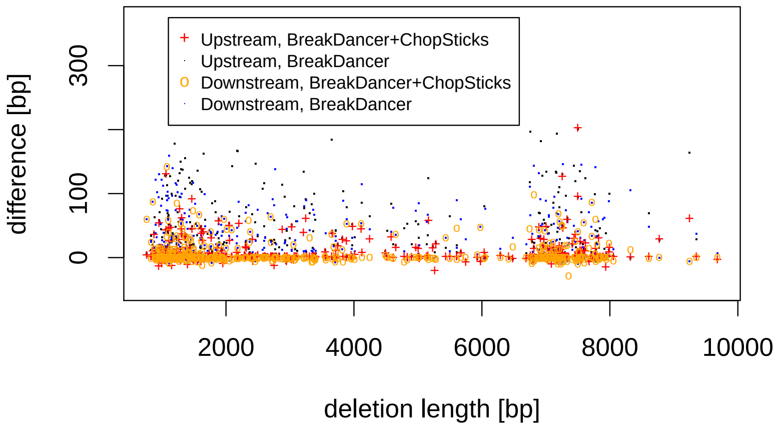


Figure 3.12: Scatter plot of deletion lengths and differences of deletion calls. No correlation between deletion lengths and differences of differences of ChopSticks was observed ($r^2 = 0.056$). ChopSticks worked well regardless of deletion lengths. Here, $k = 2$, $f = 0.5$, and coverage=5.

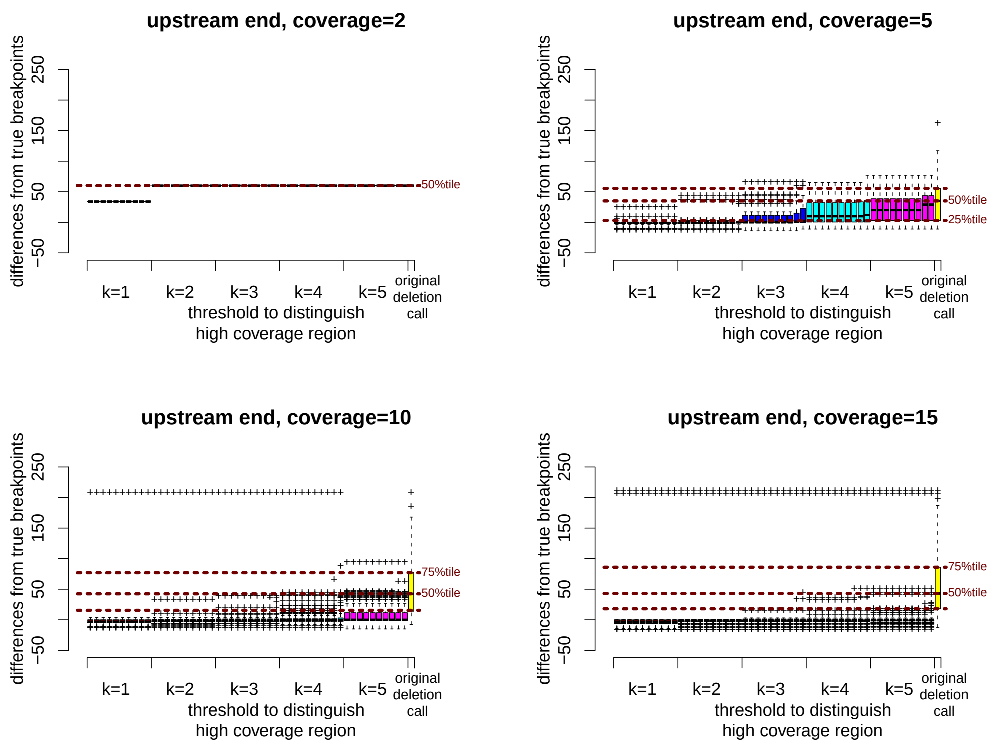


Figure 3.13: MoDIL results improved by ChopSticks. Box-and-whisker plots of upstream differences of deletion calls obtained by MoDIL and those improved by ChopSticks. The format of this plot is exactly the same as that in Figure 3.10, except that results for coverage=15 were shown instead of those for coverage=20. The results in this figure indicate that ChopSticks can also improve the resolution of MoDIL results.

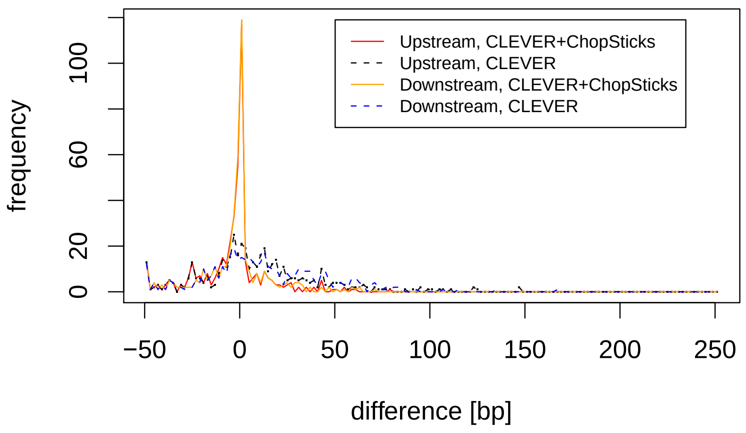


Figure 3.14: Distribution of differences of CLEVER results and those improved by ChopSticks. The distribution of differences of BreakDancer results had long tail in 0–50 bp, whereas that improved by ChopSticks concentrates around zero. Here, $k = 2$, $f = 0.5$, and coverage=5. Each frequency corresponds to the number of displacements in bins of 2 bp.

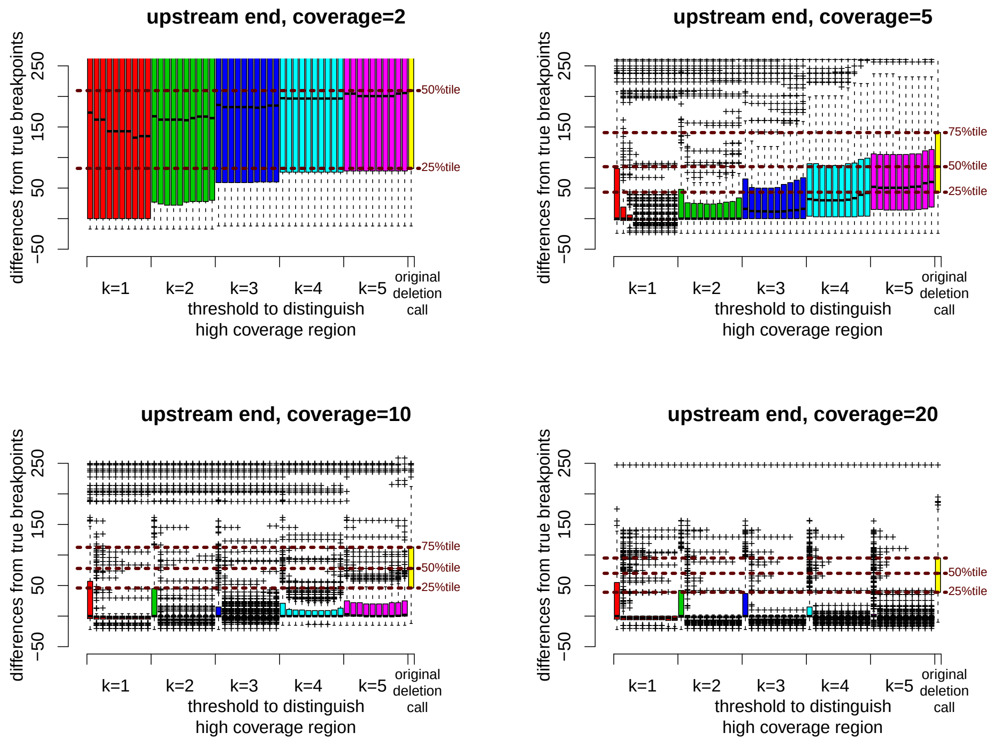


Figure 3.15: CNVnator results improved by ChopSticks. Box-and-whisker plots of upstream differences of deletion calls obtained by CNVnator and those improved by ChopSticks. The format of this plot is exactly the same as that in Figure 3.10. We expanded the original deletion calls of CNVnator outward by the window size (50 bp) because ChopSticks assumes that predicted breakpoints are outside true deletions. The results in this figure indicate that ChopSticks can improve the resolution of CNVnator results if predicted positions of breakpoints are within a few hundreds of bases from true breakpoints.

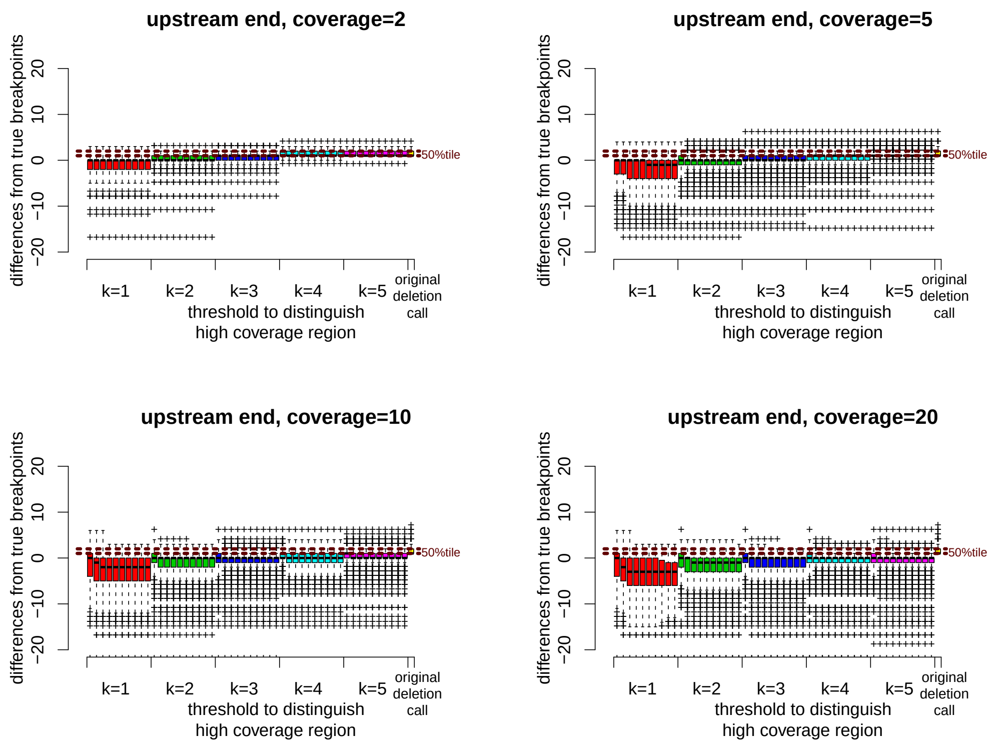
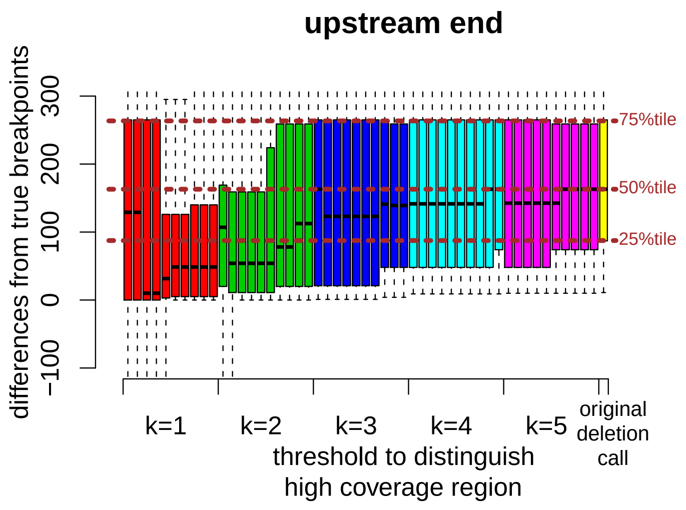


Figure 3.16: Pindel results improved by ChopSticks. Box-and-whisker plots of upstream differences of deletion calls obtained by Pindel and those modified by ChopSticks. The format of this plot is exactly the same as in Figure 3.10. The results in this figure indicate that ChopSticks should not be applied to the Pindel results because the resolution of the Pindel results is already quite high.



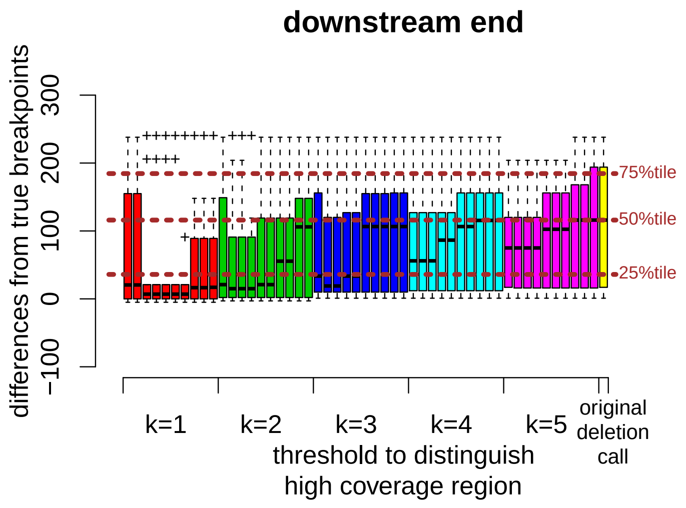


Figure 3.17: BreakDancer results for DBA/2J reads improved by ChopSticks. Box-and-whisker plots of upstream and downstream differences of deletion calls obtained by BreakDancer and those improved by ChopSticks. The results in this figure indicate that ChopSticks can improve the resolution of deletion calls for real sequences. Although ChopSticks trimmed upstream ends of deletion calls too much when $k = 1$ or $k = 2$ and f was small, such problems quickly disappeared for greater k and f values.

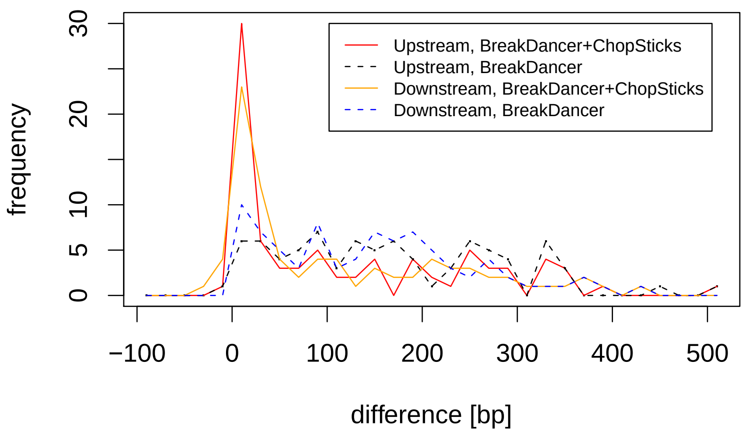


Figure 3.18: Distribution of differences of BreakDancer results and those improved by ChopSticks. The distribution of differences of BreakDancer results had long tail in 0–400 bp, whereas that improved by ChopSticks concentrates around zero and frequencies in the long tail were reduced. Here, $k = 2$, $f = 0.5$. Each frequency corresponds to the number of differences in bins of 20 bp.

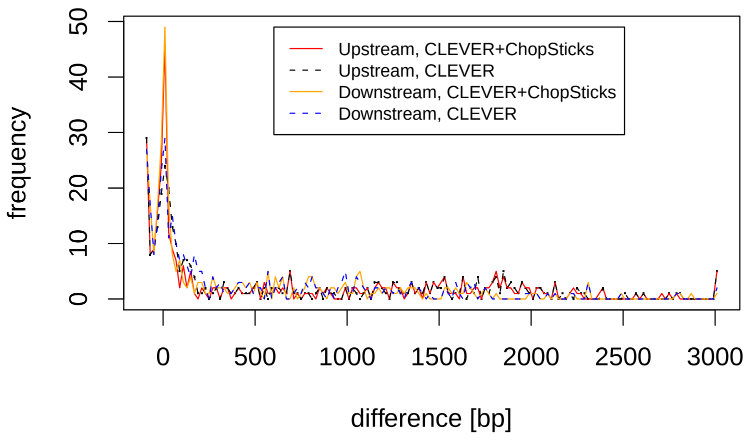


Figure 3.19: Distribution of differences of CLEVER results and those improved by ChopSticks. ChopSticks corrected some of breakpoints predicted by CLEVER so that the peak at zero became stronger. However, the distribution of differences of CLEVER results had long tail in 0–3000 bp and it was difficult for ChopSticks to correct such large differences. Here, $k = 2$, $f = 0.5$. Each frequency corresponds to the number of differences in bins of 20 bp.

Chapter 4

Genome Mapping by a Many-core Processor

For sequence analysis based on NGS, it is necessary to compare NGS sequences and the reference genome, detect positions on the genome where each NGS sequence is derived from, and find the differences between NGS sequences and the reference genome. This process, called *mapping*, is indispensable for various analysis. For example, detection of single nucleotide polymorphisms (SNPs) or structural variations (SVs) needs a mapping step [79]. Because an enormous amount of NGS sequences need to be analyzed, fast mapping methods are required.

In addition to acceleration of mapping by fast software tools [44, 24], acceleration by hardware has a great impact. As predicted by Moore's Law, the performance of computers has been steadily increasing. However, improving the performance of a single processing core has become quite difficult these days. Computing performance has therefore recently been improved mainly by increasing the number of processing cores. For high-performance computing (HPC), using GPUs (graphics processing units) is attracting attention and has achieved great success. Such an approach, called *general-purpose computing on GPUs* (or *GPGPU* for short), has also been applied to sequence alignment [31, 47, 52]. However, a lot of complex optimization techniques are required to maximize the performance of GPGPU. Rewriting software programs for GPGPU is therefore a hard task.

In 2012, Intel Corp. released a coprocessor called Xeon Phi, which contains 60 processing cores and can execute 240 threads simultaneously. Each core has an x86-architecture-based design, which is widely used in PCs and servers. This design is a unique advantage of Xeon Phi, because the same programming model for widely used x86 processors can be applied for Xeon Phi. In addition, Xeon Phi has high peak performance. It has computing performance of 1 TFLOPS with a single board. The fastest supercomputer at Top500 in June 2013[1], *Tianhe-2* of China, contains 48,000 Xeon Phi's and offered 33.86 PFLOPS. Moreover, a research that aims to apply Xeon Phi to sequence alignment has recently been reported [34].

To accelerate mapping of NGS sequences, we ported two famous mapping tools, Burrows-Wheeler Aligner (BWA) [43] and Bowtie2 [38], to Xeon Phi. The aim of porting is to obtain exactly the same mapping results of BWA and Bowtie2 on Xeon Phi as on x86 processors within a much shorter time. As shown below, it was experimentally confirmed that the performances of the ported BWA and Bowtie2 went up drastically when the number of threads increased. Although their peak performances had to be improved, this study is the first step towards the acceleration of mapping by using Xeon Phi.

Contents of this chapter are mainly from published work in [84].

[1] `http://www.top500.org/lists/2013/06/`

Table 4.1: Major incompatibilities of vector operations supported by Xeon Phi and x86

difference		Xeon Phi	x86(SSE2)
vector register	number of registers	32	8
	bit width of elements	32 bits	16 or 8 bits
	number of elements	16	8 or 16
	alignment of memory address	32-byte aligned (vectors containing 16-bit integers), 16-byte aligned (vectors containing 8-bit integers)	16-byte aligned
saturation operations		No	Yes
result of comparison		mask register	vector register

4.1 Incompatibilities between Xeon Phi and x86

BWA and Bowtie2 both use Farrar's algorithm [15] to reduce processing times. This algorithm accelerates a well-known dynamic programming (DP) algorithm for sequence alignment [75, 21], which calculates the optimal alignment of two sequences. Farrar's algorithm exploits vector operations of x86 processors, called *Streaming SIMD Extension 2 (SSE2)*. A single instruction of SSE2 can conduct one of arithmetic operations, comparison operations, logical operations, etc. for vectors containing multiple integers. Because Farrar's algorithm is used, the source codes of BWA and Bowtie2 include tens or hundreds of SSE2 operations. Although Xeon Phi also supports vector operations, they are not compatible with those of x86. The differences between vector operations of Xeon Phi and x86 are summarized in Table 4.1. To port BWA and Bowtie2 to Xeon Phi, all vector operations of x86 must be converted to those of Xeon Phi.

Moreover, the `sort` function of the C++ standard template library (STL), used by Bowtie2, has an incompatibility between Xeon Phi and x86 (Figure 4.1). This incompatibility must also be eliminated to obtain exactly the same mapping results.

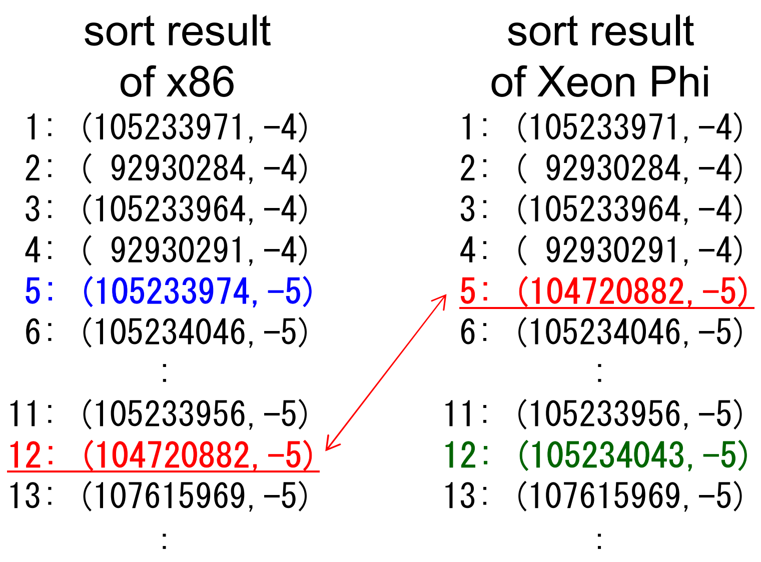
4.2 Resolving Incompatibilities

The incompatibilities between x86 and Xeon Phi were overcome as explained in the following. To implement vector operations, compiler intrinsics were used in the ported programs as in the original BWA and Bowtie2.

4.2.1 Incompatibilities of Vector Registers

Bit Width of Vector Elements The bit width of vector elements is 32 bits on Xeon Phi, which is wider than the bit width on x86 (8 or 16 bits). Because any 8-bit or 16-bit integer can be represented by a 32-bit integer, 32-bit operations were used instead of 8-bit or 16-bit operations.

However, a result of calculation by a 32-bit operation differs from that by an 8-bit or 16-bit operation when overflow occurs. This difference was resolved by



```
  sort result              sort result
    of x86                 of Xeon Phi
 1: (105233971, -4)     1: (105233971, -4)
 2: ( 92930284, -4)     2: ( 92930284, -4)
 3: (105233964, -4)     3: (105233964, -4)
 4: ( 92930291, -4)     4: ( 92930291, -4)
 5: (105233974, -5)     5: (104720882, -5)
 6: (105234046, -5)     6: (105234046, -5)
        :                      :
11: (105233956, -5)    11: (105233956, -5)
12: (104720882, -5)    12: (105234043, -5)
13: (107615969, -5)    13: (107615969, -5)
        :                      :
```

Figure 4.1: An example of inconsistent results obtained by the `sort` function in STL. In this example, a set of integer pairs were sorted. Each pair consisted of a coordinate on the genome and a score. Many pairs had the same scores (-4 or -5). Because only scores were considered during the sort, the orders of the pairs with the same score were different. For example, the 12-th pair in the x86 result appeared as the 5-th result in the Xeon Phi result (red pairs). The 5-th pair in the x86 result and the 12-th pair in the Xeon Phi result do not appear in another (blue and green pairs). This difference caused different outputs of Bowtie2 on Xeon Phi and on x86.

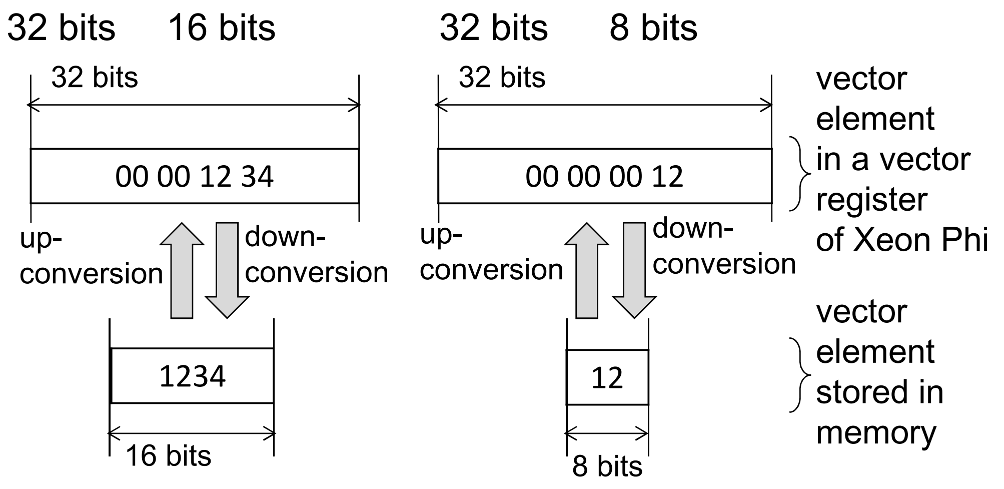


Figure 4.2: Up-conversion and down-conversion supported by Xeon Phi. Both types of conversion change the bit width of each element to fit it to the new vector. If the new bit width is too small to store the element, the element is replaced by the maximum or the minimum value that can be represented by the new bit width.

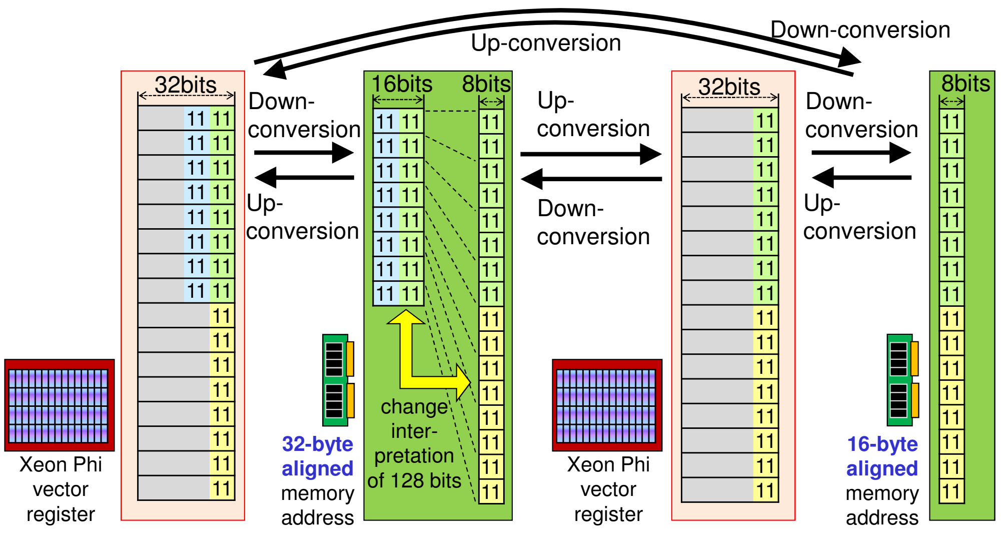


Figure 4.3: Memory access of a vector containing eight 16-bit integers or sixteen 8-bit integers. Because the memory address of such a vector has to be 32-byte aligned on Xeon Phi, a vector containing eight 16-bit integers is copied from or to a 16-byte aligned address by using a buffer that is 32-byte aligned.

emulating saturation operations as described later.

Alignment of Memory Addresses Xeon Phi can store a vector containing 32-bit integers in a vector on memory containing 8-bit integers or 16-bit integers by reducing the bit width of each element. This mechanism is called *down-conversion* (Figure 4.2). The memory address for a vector containing 8-bit integers must be 16-byte aligned, while the address for vectors containing 16-bit integers must be 32-byte aligned. Similarly, *up-conversion* converts a vector containing 8-bit or 16-bit integers on memory into a vector containing 32-bit integers in a vector register. The address of an up-converted vector on memory has to be aligned in the same manner as that of a down-converted one.

In the original source codes of BWA or Bowtie2, a memory address for a vector containing 16-bit integers may not be 32-byte aligned. This is not allowed in Xeon Phi codes. To resolve this problem, the following method (Figure 4.3) was adopted for minimizing the codes to be modified, because memory addresses of vectors are 16-byte aligned in the original source codes. Suppose that a vector containing eight 32-bit integers that emulates a vector containing eight 16-bit integers is to be stored. First, a buffer with enough size is prepared. Second, the vector containing 32-bit integers is stored in this buffer as a vector containing 16-bit integers by down-conversion. Third, the stored vector is loaded as a vector containing sixteen 8-bit integers. Finally, the loaded vector is stored in the final destination as a vector containing sixteen 8-bit integers that is an exact copy of the vector stored in the buffer. Loading a vector was similarly implemented.

Bits Shared by 16-bit and 8-bit Elements A vector register of x86 is 128-bit wide and can be used as either a vector containing eight 16-bit integers or a vector containing sixteen 8-bit integers (Figure 4.4A). Both vectors share all bits in a 128-bit vector register. Therefore, two 8-bit elements c_1 and c_2 in Figure 4.4A, for example, are stored in the same bits as a 16-bit element b_1. To obtain higher performance by exploiting this fact, Bowtie2 uses a programming practice

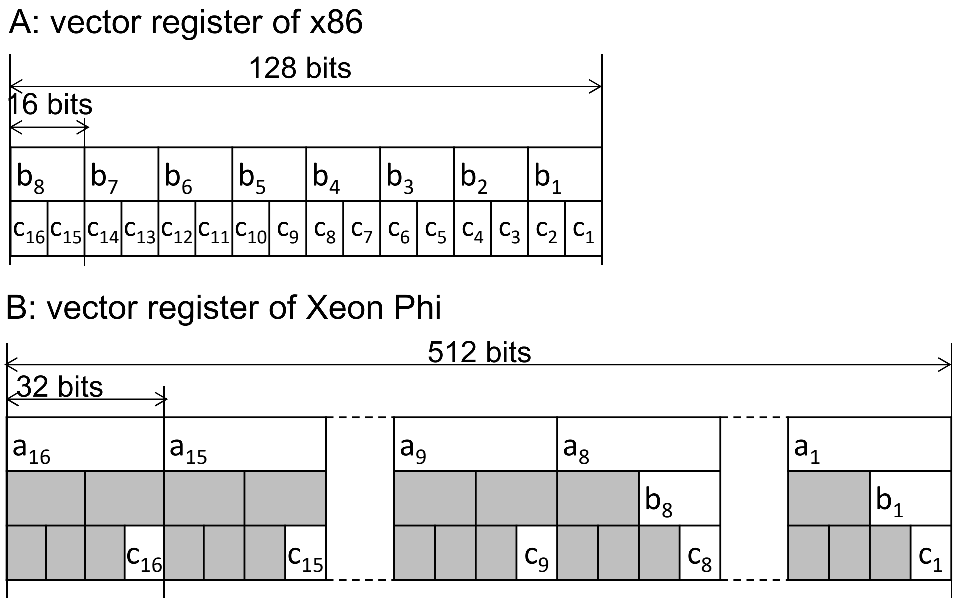

A: vector register of x86
128 bits
16 bits
b8 b7 b6 b5 b4 b3 b2 b1
c16 c15 c14 c13 c12 c11 c10 c9 c8 c7 c6 c5 c4 c3 c2 c1
B: vector register of Xeon Phi
512 bits
32 bits
a16 a15 a9 a8 a1
b8 b1
c16 c15 c9 c8 c1

Figure 4.4: Structures of vector registers of Xeon Phi and x86. A: In the vector register of x86, a 16-bit element b_i $(1 \leq i \leq 8)$ shares the same bits with 8-bit elements c_{2i-1} and c_{2i}. B: In the vector register of Xeon Phi, a 32-bit element in a vector, a_i $(1 \leq i \leq 16)$, is used to emulate both of 16-bit elements b_i and 8-bit elements c_i in this study.

that sets two flanking 8-bit elements to the same integer k at once by setting one 16-bit element to an integer $256k + k$. In our modified program ported to Xeon Phi, however, vectors containing 16-bit integers and those containing 8-bit integers share bits differently (Figure 4.4B). Accordingly, such a programming practice of Bowtie2 was removed.

4.2.2 Emulation of Saturation Operations

Saturation operations are variations of arithmetic operations. When positive overflow occurs, the result of a saturation operation is replaced with the maximum integer represented by the same bit width. Similarly, when negative overflow occurs, the result of a saturation operation is replaced with the minimum integer. Because Xeon Phi does not support saturation operations, they were emulated in the following ways.

Max Operations and Min Operations A *max operation* generates a new vector whose elements are larger elements of those at the same positions in two input vectors. A *min operation* similarly generates a new vector containing smaller elements. Because saturation operations forcedly replace results of arithmetic operations with the maximum or the minimum integer represented by the same bit width when overflow occurs, a max operation or a min operation was inserted just after arithmetic operations to emulate this replacement. In the case of vectors containing signed 16-bit integers, for example, the modified operation is explained with the following mathematical expressions. Let a and b be input vectors, c be an output vector, and x_i be the i-th element in a vector x, where x is one of a, b, and c. By using these notations, the modified operation for addition

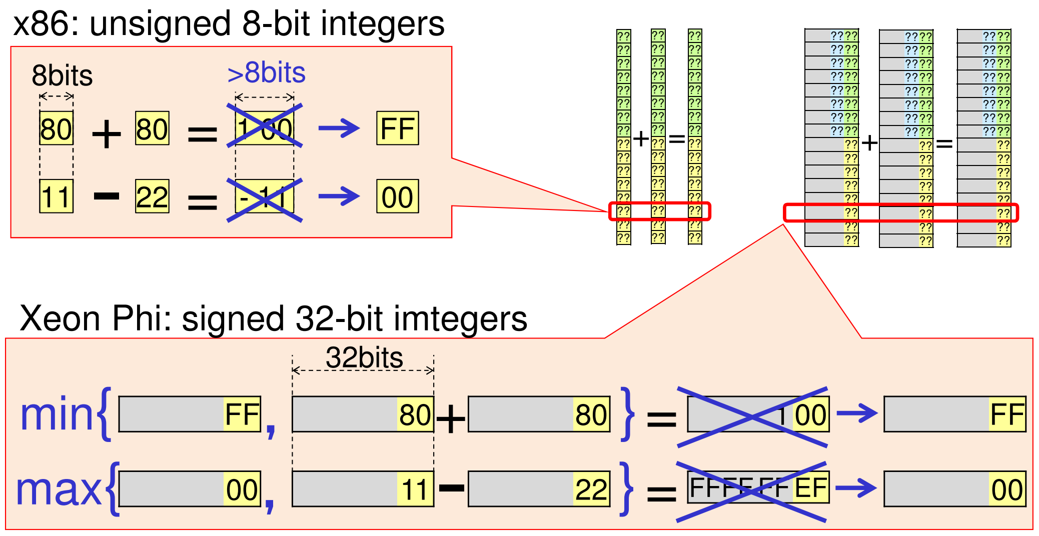

x86: unsigned 8-bit integers
8bits
>8bits
80 + 80 = 100 → FF
11 − 22 = -11 → 00
Xeon Phi: signed 32-bit imtegers
32bits
min{ FF, 80 + 80 } = 100 → FF
max{ 00, 11 − 22 } = FFFFFFEF → 00

Figure 4.5: Schematic illustration of saturation operations and emulation using max and min operations. The result of a saturation operation of n bits is replaced with the largest integer that can be represented by n bits if the result of the operation is larger than the largest integer. Similarly, the result of a saturation operation of n bits is replaced with the smallest integer that can be represented by n bits if the result of the operation is smaller than the smallest integer. We emulated this operation by using max and min operations of Xeon Phi.

is represented by the following expression:

$$c_i := \min\{2^{15} - 1, a_i + b_i\}.$$

Here, it is assumed that $b_i \geq 0$ for all i. Similarly, for subtraction,

$$c_i := \max\{-2^{15}, a_i - b_i\}.$$

See Figure 4.5 for a schematic illustration.

Down-conversion Down-conversion replaces any 32-bit element in a vector register with the maximum or the minimum integer represented by the bit width of the vector stored, if the element cannot be represented by the bit width. The rule of replacement is the same as that of saturation operations. Therefore, down-conversion was used to emulate a saturation operation if the result of a saturation operation was immediately stored in the memory.

4.2.3 Comparison Operations

Both Xeon Phi and x86 support comparison operations, which compare each element of two vectors one by one. In x86, the results of a comparison operation is stored in a vector, whose element is set to -1 if a specified condition is satisfied and to 0 otherwise. For example, the equality of elements in two vectors $(1, 2, 3, 4, 5, 6, 7, 8)$ and $(1, 2, 3, 4, 4, 3, 2, 1)$ is represented by a new vector $(-1, -1, -1, -1, 0, 0, 0, 0)$. Xeon Phi, on the contrary, stores the results of a comparison operation in a special register, called a *mask register* (Figure 4.6). A mask register is 16-bit wide. Each bit in the register corresponds to a comparison result of each pair of corresponding elements in two compared vectors. Because

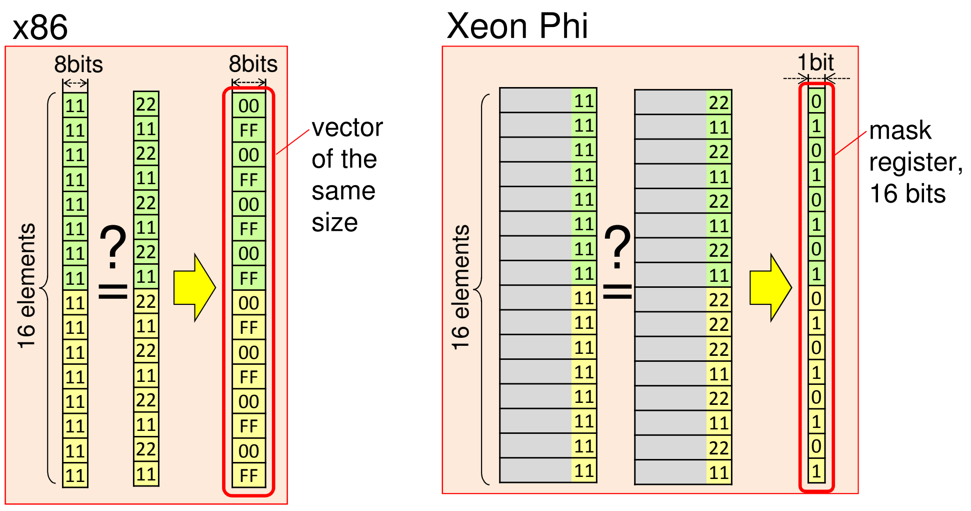


Figure 4.6: Schematic illustration of comparison operations. The results of comparison is stored in a vector register by x86, while they are stored in a mask register in Xeon Phi.

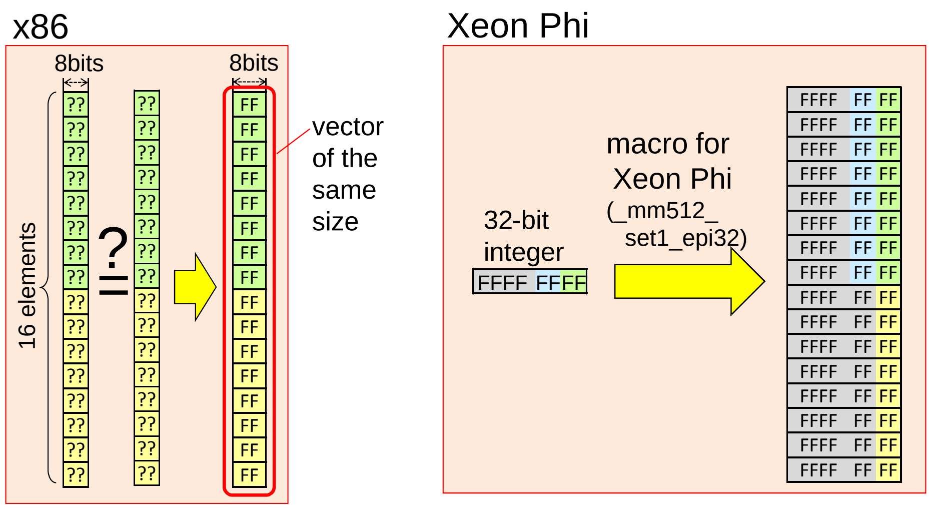


Figure 4.7: Schematic illustration of using comparison operations to set all bits to 1. When a result of comparison is yes, x86 sets 1 to all bits in the corresponding element of a vector register. This behavior can be exploited to set all bits of a vector register to 1. because the result of comparison is always yes for all elements of vectors if equality of two vectors is tested.

of this difference, all operations that depend on the comparison results had to be rewritten.

Interestingly, the most of comparison operations in the original Bowtie2 codes compare identical vectors and determine their equality, resulting in a vector whose elements are all -1. In other words, comparison operations are used for the purpose of filling vectors with -1. Such comparison operations were replaced with a macro `_mm512_set1_epi32`, which sets all elements in a vector to a given integer (Figure 4.7).

4.2.4 Other Incompatible Vector Operations

In spite of the above-described rewriting, there were extra x86 instructions not implemented in Xeon Phi. These instructions were rewritten as follows.

Filling a Vector with a Given Integer Instruction `_mm_insert_epi16` sets a single specified element in a vector to a given integer. On the other hand, instruction `_mm_shufflelo_epi16` swaps four rightmost elements in a vector containing eight 16-bit integers. They are not implemented in Xeon Phi and cannot be easily replaced even with multiple instructions. However, these two instructions were always used together for the purpose of setting all elements in a vector to a given single integer. Therefore, they were replaced with a single macro `_mm512_set1_epi32` that conducts the desired operation.

Shifting Vector Elements to the Left When applied to a vector a, instruction `_mm_slli_si128` modifies each element in a as follows:

$$a_k := \begin{cases} a_{k-i} & \text{if } k - i \geq 1, \\ 0 & \text{otherwise,} \end{cases} \tag{4.1}$$

where i and k are non-negative integers such that $1 \leq i, k \leq 16$. When $i = 1$, for example, a vector $(a_{16}, a_{15}, a_{14}, \ldots, a_2, a_1)$ becomes $(a_{15}, a_{14}, a_{13}, \ldots, a_1, 0)$.

Because i is always one in the source code of BWA and Bowtie2, the instruction was emulated with the following two steps. First, all elements except the rightmost one are appropriately set by instruction `_mm512_permutevar_epi32` , which sets each element in a vector to any element in the same vector. In this step, a vector $(a_{16}, a_{15}, a_{14}, \ldots, a_2, a_1)$ is modified to be $(a_{15}, a_{14}, a_{13}, \ldots, a_1, a_1)$. Second, the rightmost element is set to zero by copying the rightmost element in a vector whose elements are all zeros. A mask register was used to implement this copy operation.

Shifting Vector Elements to the Right Similarly, `_mm_srli_si128` modifies each element in a as follows:

$$a_k := \begin{cases} a_{k+i} & \text{if } k + i \leq n, \\ 0 & \text{otherwise,} \end{cases} \tag{4.2}$$

where n is the number of elements in a. Unlike `_mm_slli_si128`, `_mm_srli_si128` is also used when $i \neq 1$ in the source codes of BWA and Bowtie2. However, `_mm_srli_si128` is used only in the following two cases.

1. Setting all elements except the rightmost one to zero:

   To emulate the operation in this case, the rightmost element was copied to a vector whose elements are all zeros by using a mask register.

2. Choosing the largest element in a vector:

   A single macro `_mm512_reduce_max_epi32` of Xeon Phi, which conducts the desired operation by itself, was used.

4.2.5 Sort Function of STL Library

The `sort` function and other functions called by the `sort` function were extracted from the source code of STL. The extracted functions were then integrated into the source code of Bowtie2. This integration produced exactly the same mapping results on Xeon Phi as on x86.

Table 4.2: Processing times for ERR246054. Bowtie2 did not work with 480 threads. All times are in seconds.

no. of	processing time		ratio	
threads	Bowtie2	BWA	Bowtie2	BWA
1	8261	9723	1.0	1.0
4	2060	2516	4.0	3.9
8	1038	1295	8.0	7.5
16	534	688	15.5	14.1
30	312	395	26.5	24.6
60	220	231	37.6	42.1
120	860	207	9.6	47.0
240	2401	268	3.4	36.3
480	N/A	280	N/A	34.7

4.3 Results of Evaluation

The ported BWA and Bowtie2 programs were evaluated on a Linux server with Xeon Phi 5110P (60 cores, 1.053 GHz, 8 GB RAM). NGS sequence data used for evaluation were those of a Japanese person (ERR246054) sequenced in the 1000 Genomes Project [79]. They consisted of 1,809,507 pairs of NGS sequences whose length was 100 bases each. Processing times of BWA and Bowtie2 for 1, 4, 8, 16, 30, 60, 120, 240, and 480 threads were measured. Whenever the executed process finished normally, the mapping results were exactly the same as the results obtained by the original BWA and Bowtie2 on x86 processors.

To obtain mapping results, BWA must be invoked two times with `aln` subcommand, and once with `sampe` subcommand. Because `aln` subcommand is much more time consuming than `sampe` subcommand, we focused on the processing time of `aln` subcommand and evaluated its processing time as that of BWA.

The processing times of mapping ERR246054 sequences onto chromosome 1 by the ported BWA and Bowtie2 are shown in Figures 4.8 and 4.9 and listed in Table 4.2. The performances went up almost proportionally to the number of threads (up to 30 threads), and peaked when 60 or 120 threads were used. However, using more than 120 threads deteriorated the performance. One possible reason is that the amount of communications between cores and the memory exceeded the capacity of the internal ring bus of Xeon Phi when 120 or more threads were used. It can thus be concluded that while Xeon Phi has 60 cores that can execute four threads each, the number of threads executed by each core should be one or two to get the best performance on Xeon Phi for BWA and Bowtie2.

Meanwhile, a quad-core x86 CPU, Core i7 920 2.67GHz, did the same task by using eight threads in 145 and 190 seconds for Bowtie2 and BWA, respectively. Accordingly, this study is only the first step towards acceleration of genome mapping by using Xeon Phi.

4.4 Discussion and Summary

Two well-known mapping tools, BWA and Bowtie2, were ported to a many-core processor Xeon Phi. Primary obstacles in porting BWA and Bowtie2 were incompatibilities of vector operations used in these programs. These incompatibilities were circumvented by emulating vector operations of x86 processors with those

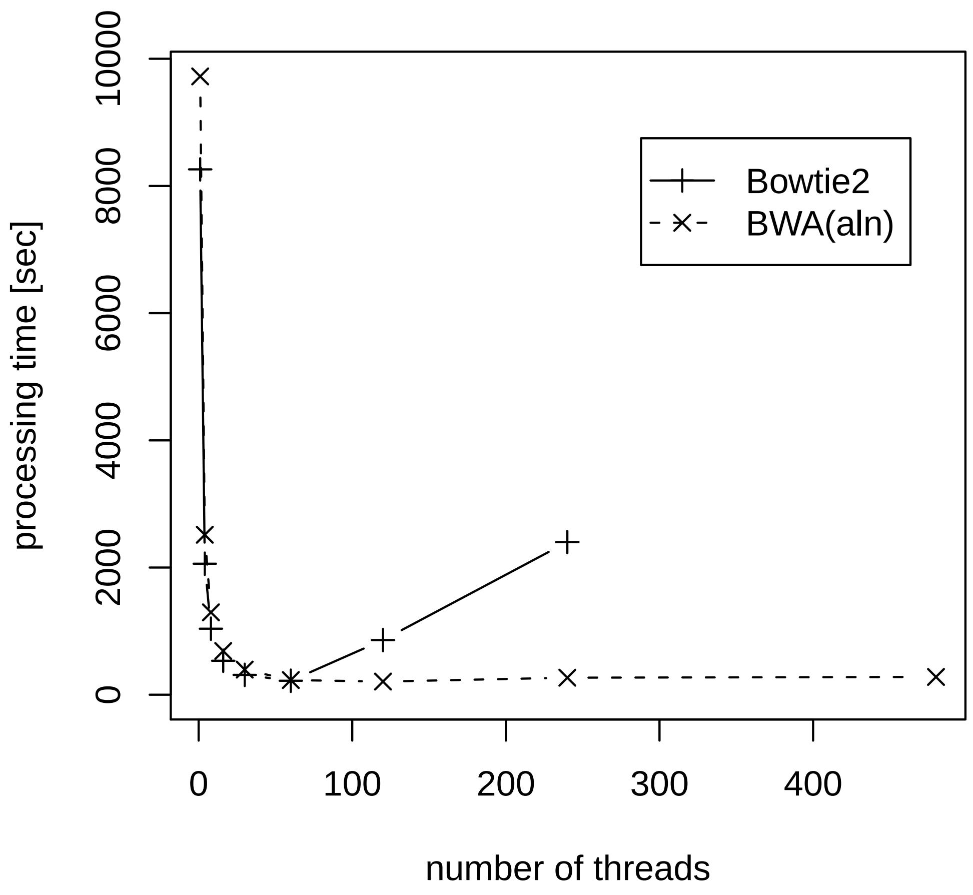


Figure 4.8: Processing times for ERR246054.

of Xeon Phi. In a computational experiment, it was confirmed that the more threads were used up to 60 threads, the higher the performances of ported programs were. The peak performances for BWA and Bowtie2 were observed when 120 and 60 threads are used, respectively. These results imply that using tens of threads on the many-core processor Xeon Phi is very much promising for accelerating mapping. In addition, the ported programs successfully generated exactly the same mapping results as the original BWA and Bowtie2.

In the future, the performances of BWA and Bowtie2 on Xeon Phi are expected to be further improved by three ways. First, fully exploiting computation power of Xeon Phi; for example, using all 32 vector registers at once (Figure 4.10). In this study, only vector operations of x86 that has eight 128-bit vector registers were emulated. Second, using Xeon Phi with x86 processors in a coordinated manner. This enables x86 processors and Xeon Phi to execute steps that fit their respective architectures. Because the latest x86 processors are faster than Xeon Phi for single-threaded processes, steps that cannot be concurrently executed should be done on x86 processors. Third, improving the rewritten code; for example, removing max operations and min operations when results of mapping are not affected by removal.

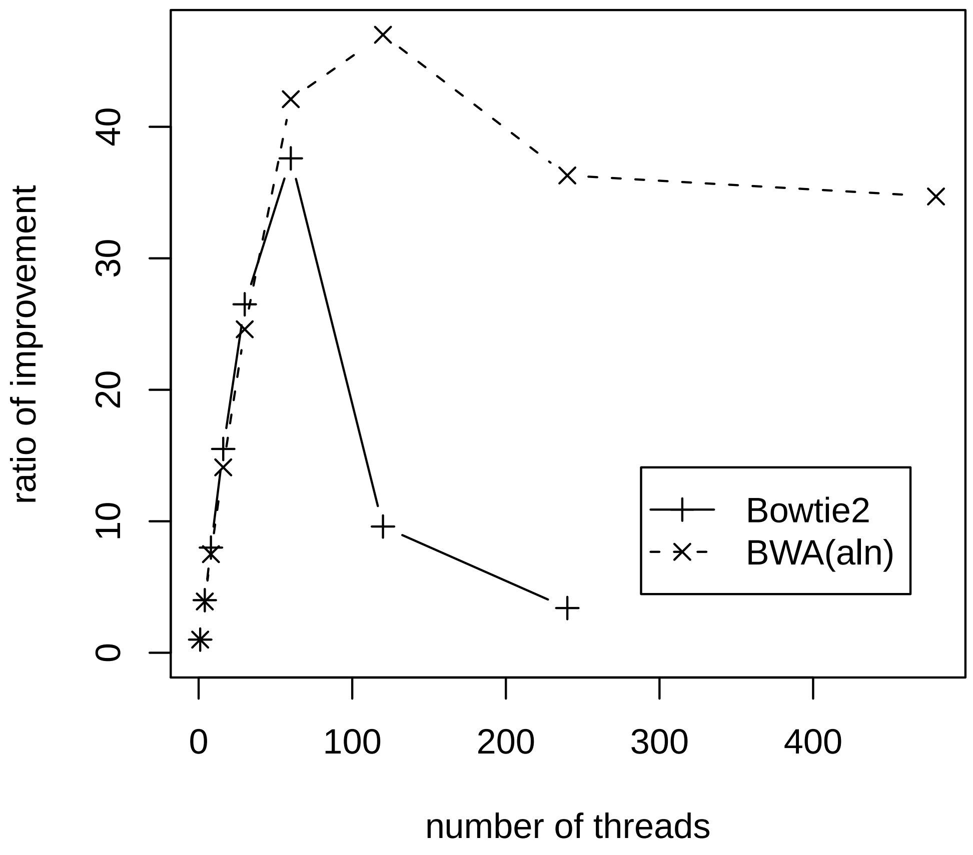


Figure 4.9: Performance improvement for ERR246054.

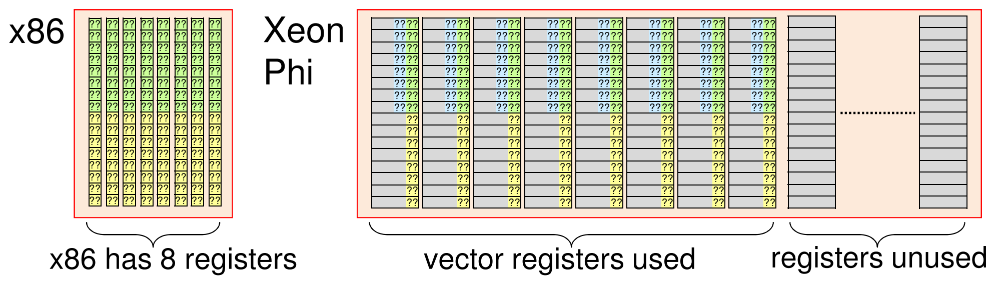


Figure 4.10: Vector registers not used in this study. Because we emulated vector operations of x86, there remain vector registers of Xeon Phi not used in this study.

The hardware of Xeon Phi will also be updated. The current release of Xeon Phi, codenamed *Knights Corner*, is only the first product of a lineup of many-core processors. It adopts a ring bus that becomes a bottleneck when a large amount of data is moved between cores and the memory. As new designs come out, the architecture of Xeon Phi will evolve to provide low-latency and high-bandwidth communications between cores and the memory.

Chapter 5

Mutual Comparison of Strings

Let $\mathcal{T}$ be a set of hidden strings and $\mathcal{S}$ be a set of their concatenations. We consider the problem of inferring $\mathcal{T}$ from $\mathcal{S}$ when only $\mathcal{S}$ is given. Solving this problem is useful for the analysis of rearrangements of assembled genomes. Because new sequencing technologies[3, 14] will enable us to determine long sequences consisting of thousands of bases at once, *de novo* assembly would be much easier in the future. Our problem can be applied to compare assembled genomes and to elucidate their common and different regions. Another motivation is the analysis of cDNA sequences. A gene might have more than one cDNA sequence by inserting or deleting alternative segments. This phenomenon is prevalent in many organisms [53]. Usually, alternatively spliced sequences are detected by aligning them with the reference genome [37]. Nonetheless, methods that require as input only cDNA sequences would be useful because not all organisms have had their genomic sequences determined.

To clarify the problem, we show a small example.

Example 5.1. *Suppose that we are given $\mathcal{S} = \{S_0, S_1, S_2\}$, where*

$$\begin{aligned} S_0 &= \texttt{ACGGTCTAGAATAGCAGGCTCGTCCTATGGCATTTT}, \\ S_1 &= \texttt{CATCTGGTAGCAGGCTCGTCCTATCCAAGTAAAGGAC}, \\ S_2 &= \texttt{CATCTGGTAAGTGGGCCGTCCTAT}. \end{aligned}$$

These are concatenations of strings in a set $\mathcal{T} = \{T_i | 0 \leq i < 8\}$, where

$$\begin{aligned} T_0 &= \texttt{ACGGTCTAGAAT}, & T_1 &= \texttt{AGCAGGCTC}, \\ T_2 &= \texttt{GTCCTAT}, & T_3 &= \texttt{GGCATTTT}, \\ T_4 &= \texttt{CATCTGGT}, & T_5 &= \texttt{CCAAGT}, \\ T_6 &= \texttt{AAAGGAC}, & T_7 &= \texttt{AAGTGGGCC}. \end{aligned}$$

In fact, S_0, S_1 and S_2 can be rewritten as:

$$S_0 = T_0T_1T_2T_3,\ S_1 = T_4T_1T_2T_5T_6,\ S_2 = T_4T_7T_2.$$

We aim to infer $\mathcal{T}$ from $\mathcal{S}$.

We propose a fast and scalable algorithm for inferring $\mathcal{T}$ from $\mathcal{S}$. Our approach is based not on optimization but on finding common substrings and splitting them. Our contributions are summarized as follows. First, we formally define a class of strings called disjoint common substrings (DCS's) that can be determined only from $\mathcal{S}$ and a positive integer parameter. Each of DCS's corresponds to a string in $\mathcal{T}$ or a concatenation of strings in $\mathcal{T}$ that always occur adjacent in the same order in $\mathcal{S}$. Second, we propose an algorithm that identifies all DCS's by all-to-all comparison of strings in $\mathcal{S}$. This algorithm can be completed in $O(L)$ time regardless of $|\mathcal{S}|$, where L is the sum of the lengths of all strings in $\mathcal{S}$. These contributions enable us to efficiently decompose multiple strings into non-trivial, non-overlapping substrings.

Contents of this chapter are mainly from published work in [83][1].

5.1 Related Works

5.1.1 Elementariness

If we formulate the problem as an optimization problem, it is difficult to give an efficient algorithm to obtain a solution. Let $|\mathcal{S}|$ and $|\mathcal{T}|$ respectively be the cardinalities of $\mathcal{S}$ and $\mathcal{T}$. Néraud [65] considered the problem of determining, for a given set $\mathcal{S}$ of strings and an integer k, whether there exists a set $\mathcal{T}$ of strings such that $\mathcal{S} \subseteq \mathcal{T}^*$ and $|\mathcal{T}| \leq k$. Néraud proved that this problem is NP-complete even when $k = |\mathcal{S}| - 1$; that is, it is NP-complete to determine the existence of $\mathcal{T}$ that is smaller than $\mathcal{S}$.

5.1.2 Maximum Common Substring Problem

Lopresti and Tomkins [48] studied the problem of comparing two strings by extracting a multi-set of substrings and placing them into correspondence. They proved that, when substrings in the multi-set do not overlap each other in given strings whereas they cover the whole of the given strings, finding the smallest such multi-set is NP-complete. This problem is called the minimum common substring problem (MCSP) [20]. It was also proved that a restricted variant of MCSP called k-MCSP, where each symbol in the given strings occurs at most k times in each given string, is NP-hard for $k \geq 2$ [20]. Thus, several approximation algorithms have been proposed for MCSP [11, 12, 20]. In addition, Lopresti and Tomkins [48] proposed polynomial time algorithms for cases where the constraints on multi-sets of substrings were relaxed. However, these algorithms compare only two strings. Any extension for more than two strings would be computationally much harder. taking into account that obtaining the optimal multiple sequence alignment (MSA) is NP-hard [4, 81] whereas the optimal alignment of two strings can be obtained in polynomial time [64, 76]. However, a method scalable to $|\mathcal{S}|$ is preferable for reducing computation time and and even for accuracy, since the more strings we compare, the more chance we have to find strings in $\mathcal{T}$ by exploiting differences among strings in $\mathcal{S}$.

5.1.3 Multiple Sequence Alignment

For practical uses, a lot of MSA programs have been available [7]. Unfortunately, their purpose is not to decompose given strings into substrings of which given strings are concatenations, but to obtain alignments by finding similar regions.

5.2 Preliminaries

Let N be the cardinality of $\mathcal{S}$, and L be the sum of the lengths of all strings in $\mathcal{S}$. We denote by Σ a finite alphabet of which strings in $\mathcal{S}$ consist, by Σ^*

[1]The copyright of the material in this chapter is retained by the Information Processing Society of Japan (IPSJ). This material is published on this web site with the agreement of the author and the IPSJ. All Rights Reserved, Copyright (C) Information Processing Society of Japan.

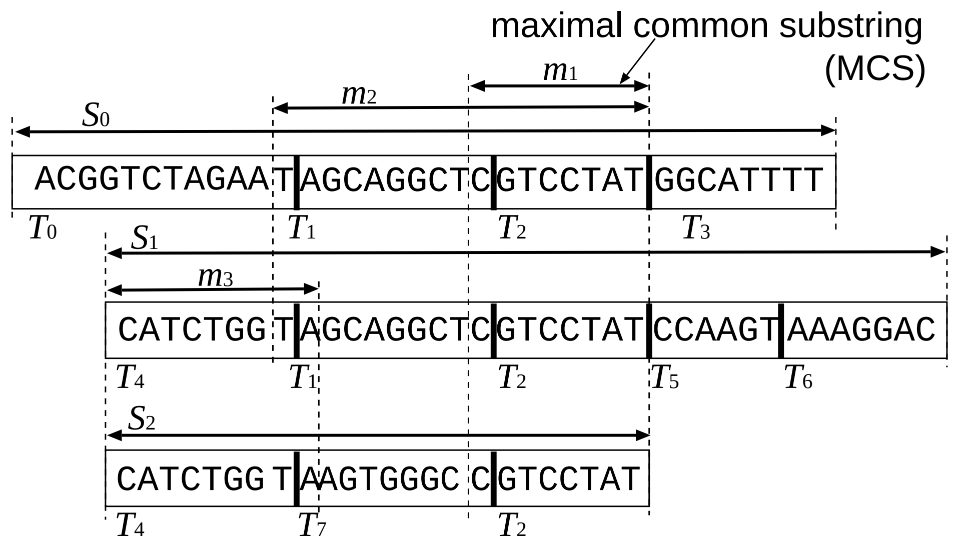

maximal common substring (MCS)
m1
m2
S0
ACGGTCTAGAA T AGCAGGCT C GTCCTAT GGCATTTT
T0
T1
T2
T3
S1
m3
CATCTGG T AGCAGGCT C GTCCTAT CCAAGT AAAGGAC
T4
T1
T2
T5
T6
S2
CATCTGG T AAGTGGGC C GTCCTAT
T4
T7
T2

Figure 5.1: MCS$(\mathcal{S}, l)$ for Example 5.1. Here, $l = 6$. Strings m_1, m_2, m_3, S_0, S_1, and S_2 are MCS's. They are all maximal, that is, they lose some of their occurrences if they are extended to either the left or right.

the set of possibly empty strings, and by Σ^+ the set of non-empty strings. For a string $s \in \Sigma^*$, the length of s is denoted by $|s|$. When $s = s_1 s_2 s_3$ for some $s_1, s_2, s_3 \in \Sigma^*$, they are respectively called a *prefix*, a *substring*, and a *suffix* of s. Each of them is *proper* if it is not identical to s. When s is a substring of $S_i \in \mathcal{S}$ beginning at the j-th position of S_i, we say that s *occurs at* (i, j) or that (i, j) is an *occurrence* of s. Let $Occ(s)$ be the set of all occurrences of $s \in \Sigma^+$. For an integer k, we define $Occ(s) + k = \{(i, j + k) | (i, j) \in Occ(s)\}$. An empty set is denoted by $\emptyset$.

Let STree$(\mathcal{S})$ be a generalized suffix tree [22] of all strings in $\mathcal{S}$. Each string in $\mathcal{S}$ is appended a distinct termination symbol at its right end [22]. A *path-label* of a node v in STree$(\mathcal{S})$ is the concatenation of edge labels from the root to v. We denote by $p(v)$ the string obtained by removing any termination symbol from the path-label of v. Let $\mathcal{L}(i, j)$ be the leaf of STree$(\mathcal{S})$ that represents the j-th suffix of $S_i \in \mathcal{S}$.

We capture common substrings in $\mathcal{S}$ as maximal common substrings defined below.

Definition 5.1 (MCS)**.** *A string* $m \in \Sigma^+$ *is* a maximal common substring (MCS) for $\mathcal{S}$ and l, *if* m *is a substring of some* $S_i \in \mathcal{S}$ *and has the following properties:*

(M1) $|m| \geq l$.

(M2) $Occ(m) \neq Occ(ms)$ *for any* $s \in \Sigma^+$.

(M3) $Occ(m) \neq Occ(sm) + |s|$ *for any* $s \in \Sigma^+$.

Let MCS$(\mathcal{S}, l)$ *be the set of all MCS's for* $\mathcal{S}$ *and* l.

We show MCS$(\mathcal{S}, l)$ for Example 5.1 in Figure 5.1. MCS's are a natural extension of maximal repeats [22]. MCS's can exist in an arbitrary number of given strings, and can be identified in $O(L)$ time in the same way to identify maximal repeats. MCS's were also known to as *core blocks* [42]. However, to

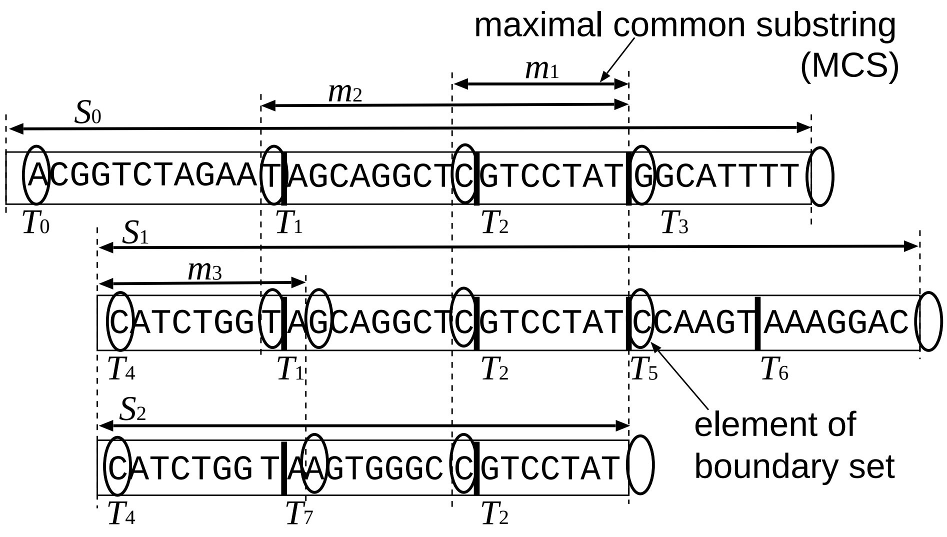

maximal common substring (MCS)
m1
m2
S0
ACGGTCTAGAATAGCAGGCTCGTCCTATGGCATTTT
T0
T1
T2
T3
S1
m3
CATCTGGTAGCAGGCTCGTCCTATCCAAGTAAAGGAC
T4
T1
T2
T5
T6
S2
element of boundary set
CATCTGGTAAGTGGGCCGTCCTAT
T4
T7
T2

Figure 5.2: The boundary set $\mathcal{B}(\mathcal{S}, l)$ for Example 5.1. Each circle at the j-th position of S_i indicates that $(i, j) \in \mathcal{B}(\mathcal{S}, l)$.

emphasize that MCS's are common substrings in $\mathcal{S}$ and to simplify the exposition, we use this name and definition. We also use the following class of substrings.

Definition 5.2. RightMCS$(\mathcal{S}, l)$ *is a set of non-empty strings, each of which is a substring of some* $S_i \in \mathcal{S}$ *and satisfies (M1) and (M2).*

RightMCS$(\mathcal{S}, l)$ is a natural extension of strings considered in the DNA contamination problem[22]. All strings in RightMCS$(\mathcal{S}, l)$ can be found in $O(L)$ time as $p(v)$ of nodes v in STree$(\mathcal{S})$ such that $|p(v)| \geq l$ [22]. Note that any $r \in$ RightMCS$(\mathcal{S}, l)$ is a suffix of some MCS. In fact, there exists some $m \in$ MCS$(\mathcal{S}, l)$ such that $Occ(m) + |m| = Occ(r) + |r|$ for any $r \in$ RightMCS$(\mathcal{S}, l)$.

5.3 Definition of DCS's

In Figure 5.1, m_1(=`CGTCCTAT`) is an MCS shared by all of S_0, S_1, and S_2 of Example 5.1, while m_2(=`TAGCAGGCT CGTCCTAT`) is shared by only S_0 and S_1. This suggests that there is a string in $\mathcal{T}$ shared by all of S_0, S_1, and S_2, and on its left, there is another shared by only S_0 and S_1. To infer both of them, we should split m_2 at a boundary of m_1. We generalize this inference.

Definition 5.3 (Boundary set)**.** *The* boundary set for $\mathcal{S}$ and l*, denoted by* $\mathcal{B}(\mathcal{S}, l)$*, is defined as follows:*

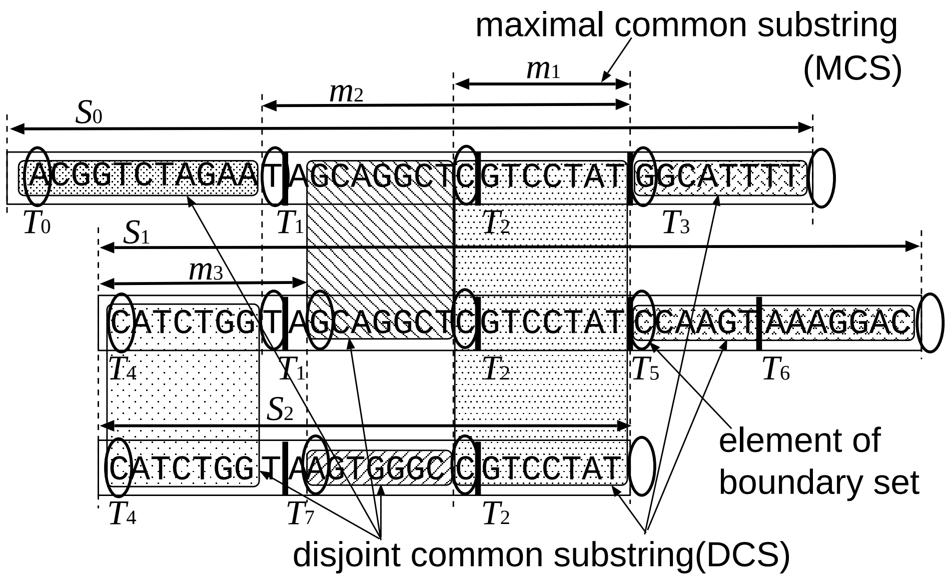
$$\mathcal{B}(\mathcal{S}, l) = \mathcal{B}_L(\mathcal{S}, l) \cup \mathcal{B}_R(\mathcal{S}, l),$$

where

$$\mathcal{B}_L(\mathcal{S}, l) = \bigcup_{m \in \mathrm{MCS}(\mathcal{S}, l)} Occ(m),$$

$$\mathcal{B}_R(\mathcal{S}, l) = \bigcup_{m \in \mathrm{MCS}(\mathcal{S}, l)} (Occ(m) + |m|).$$

In Figure 5.2, we show $\mathcal{B}(\mathcal{S}, l)$ for Example 5.1. By using boundary sets, we infer strings in $\mathcal{T}$ as substrings of given strings that do not cross over any



maximal common substring (MCS)
m_1
m_2
m_3
S_0
S_1
S_2
ACGGTCTAGAATAGCAGGCTCGTCCTATGGCATTTT
CATCTGGTAGCAGGCTCGTCCTATCCAAGTAAAGGAC
CATCTGGTAAGTGGGCCGTCCTAT
T_0
T_1
T_2
T_3
T_4
T_5
T_6
T_7
element of boundary set
disjoint common substring(DCS)

Figure 5.3: DCS($\mathcal{S}, l$) for Example 5.1. DCS's, indicated by hatched areas, do not cross over any boundaries of MCS's. All of $T_0, \ldots, T_7$ except T_5 and T_6 are captured almost as a whole. However, There are unavoidable ambiguities at their boundaries.

boundaries of MCS's. It is these substrings that are identified by our method to infer strings in $\mathcal{T}$.

Definition 5.4 (DCS)**.** *A string* $e \in \Sigma^+$ *is* a disjoint common substring (DCS) for $\mathcal{S}$ and l, *if* e *is a substring of some* $S_i \in \mathcal{S}$ *and has the following properties:*

(D1) $|e| \geq l$.

(D2) For any $(i, j) \in Occ(e)$ *and any integer* k *such that* $1 \leq k < |e|$, $(i, j + k) \notin \mathcal{B}(\mathcal{S}, l)$.

(D3) $\mathcal{B}(\mathcal{S}, l) \cap (Occ(e) + |e|) \neq \emptyset$.

(D4) $\mathcal{B}(\mathcal{S}, l) \cap Occ(e) \neq \emptyset$.

Let DCS($\mathcal{S}, l$) *be the set of all DCS's for* $\mathcal{S}$ *and* l.

Intuitively, (D2) means that e does not contain any boundaries of MCS's in its middle. In addition, (D3) and (D4) mean that e is maximal among those that satisfy (D2). In Figure 5.3, we show DCS($\mathcal{S}, l$) for Example 5.1.

The choice of the parameter l is important. Suppose that a random string of length L is generated by concatenating symbols independently chosen from Σ with the same probability. Then, the expected number of identical pairs of substrings of length l is no more than $E = L^2/(2|\Sigma|^l)$. To avoid matches only by chance, it is recommended to set l so that E is sufficiently small. Since our algorithm that identifies DCS($\mathcal{S}, l$) runs fast as shown later and since l is the only parameter, it is easy to determine the value of l by try and error. The value of l can also be easily determined by try and error, because our algorithm that identifies DCS($\mathcal{S}, l$) runs fast as shown later and because l is the only parameter of our algorithm.

Note that inference of $\mathcal{T}$ cannot be perfect as long as we are given only $\mathcal{S}$. Some ambiguities are unavoidable. In particular, it is impossible to determine boundaries of strings in $\mathcal{T}$ in general. In addition, strings in $\mathcal{T}$ that always occur adjacent in the same order in $\mathcal{S}$ would be fused.

5.4 Algorithm That Identifies DCS's

Since a DCS is not always a path-label or $p(v)$ of a node v in $\mathrm{STree}(\mathcal{S})$, $\mathrm{STree}(\mathcal{S})$ cannot be directly used to identify $\mathrm{DCS}(\mathcal{S}, l)$. We We introduce a class of strings that bridge $\mathrm{DCS}(\mathcal{S}, l)$ and $\mathrm{STree}(\mathcal{S})$.

Definition 5.5. *Let $H(\mathcal{S}, l)$ be a set of strings such that any $h \in H(\mathcal{S}, l)$ has the following properties:*

(H1) $h \in \mathrm{RightMCS}(\mathcal{S}, l)$.

(H2) For any proper prefix s of h, $s \notin \mathrm{RightMCS}(\mathcal{S}, l)$.

(H3) $\mathcal{B}(\mathcal{S}, l) \cap Occ(h) \neq \emptyset$.

Then, the following lemma hold.

Lemma 5.1. *Any $e \in \mathrm{DCS}(\mathcal{S}, l)$ is a prefix of some $h \in H(\mathcal{S}, l)$ such that $Occ(e) = Occ(h)$.*

Proof. Let $s \in \Sigma^*$ be the longest such that $Occ(e) = Occ(es)$. We prove that $es \in H(\mathcal{S}, l)$, claiming that es satisfies (H1)–(H3).

(H1) Since $|es| \geq |e| \geq l$, es satisfies (M1). Because s is the longest such that $Occ(e) = Occ(es)$, es satisfies (M2). Therefore $es \in \mathrm{RightMCS}(\mathcal{S}, l)$, and thus es satisfies (H1).

(H2) The proof is by contradiction. Suppose that some $r \in \mathrm{RightMCS}(\mathcal{S}, l)$ is a proper prefix of es. If $|e| \leq |r|$, e is a prefix of r. Then, $Occ(e) \supseteq Occ(r) \supseteq Occ(es)$. Since r satisfies (M2), $Occ(r) \neq Occ(es)$. This implies $Occ(e) \neq Occ(es)$, which contradicts the definition of s. Therefore $|r| < |e|$, implying that r is a proper prefix of e. Since $Occ(r) \supseteq Occ(e)$, for some $(i, j) \in Occ(e)$, $(i, j+|r|) \in Occ(r)+|r|$. Because $Occ(r)+|r| = Occ(m)+|m|$ for some $m \in \mathrm{MCS}(\mathcal{S}, l)$, $(i, j + |r|) \in \mathcal{B}(\mathcal{S}, l)$. Therefore e does not satisfy (D2), which contradicts the definition of e. Consequently, such r cannot exist. Hence es satisfies (H2).

(H3) Since e satisfies (D4), $\mathcal{B}(\mathcal{S}, l) \cap Occ(es) = \mathcal{B}(\mathcal{S}, l) \cap Occ(e) \neq \emptyset$. Thus es satisfies (H3).

The string es is h claimed in the lemma. □

By Lemma 5.1, the definition of $\mathrm{DCS}(\mathcal{S}, l)$ can be transformed as follows.

Lemma 5.2. *For a non-empty substring e of some $S_i \in \mathcal{S}$, $e \in \mathrm{DCS}(\mathcal{S}, l)$ if and only if e satisfies (D1)–(D3) and the following condition:*

(D5) For some $h \in H(\mathcal{S}, l)$, e is a prefix of h.

In other words, (D4) can be replaced with (D5).

Proof. By Lemma 5.1, any $e \in \mathrm{DCS}(\mathcal{S}, l)$ satisfies (D5). For the converse, let e be a non-empty substring of some $S_i \in \mathcal{S}$ that satisfies (D1)–(D3) and (D5). By the condition (D5), e is a prefix of some $h \in H(\mathcal{S}, l)$. Then, $Occ(e) \supseteq Occ(h)$. Since h satisfies (H3), $Occ(e) \cap \mathcal{B}(\mathcal{S}, l) \supseteq Occ(h) \cap \mathcal{B}(\mathcal{S}, l) \neq \emptyset$. Accordingly, e satisfies (D4) and therefore $e \in \mathrm{DCS}(\mathcal{S}, l)$. □

By Lemma 5.2, we can identify $\mathrm{DCS}(\mathcal{S}, l)$ by the following algorithm.

Algorithm: GET-DCS($\mathcal{S}, l$)	
Step 1:	Construct STree($\mathcal{S}$).
Step 2:	Identify RightMCS($\mathcal{S}, l$) and MCS($\mathcal{S}, l$).
Step 3:	Identify $\mathcal{B}(\mathcal{S}, l)$.
Step 4:	Identify $H(\mathcal{S}, l)$.
Step 5:	Identify DCS($\mathcal{S}, l$).

Steps 1 and 2 can be completed in $O(L)$ time [22].

Step 3: Identify $\mathcal{B}(\mathcal{S}, l)$

Clearly, $\mathcal{B}_L(\mathcal{S}, l)$ can be identified by a depth-first traversal on STree($\mathcal{S}$). Let us focus on $\mathcal{B}_R(\mathcal{S}, l)$. After initializing a set B to $\emptyset$, a depth-first traversal on STree($\mathcal{S}$) is conducted. For any $\mathcal{L}(i, j)$ encountered, we add $(i, j + l)$ to B if there is a node v such that $p(v) \in \mathrm{RightMCS}(\mathcal{S}, l)$ and $|p(v)| = l$ on the path from the root to $\mathcal{L}(i, j)$. When this process is completed, $B = \mathcal{B}_R(\mathcal{S}, l)$.

We show that this method correctly identifies $\mathcal{B}_R(\mathcal{S}, l)$. If (i, j) is added to B, $(i, j) \in Occ(r) + |r|$ for some $r \in \mathrm{RightMCS}(\mathcal{S}, l)$. Since some $m \in \mathrm{MCS}(\mathcal{S}, l)$ exists such that $Occ(r)+|r| = Occ(m)+|m|$, $(i, j) \in \mathcal{B}_R(\mathcal{S}, l)$. For the converse, suppose that $(i, j) \in \mathcal{B}_R(\mathcal{S}, l)$. Then, $(i, j - l) \in Occ(r)$ for some $r \in \mathrm{RightMCS}(\mathcal{S}, l)$ such that $|r| = l$.

Lemma 5.3. *Let r be a suffix of some $m \in \mathrm{MCS}(\mathcal{S}, l)$ such that $|r| = l$. Then, $r \in \mathrm{RightMCS}(\mathcal{S}, l)$.*

Proof. Since $|r| = l$, r satisfies (M1). We prove that r satisfies (M2). By contradiction, suppose that $Occ(r) = Occ(rs)$ for a string $s \in \Sigma^+$. Since r is a suffix of m wherever m occurs, $Occ(m) = Occ(ms)$. Therefore m does not satisfy (M2), which contradicts the assumption that $m \in \mathrm{MCS}(\mathcal{S}, l)$. Accordingly s cannot exist, hence r satisfies (M2). Consequently, $r \in \mathrm{RightMCS}(\mathcal{S}, l)$. □

Step 4: Identify $H(\mathcal{S}, l)$

We identify $H(\mathcal{S}, l)$ by discarding any $p(v)$ from RightMCS($\mathcal{S}, l$) if $p(v)$ does not satisfy any one of (H2) or (H3), where v is a node in STree($\mathcal{S}$).

Step 5: Identify DCS($\mathcal{S}, l$)

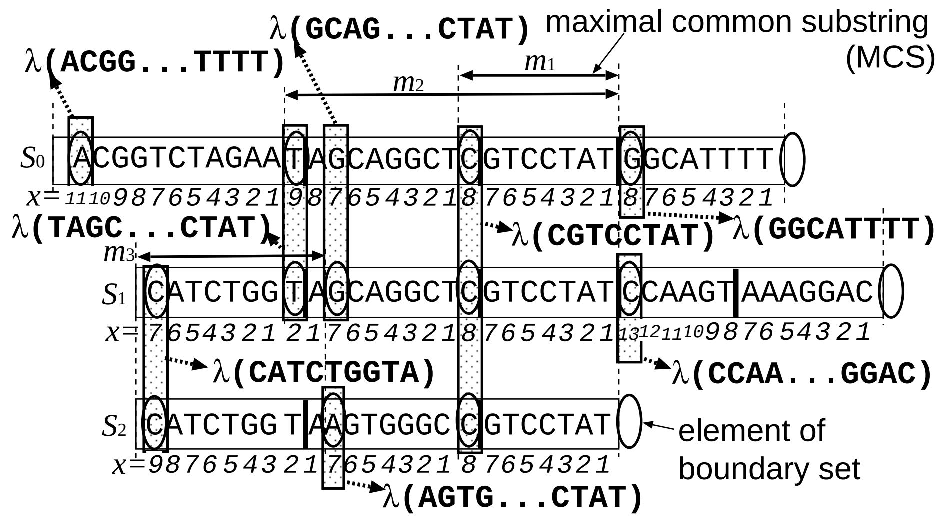
By Lemma 5.2, DCS($\mathcal{S}, l$) can be obtained by searching for substrings that satisfy (D1)–(D3) and (D5). To avoid exhaustive search, we use variables $\lambda(h)$ for each $h \in H(\mathcal{S}, l)$, and a table of pointers.

Definition 5.6 (Pointer table)**.** *$P[i, j]$ $(0 \leq i < N, 0 \leq j < |S_i|)$ is a pointer such that:*

- *$P[i, j] \to \lambda(h)$ if $(i, j) \in Occ(h)$ for some $h \in H(\mathcal{S}, l)$, where $P[i, j] \to \lambda(h)$ means $P[i, j]$ points to $\lambda(h)$,*

```
Algorithm: PREFIX-DCS($\mathcal{S}, l$)
for $i := 0$ to $N - 1$ begin
  $x := 1$, $j := |S_i| - 1$
  repeat
    if $P[i,j] \to \lambda(h)$ then $\lambda(h) := \min\{x, \lambda(h)\}$
    $x := x + 1$
    if $(i,j) \in \mathcal{B}(\mathcal{S}, l)$ then $x := 1$
    $j := j - 1$
  until $j < 0$
end
```

Figure 5.4: The algorithm PREFIX-DCS($\mathcal{S}, l$) executed in Step 5 of GET-DCS($\mathcal{S}, l$). Each $S_i \in \mathcal{S}$ is scanned from right to left by decreasing j one by one. When $\min\{x, \lambda(h)\}$ is evaluated in the inner-most loop, x is the length of the longest string s that occurs at (i,j) and satisfies $(i, j+k) \notin \mathcal{B}(\mathcal{S}, l)$ for $1 \leq k < |s|$.



λ(GCAG...CTAT)
maximal common substring (MCS)
λ(ACGG...TTTT)
m1
m2
S0
ACGGTCTAGAATAGCAGGCTCGTCCTATGGCATTTT
x=11 10 9 8 7 6 5 4 3 2 1 9 8 7 6 5 4 3 2 1 8 7 6 5 4 3 2 1 8 7 6 5 4 3 2 1
λ(TAGC...CTAT)
λ(CGTCCTAT)
λ(GGCATTTT)
m3
S1
CATCTGGTAGCAGGCTCGTCCTATCCAAGTAAAGGAC
x=7 6 5 4 3 2 1 2 1 7 6 5 4 3 2 1 8 7 6 5 4 3 2 1 13 12 11 10 9 8 7 6 5 4 3 2 1
λ(CATCTGGTA)
λ(CCAA...GGAC)
S2
CATCTGGTAAGTGGGCCGTCCTAT
element of boundary set
x=9 8 7 6 5 4 3 2 1 7 6 5 4 3 2 1 8 7 6 5 4 3 2 1
λ(AGTG...CTAT)

Figure 5.5: Behavior of the algorithm PREFIX-DCS($\mathcal{S}, l$) for strings of Example 5.1. Digits below S_0, S_1 and S_2 indicate the values of x when $\min\{x, \lambda(h)\}$ is evaluated. Hatched areas indicate positions where $P[i,j]$ is not a null pointer.

- $P[i,j]$ *is a null pointer otherwise.*

We conduct the following procedures.

1. Each $P[i,j]$ is initialized to a null pointer.
2. To set up the pointer table, conduct a depth-first traversal on STree($\mathcal{S}$). For each $\mathcal{L}(i,j)$, $P[i,j]$ is set so that $P[i,j] \to \lambda(p(v))$ if there is a node v such that $p(v) \in H(\mathcal{S}, l)$ on the path from the root to $\mathcal{L}(i,j)$.
3. For each $h \in H(\mathcal{S}, l)$, $\lambda(h)$ is initialized to $|h|$.
4. The algorithm PREFIX-DCS($\mathcal{S}, l$) in Figure 5.4 is applied. For any $h \in H(\mathcal{S}, l)$, PREFIX-DCS($\mathcal{S}, l$) sets variables $\lambda(h)$ to the lengths of prefixes of h that satisfy (D2) and (D3). We show the behavior of PREFIX-DCS($\mathcal{S}, l$) for strings of Example 5.1 in Figure 5.5.

Table 5.1: Summary statistics of DCS's and $\mathcal{T}_1$.

	DCS's	$\mathcal{T}_1$
number of strings	96,955	97,217
average length (bases)	142.3	144.9
average occurrences	3.71	3.70

When PREFIX-DCS($\mathcal{S}, l$) is completed, for each $h \in H(\mathcal{S}, l)$, the prefix of h whose length is $\lambda(h)$ is a DCS if $\lambda(h) \geq l$. All Steps 1–5 can be completed in $O(L)$ time. Therefore, we have the following result.

Theorem 5.1. *There is an algorithm that identifies* DCS($\mathcal{S}, l$) *in $O(L)$ time, where L is the sum of the lengths of all strings in $\mathcal{S}$.*

5.5 Computational Experiments

We evaluated GET-DCS($\mathcal{S}, l$) by computational experiments. We say that $e \in$ DCS($\mathcal{S}, l$) is *consistent* with a string $t \in \mathcal{T}$ if and only if $|Occ(e)| = |Occ(t)|$ and the overlap of e and t occupies at least 90% of both e and t wherever e or t occurs. Let n_{OK} be the number of strings in DCS($\mathcal{S}, l$) consistent with some $t \in \mathcal{T}$. Below *recall* means $n_{OK}/|\mathcal{T}|$, while *precision* means $n_{OK}/|\text{DCS}(\mathcal{S}, l)|$. We used a Linux server with Opteron(tm) 252 processors.

Although the primary target of this thesis is the analysis of SVs, it is difficult to apply GET-DCS($\mathcal{S}, l$) for long genome sequences because it consumes a large amount of memory because it uses suffix trees and it cannot tolerate sequencing errors and small variations. Therefore, we evaluated GET-DCS($\mathcal{S}, l$) by using simulated or real cDNA sequences.

5.5.1 Randomly Generated Strings

First, we tested GET-DCS($\mathcal{S}, l$) against randomly generated strings. Let $\mathcal{T}_1$ be a set of 97,217 random strings consisting of `A`, `T`, `G`, and `C`, whose lengths were 50–240 bases and 145 bases on average. We applied GET-DCS($\mathcal{S}, l$) to 40,000 strings, each of which was a concatenation of nine strings of $\mathcal{T}_1$. These parameters were determined to simulate the scale of coding sequences of *Homo sapiens* estimated with draft genomic sequences [37]. However, we used small $\mathcal{T}_0$ to see the ability of GET-DCS($\mathcal{S}, l$) to detect strings in $\mathcal{T}_0$ when they have enough chances to occur with different strings in $\mathcal{T}_0$. The lengths of strings in $\mathcal{S}$ were 1,305 bases on average, and the lengths of the 40,000 strings were 5.218×10^7 bases in total. We set l to 30. As shown in Table 5.1, the summary statistics of DCS($\mathcal{S}, l$) and $\mathcal{T}_1$ were very close. As shown in Table 5.2, the result was quite accurate. We examined DCS's that were not consistent with any strings in $\mathcal{T}_1$. There were 258 groups of strings in $\mathcal{T}_1$ that always occurred adjacent in $\mathcal{S}$ in the same order. In addition, there were DCS's shortened to less than 90% of corresponding strings in $\mathcal{T}_1$.

To demonstrate the scalability of our implementation of GET-DCS($\mathcal{S}, l$), we measured the increase in computation time while the number of given strings was increased. As shown in Figure 5.6, the computation time increased only linearly.

Table 5.2: Consistency of DCS's against $\mathcal{T}_1$.

n_{OK}	recall	precision
96,198	0.9895	0.9922

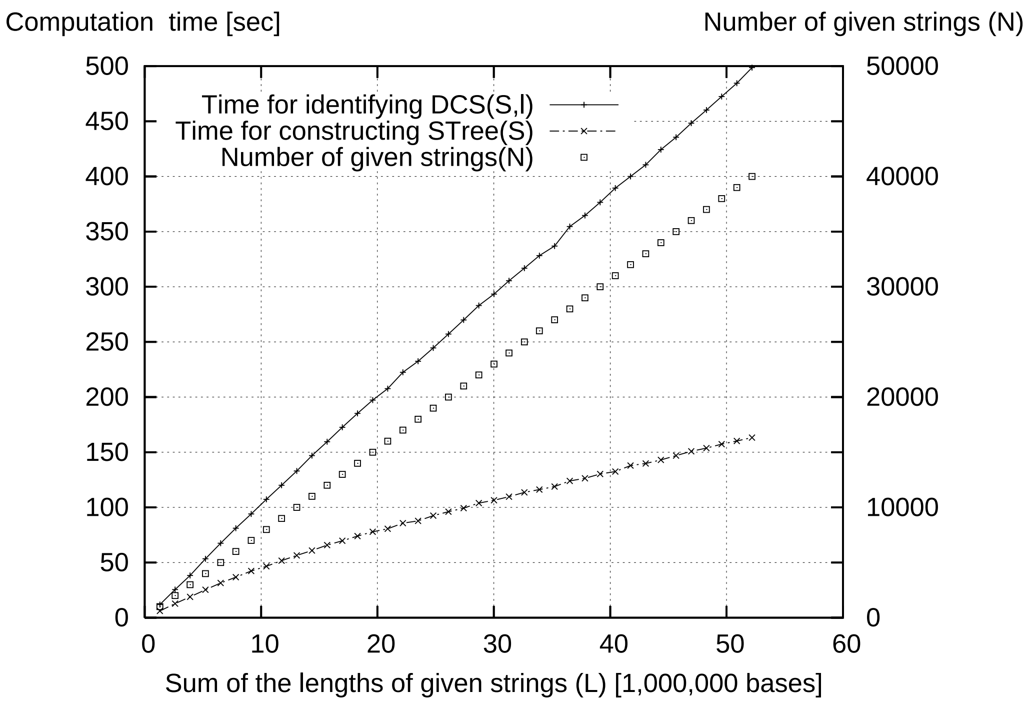


Figure 5.6: Increase in computation time to identify DCS($\mathcal{S}, l$) while the number of given strings was increased.

5.5.2 Transcriptome of *Homo sapiens*

Next, we tested our method against all cDNA sequences of *Homo sapiens* in the RefSeq database [69] of release 28. Although sequence differences were reconciled to finished genomic sequences in this database [69], they still contain plenty of complex features of real cDNA sequences. We removed consecutive `A`'s at the end of each sequence to exclude poly(A) tails. There were 25,199 sequences, whose lengths were 3,050 bases on average and 7.686×10^7 bases in total. We set l to 30. It took 826 seconds for GET-DCS($\mathcal{S}, l$) to identify DCS($\mathcal{S}, l$) from $\mathcal{S}$. For 23,777 sequences out of the 25,199 sequences, positions of exons and alternative ends of exons were available[2]. Let $\mathcal{T}_2$ be a set of strings obtained by splitting the 23,777 sequences at alternatively spliced positions. After merging substrings corresponding to exons which always occurred adjacent together in $\mathcal{S}$, a set of the substrings, denoted by $\mathcal{T}_2$, was used for evaluation of DCS's in the 23,777 sequences. In this experiment, alternative ends of exons were treated in the same way as independent exons.

As shown in Table 5.3, DCS's **in the 23,777 sequences** were much shorter than strings in $\mathcal{T}_2$ on average, while the number of DCS's was much larger than $|\mathcal{T}_2|$. As shown in row (A) of Table 5.4, the result was not satisfactory. Major causes of problems are sequence variations such as SNPs, repeated elements,

[2]`http://www.ncbi.nlm.nih.gov/mapview/`

Table 5.3: Summary statistics of DCS's and $\mathcal{T}_2$.

	DCS's	$\mathcal{T}_2$
number of strings	51,811	29,833
average length (bases)	1,004	1,813
average occurrences	1.70	1.57

Table 5.4: Consistency of DCS's against $\mathcal{T}_2$.

	n_{OK}	recall	precision
(A)	21,803	0.7308	0.4208
(B)	23,798	0.7977	0.6278

and family genes sharing long identical regions irrelevant to alternative splicing. To partly circumvent these problems, we merged DCS's that always occurred adjacent in the same order and removed DCS's that occurred at least twice in a sequence Then, we obtained an improved result shown in row (B) of Table 5.4. One direction to overcome the third problem is to combine information of more than one DCS. When two sequences in $\mathcal{S}$ share a DCS, investigating whether they share other DCS's is a way to discriminate DCS's irrelevant to alternative splicing. Another way to screen out erroneous DCS's is to examine whether the order of DCS's is preserved in more than one sequence in $\mathcal{S}$.

5.5.3 Comparison with MSA Program POA

We compared the accuracy of our method with that of an MSA program POA [39]. Although we tried several MSA programs, all except POA suffered from weak similarities between irrelevant substrings. As a test data set, we picked up 21 cDNA sequences of the cAMP-responsive element modulator (CREM) gene from the data set of the previous experiment. Their lengths were 41,535 bases in total. We set the mismatch parameter of POA to a huge negative value (-10^6). For GET-DCS($\mathcal{S}, l$), we set l to 20.

As shown in Table 5.5, results of our method and that of POA were quite consistent with to exons of the gene, although exons 7 and $8'$ were fused since they always occurred together. Both methods wrongly dropped 10 bases at 5'-end of exons $14'$ and $14''$ due to their identical 10-base prefixes. POA added the 10 bases to the 3'-ends of exons 13 and $14'$, while our method excluded these 10 bases from any DCS since they are ambiguous. It took only 0.035 seconds for our method to obtain the result, while it took 97.797 seconds for POA. Our method were about 2800 times faster. Note that POA had to be repeatedly executed to find parameters that enabled POA to produce satisfactory results.

5.6 Summary

We considered the problem of inferring a set $\mathcal{T}$ of hidden strings from a set $\mathcal{S}$ of their concatenations, and proposed a linear time algorithm. If there is possibility that given sequences are concatenations of an unknown set of strings, it is worth trying our method to identify such set of strings.

Table 5.5: Results of our method and POA against the CREM gene. The exons 8, 11 and 14 had alternative ends, which were treated as independent exons. For example, exon 8 was divided into $8'$ and $8''$. Lengths of extra bases not in exons are in parentheses.

exon	length of exons	overlap with DCS's	overlap with POA substrings
1	321	320(0)	321(0)
2	108	108(0)	108(0)
3	98	97(0)	98(0)
4	124	122(0)	122(0)
5	265	263(0)	263(0)
6	110	109(0)	109(0)
$7–8'$	98+143	241(1)	241(1)
$8''$	571	570(0)	570(0)
9	189	187(0)	188(0)
10	88	86(0)	87(0)
$11'$	198	196(0)	198(0)
$11''$	43	40(0)	42(0)
12	36	33(0)	35(0)
13	157	154(0)	157(10)
$14'$	402	389(0)	392(10)
$14''$	1302	1292(0)	1292(0)

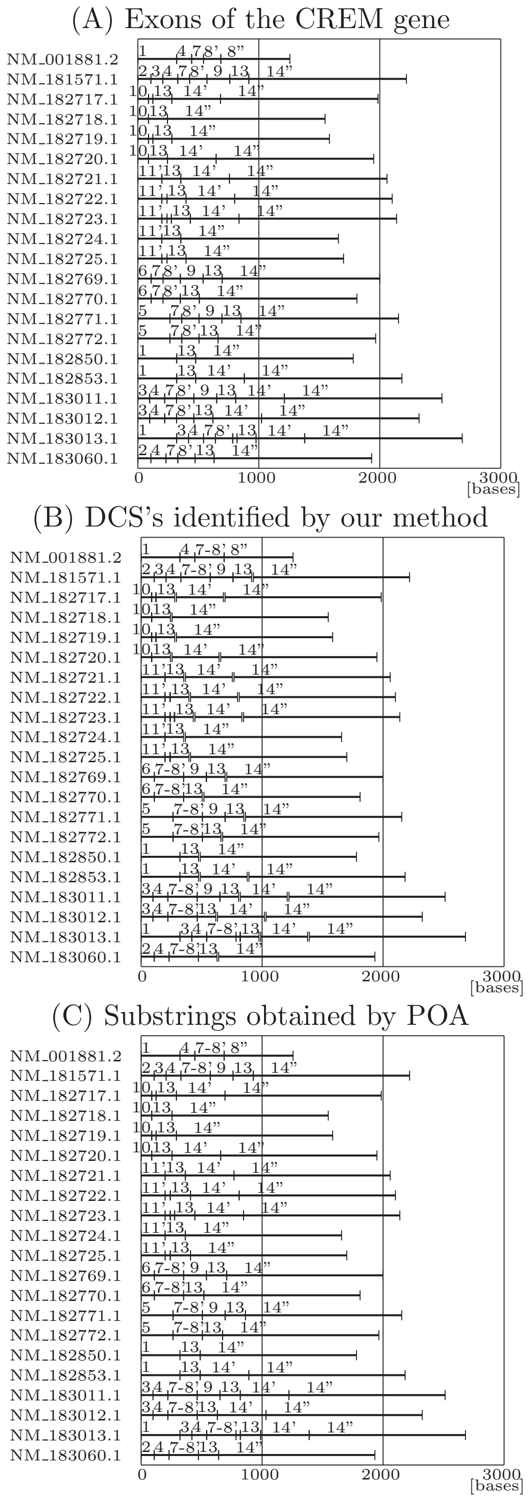


Figure 5.7: (A) Exons of the CREM gene, (B) DCS's identified by GET-DCS($\mathcal{S}, l$), and (C) substrings obtained by POA. Numbers over substrings of sequences indicate corresponding exons. Since there are no spaces, numbers $11''$ and 12 are not shown.

Chapter 6

Conclusions

The next-generation sequencing (NGS) technologies enabled us to exhaustively detect structural variations (SVs). Continuing technological innovations in DNA sequencing will, in future, provide predictions of even more SVs. However, detecting SVs is still a difficult problem. In addition, each SVs are local phenomena in genomes and only detecting SVs is not sufficient to reveal global structure of chromosomes. This thesis addressed wide range of problems required in the analysis of SVs.

First, this thesis formulated the problem of inferring global structures of chromosomes as the *chromosome problem (ChrP)*. This is an optimization problem to search for an optimal set of chromosomes that minimizes a cost function that takes into account the number of chromosomes and detected SVs. In addition, the length of each chromosome is bounded by the estimated length. ChrP was proved to be NP-complete. We also proposed a biologically meaning restriction on instances of the problem, which we term as the *weakly connected constraint (WCC)*. By using WCC, we defined a variation of ChrP, termed as ChrW. Instances of ChrW is restricted by WCC and the length of chromosomes is not bounded. We proposed an algorithm that solves ChrW in polynomial time by reducing the problem into a cycle-finding problem on a bidirected graph. We also considered another variation, termed as ChrL, which is the same as ChrW except that the length of chromosomes are bounded as ChrP. Because ChrL is NP-complete, it is clear that removing upper bounds on the length of chromosomes is necessary to make ChrW solvable in polynomial time. This work provides a theoretical basis for the development of practical computational tools that are emerging for use in the analysis of the global structure of chromosomes based on SVs. In computational experiments with simulated SV data, our algorithm that solves ChrW was confirmed to be able to mitigate noise added to simulated SV data, where noise contains modified CNVs or false positive translocations. It was also confirmed that considering the number of chromosomes and truncations was effective to infer an accurate set of chromosomes. However, it was difficult to handle missing translocations. In addition, further evaluation for real SV data is necessary.

Second, in order to accurately detect breakpoints, i.e. positions of boundaries of SVs, this thesis proposed a new method ChopSticks which improves the resolution of breakpoints of homozygous deletions by exploiting paired reads whose mapping distances and strands are normal. When the depth of coverage approaches zero or infinity, the resolution of our method approaches to that of RP methods with doubled amount of NGS sequences. In computational experiments with simulated NGS sequences, ChopSticks successfully improved the resolution of BreakDancer [10], MoDIL [40], CLEVER [54], and CNVnator [1]. In computational experiments with real NGS sequences, ChopSticks also successfully improved the resolution of BreakDancer and CLEVER.

Third, because mapping NGS sequences to the reference genome is a necessary and computationally intensive step of most SV detection methods, we conducted mapping with a many-core processor Xeon Phi toward performance improvement in the future. Because a lot of programs that map NGS sequences to the reference genome have already been proposed, developing a new program may confuse users. Therefore, we decided to port two widely used programs, Burrows-Wheeler Aligner (BWA) [45] and Bowtie2 [38], to Xeon Phi. A major obstacle of porting was incompatibility of vector operations between Xeon Phi and x86 processors for which BWA and Bowtie2 are originally developed. We replaced 8-bit operations and 16-bit operations with 32-bit operations, and overcame other incompatibilities one by one. We also circumvented an incompatibility found in the `sort` function in the standard template library (STL). In computational experiments, the performance of ported programs was almost proportional to the number of thread increases up to 60. This result was promising because the many-core architecture of Xeon Phi turned out to suit to mapping NGS sequences to the reference genome. Although the peak performance was still inferior to that of a normal x86 CPU, we believe that the improvement of the ported program and the architecture of Xeon Phi will achieve better performance in the future.

Finally, in order to compare assembled genomic sequences for the purpose of detecting SVs, it is necessary to identify common substrings and substrings specific to each string that resulted from rearrangements. Therefore, we addressed a problem in which a set $\mathcal{T}$ of strings are inferred from a set $\mathcal{S}$ of their concatenations. We defined a set $\mathrm{DCS}(\mathcal{S}, l)$ of strings each of which corresponds to a string in $\mathcal{T}$ or a concatenation of strings in $\mathcal{T}$ that always occur adjacent in the same order, where l is a positive integer parameter. In addition, we developed an algorithm that identifies $\mathrm{DCS}(\mathcal{S}, l)$. We proved that $\mathrm{DCS}(\mathcal{S}, l)$ can be identified within $O(L)$ time by comparing strings in $\mathcal{S}$, where L is the sum of the lengths of all strings in $\mathcal{S}$. In a computational experiment, a set of 40,000 randomly generated strings were successfully decomposed into substrings of which they are concatenations. In addition, the cDNA sequences of the human CREM gene were also decomposed into exons only with minor errors by our method about 2,800 times faster than by a multiple sequence alignment (MSA) program POA, while other MSA programs suffered from weak similarities between different exons.

Future Directions

As a future work, we would like to improve the proposed methods so that they can be applied to a wide range of real data.

First, inference of the global structure of chromosomes currently requires that given instances satisfy WCC to calculate an optimal solution in polynomial time. Because SV data do not usually include data to satisfy WCC, additional data have to be collected by examining raw NGS sequences again or by conducting additional biological experiments. To overcome this problem, we need a method that infers ends of chromosomes and highly probable adjacencies of segments in the target genome from SV data. Another direction is to develop an approximation algorithm that are available to instances that do not satisfy the proposed WCC. Handling missing translocations is another problem that should be overcome. It is also necessary to further evaluate the effectiveness of our algorithm for real SV data.

Second, because ChopSticks is currently available to only homozygous deletions, it should be improved so that it can cope with other types of SVs. By adding a step that distinguishes homozygous deletions from heterozygous ones and to apply ChopSticks to the former, ChopSticks can be used in applications where organisms other than inbred mice are analyzed.

Third, as we explained in Chapter 4, the performances of BWA and Bowtie2 on Xeon Phi are expected to be further improved by three ways: (i) by fully exploiting computation power of Xeon Phi, e.g. vector registers not used in this study, (ii) by using Xeon Phi with x86 processors in a coordinated manner, and (iii) by improving our ported codes.

Finally, to apply the algorithm that identifies DCS$(\mathcal{S}, l)$ to a large-scale real data, the algorithm have to tolerate sequencing errors and sequence variations in genomes, as well as repeated elements and identical regions of sequences of different genes. Because the algorithm involves construction of a suffix tree for given strings, eliminating consumption of memory is also preferable to be applied to long genome sequences. The latter improvement would be achieved by using the technique of compressed full-text indexes [49, 62].

References

[1] A. Abyzov, A. E. Urban, M. Snyder, and M. Gerstein. CNVnator: An approach to discover, genotype, and characterize typical and atypical CNVs from family and population genome sequencing. *Genome Res*, 21(6):974–984, 2011.

[2] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice Hall, New Jersey, 1993.

[3] Rena Akahori, Takanobu Haga, Toshiyuki Hatano, Itaru Yanagi, Takeshi Ohura, Hirotaka Hamamura, Tomio Iwasaki, Takahide Yokoi, and Takashi Anazawa. Slowing single-stranded DNA translocation through a solid-state nanopore by decreasing the nanopore diameter. *Nanotechnology*, 25(27):275501, 2014.

[4] T. Akutsu, H. Arimura, and S. Shimozono. Hardness results on local multiple alignment of biological sequences. *IPSJ Trans. Bioinformatics*, 48(SIG 5(TBIO 2)):30–38, 2007.

[5] Sylvan C. Baca, Davide Prandi, Michael S. Lawrence, Juan Miguel Mosquera, Alessandro Romanel, Yotam Drier, Kyung Park, Naoki Kitabayashi, Theresa Y. MacDonald, Mahmoud Ghandi, Eliezer Van Allen, Gregory V. Kryukov, Andrea Sboner, Jean-Philippe Theurillat, T. David Soong, Elizabeth Nickerson, Daniel Auclair, Ashutosh Tewari, Himisha Beltran, Robert C. Onofrio, Gunther Boysen, Candace Guiducci, Christopher E. Barbieri, Kristian Cibulskis, Andrey Sivachenko, Scott L. Carter, Gordon Saksena, Douglas Voet, Alex H. Ramos, Wendy Winckler, and et al. Punctuated evolution of prostate cancer genomes. *Cell*, 153(3):666–677, 2013.

[6] A. Bashir, S. Volik, C. Collins, V. Bafna, and B. J. Raphael. Evaluation of paired-end sequencing strategies for detection of genome rearrangements in cancer. *PLoS Comput Biol*, 4(4):e1000051, 2008.

[7] G. Blackshields, I.M. Wallace, M. Larkin, and D.G. Higgins. Analysis and comparison of benchmarks for multiple sequence alignment. *In Silico Biology*, 6(4):321–339, 2006.

[8] P.J. Campbell, P.J. Stephens, E.D. Pleasance, S. O'Meara, H. Li, T. Santarius, L.A. Stebbings, C. Leroy, S. Edkins, C. Hardy, J.W. Teague, A. Menzies, I. Goodhead, D.J. Turner, C.M. Clee, M.A. Quail, A. Cox, C. Brown, R. Durbin, M.E. Hurles, P.A.W. Edwards, G.R. Bignell, M.R. Stratton, and P.A. Futreal. Identification of somatically acquired rearrangements in cancer using genome-wide massively parallel paired-end sequencing. *Nat Genet*, 40(6):722–729, 2008.

[9] Mark J. Chaisson, Dumitru Brinza, and Pavel A. Pevzner. De novo fragment assembly with short mate-paired reads: Does the read length matter? *Genome Res*, 19:336–346, 2009.

[10] Ken Chen, John W. Wallis, Michael D. McLellan, David E. Larson, Joelle M. Kalicki, Craig S. Pohl, Sean D. McGrath, Michael C. Wendl, Qunyuan Zhang, Devin P. Locke, Xiaoqi Shi, Robert S. Fulton, Timothy J. Ley, Richard K. Wilson, Li Ding, and Elaine R. Mardis. BreakDancer: an algorithm for high-resolution mapping of genomic structural variation. *Nat Methods*, 6(9):677–681, 2009.

[11] X. Chen, J. Zheng, Z. Fu, P. Nan, Y. Zhong, S. Lonardi, and T. Jiang. Computing the assignment of orthologous genes via genome rearrangement. In *Proc. 3rd Asia-Pacific Bioinformatics Conference (APBC)*, pages 363–378, 2005.

[12] M. Chrobak, P. Kolman, and J. Sgall. The greedy algorithm for the minimum common string partition problem. *ACM Trans. Algorithms (TALG)*, 1(2):350–366, 2005.

[13] The International Cancer Genome Consortium. International network of cancer genome projects. *Nature*, 464(7291):993–998, 2010.

[14] John Eid, Adrian Fehr, Jeremy Gray, Khai Luong, John Lyle, Geoff Otto, Paul Peluso, David Rank, Primo Baybayan, Brad Bettman, Arkadiusz Bibillo, Keith Bjornson, Bidhan Chaudhuri, Frederick Christians, Ronald Cicero, Sonya Clark, Ravindra Dalal, Alex deWinter, John Dixon, Mathieu Foquet, Alfred Gaertner, Paul Hardenbol, Cheryl Heiner, Kevin Hester, David Holden, Gregory Kearns, Xiangxu Kong, Ronald Kuse, Yves Lacroix, Steven Lin, Paul Lundquist, Congcong Ma, Patrick Marks, Mark Maxham, Devon Murphy, Insil Park, Thang Pham, Michael Phillips, Joy Roy, Robert Sebra, Gene Shen, Jon Sorenson, Austin Tomaney, Kevin Travers, Mark Trulson, John Vieceli, Jeffrey Wegener, Dawn Wu, Alicia Yang, Denis Zaccarin, Peter Zhao, Frank Zhong, Jonas Korlach, and Stephen Turner. Real-time dna sequencing from single polymerase molecules. *Science*, 323(5910):133–138, 2009.

[15] Michael Farrar. Striped smith-waterman speeds database searches six times over other simd implementations. *Bioinformatics*, 23(2):156–161, 2007.

[16] Harold N. Gabow. An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. *In Proceedings of the 15-th annual ACM symposium on Theory of computing (STOC)*, pages 448–456, 1983.

[17] M. R. Garey and D. S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W.H. Freeman & Company, New York, 1979.

[18] Éric Gaul and Mathieu Blanchette. Ordering partially assembled genomes using gene arrangements. In Guillaume Bourque and Nadia El-Mabrouk, editors, *Comparative Genomics*, volume 4205 of *Lecture Notes in Computer Science*, pages 113–128. Springer Berlin Heidelberg, Germany, 2006.

[19] Sante Gnerre, Iain MacCallum, Dariusz Przybylski, Filipe J. Ribeiro, Joshua N. Burton, Bruce J. Walker, Ted Sharpe, Giles Hall, Terrance P. Shea, Sean Sykes, Aaron M. Berlin, Daniel Aird, Maura Costello, Riza Daza, Louise Williams, Robert Nicol, Andreas Gnirke, Chad Nusbaum, Eric S. Lander, and David B. Jaffe. High-quality draft assemblies of mammalian genomes from massively parallel sequence data. *Proc Natl Acad Sci USA*, 108(4):1513–1518, 2011.

[20] A. Goldstein, P. Kolman, and J. Zheng. Minimum common string partition problem: hardness and approximations. In *Proc. 15th International Symp. Algorithms and Computation (ISAAC)*, pages 473–484, 2004.

[21] Osamu Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162(3):705 – 708, 1982.

[22] D. Gusfield. *Algorithms on strings, trees, and sequences.* Cambridge University Press, New York, 1997.

[23] RE Handsaker, JM Korn, J. Nemesh, and SA McCarroll. Discovery and genotyping of genome structural polymorphism by sequencing on a population scale. *Nat Genet*, 43(3):269–276, 2011.

[24] Ayat Hatem, Doruk Bozdag, Amanda Toland, and Umit Catalyurek. Benchmarking short sequence mapping tools. *BMC Bioinformatics*, 14:184, 2013.

[25] F. Hormozdiari, C. Alkan, E. E. Eichler, and S. C. Sahinalp. Combinatorial algorithms for structural variation detection in high-throughput sequenced genomes. *Genome Res*, 19(7):1270–1278, 2009.

[26] Sahar Jahani and Seyed Kamaledin Setarehdan. Centromere and length detection in artificially straightened highly curved human chromosomes. *International Journal of Biological Engineering*, 2(5):56–61, 2012.

[27] Fumio Kasai, Patricia C.M. O'Brien, and Malcolm A. Ferguson-Smith. Afrotheria genome; overestimation of genome size and distinct chromosome GC content revealed by flow karyotyping. *Genomics*, 102(5-6):468–471, 2013.

[28] Jaebum Kim, Denis M. Larkin, Qingle Cai, Asan, Yongfen Zhang, Ri-Li Ge, Loretta Auvil, Boris Capitanu, Guojie Zhang, Harris A. Lewin, and Jian Ma. Reference-assisted chromosome assembly. *Proc Natl Acad Sci USA*, 110(5):1785–1790, 2013.

[29] Wigard P. Kloosterman, Victor Guryev, Mark van Roosmalen, Karen J. Duran, Ewart de Bruijn, Saskia C. M. Bakker, Tom Letteboer, Bernadette van Nesselrooij, Ron Hochstenbach, Martin Poot, and Edwin Cuppen. Chromothripsis as a mechanism driving complex de novo structural rearrangements in the germline. *Human Molecular Genetics*, 20(10):1916–1924, May 2011.

[30] W.P. Kloosterman, M. Tavakoli-Yaraki, M.J. van Roosmalen, E. van Binsbergen, I. Renkens, K. Duran, L. Ballarati, S. Vergult, D. Giardino, K. Hansson, C.A.L. Ruivenkamp, M. Jager, A. van Haeringen, E.F. Ippel, T. Haaf, E. Passarge, R. Hochstenbach, B. Menten, L. Larizza, V. Guryev, M. Poot, and E. Cuppen. Constitutional chromothripsis rearrangements involve clustered double-stranded DNA breaks and nonhomologous repair mechanisms. *Cell Rep*, 1(6):648–55, 2012.

[31] Petr Klus, Simon Lam, Dag Lyberg, Ming Cheung, Graham Pullan, Ian McFarlane, Giles Yeo, and Brian Lam. Barracuda - a fast short read sequence aligner using graphics processing units. *BMC Research Notes*, 5(1):27, 2012.

[32] Martin Krzywinski, Jacqueline Schein, nan Birol, Joseph Connors, Randy Gascoyne, Doug Horsman, Steven J. Jones, and Marco A. Marra. Circos: An information aesthetic for comparative genomics. *Genome Res*, 19(9):1639–1645, 2009.

[33] Vamsi Kundeti, Sanguthevar Rajasekaran, and Heiu Dinh. An efficient algorithm for chinese postman walk on bi-directed de bruijn graphs. *Proc. Combinatorial optimization and applications (COCOA)*, pages 184–196, 2010.

[34] Myriam Kurtz, Francisco J. Esteban, Pilar Hernández, Juan A. Caballero, Antonio Guevara, Gabriel Dorado, and Sergio Gálvez. Many-core Tile64 vs. multi-core Intel Xeon: Bioinformatics performance comparison. In *VI Latin American Symposium on High Performance Computing HPCLatAm 2013*, 2013.

[35] Hugo Y. K. Lam, Xinmeng Jasmine Mu, Adrian M. Stutz, Andrea Tanzer, Philip D. Cayting, Michael Snyder, Philip M. Kim, Jan O. Korbel, and Mark B. and Gerstein. Nucleotide-resolution analysis of structural variants using breakseq and a breakpoint library. *Nature Biotechnology*, 28:47–55, 2010.

[36] E. S. Lander and M. S. Waterman. Genomic mapping by fingerprinting random clones: a mathematical analysis. *Genomics*, 2(3):231–239, 1988.

[37] E.S. Lander et al. Initial sequencing and analysis of the human genome. *Nature*, 409:860–921, 2001.

[38] Ben Langmead and Steven L. Salzberg. Fast gapped-read alignment with Bowtie 2. *Nat Meth*, 9(4):357–359, April 2012.

[39] C. Lee, C. Grasso, and M.F. Sharlow. Multiple sequence alignment using partial order graphs. *Bioinformatics*, 18(3):452–464, 2002.

[40] Seunghak Lee, Fereydoun Hormozdiari, Can Alkan, and Michael Brudno. Modil: detecting small indels from clone-end sequencing with mixtures of distributions. *Nat Methods*, 6:473–474, 2009.

[41] J. K. Lenstra and A. H. G. Rinnooy Kan. On general routing problems. *Networks*, 6(3):273–280, 1976.

[42] MY Leung, BE Blaisdell, C. Burge, and S. Karlin. An efficient algorithm for identifying matches with errors in multiple long molecular sequences. *J. Molecular Biology*, 221(4):1367–1378, 1991.

[43] Heng Li and Richard Durbin. Fast and accurate short read alignment with burrows-wheeler transform. *Bioinformatics*, 25(14):1754–1760, 2009.

[44] Heng Li and Nils Homer. A survey of sequence alignment algorithms for next-generation sequencing. *Briefings in Bioinformatics*, 11(5):473–483, 2010.

[45] R. Li, H. Zhu, J. Ruan, W. Qian, X. Fang, Z. Shi, Y. Li, S. Li, G. Shan, K. Kristiansen, S. Li, H. Yang, Jian Wang, and Jun Wang. Fast and accurate short read alignment with burrows-wheeler transform. *Bioinformatics*, 25:1754–1760, 2009.

[46] R. Li, H. Zhu, J. Ruan, W. Qian, X. Fang, Z. Shi, Y. Li, S. Li, G. Shan, K. Kristiansen, S. Li, H. Yang, Jian Wang, and Jun Wang. De novo assembly of human genomes with massively parallel short read sequencing. *Genome Res*, 20(2):265–272, 2010.

[47] Yang Liu, Jiang-Yu Li, Yi-Qing Mao, Xiao-Lei Wang, and Dong-Sheng Zhao. A literature evaluation of CUDA compatible sequence aligners. In *Bioinformatics 2013*, 2013.

[48] D. Lopresti and A. Tomkins. Block edit models for approximate string matching. *Theoretical Computer Science*, 181(1):159–179, 1997.

[49] Veli Mäkinen and Gonzalo Navarro. Dynamic entropy-compressed sequences and full-text indexes. *ACM Trans Algorithms*, 4(3):32:1–32:38, 2008.

[50] Ankit Malhotra, Michael R. Lindberg, Greg G. Faust, Mitchell L. Leibowitz, Royden A. Clark, Ryan Layer, Aaron R. Quinlan, and Ira M. Hall. Breakpoint profiling of 64 cancer genomes reveals numerous complex rearrangements spawned by homology-independent mechanisms. *Genome Res*, 23(5):762–776, 2013.

[51] Lira Mamanova, Alison J. Coffey, Carol E. Scott, Iwanka Kozarewa, Emily H. Turner, Akash Kumar, Eleanor Howard, Jay Shendure, and Daniel J. Turner. Target-enrichment strategies for next-generation sequencing. *Nat Methods*, 7:111–118, 2010.

[52] Svetlin Manavski and Giorgio Valle. CUDA compatible GPU cards as efficient hardware accelerators for smith-waterman sequence alignment. *BMC Bioinformatics*, 9(Suppl 2):S10, 2008.

[53] T. Maniatis and B. Tasic. Alternative pre-mRNA splicing and proteome expansion in metazoans. *Nature*, 418:236–243, 2002.

[54] Tobias Marschall, Ivan Costa, Stefan Canzar, Markus Bauer, Gunnar W. Klau, Alexander Schliep, and Alexander Schnhuth. Clever: Clique-enumerating variant finder. *Bioinformatics*, 2012.

[55] P. Medvedev, M. Stanciu, and M. Brudno. Computational methods for discovering structural variation with next-generation sequencing. *Nat Methods*, 6:S13–S20, 2009.

[56] Paul Medvedev and Michael Brudno. Maximum likelihood genome assembly. *J Comput Biol*, 16(8):1101–1116, 2009.

[57] Paul Medvedev, Marc Fiume, Misko Dzamba, Tim Smith, and Michael Brudno. Detecting copy number variation with mated short reads. *Genome Res*, 20(11):1613–1622, 2010.

[58] Ryan E. Mills, Klaudia Walter, Chip Stewart, Robert E. Handsaker, Ken Chen, Can Alkan, Alexej Abyzov, Seungtai Chris Yoon, Kai Ye, R. Keira Cheetham, Asif Chinwalla, Donald F. Conrad, Yutao Fu, Fabian Grubert, Iman Hajirasouliha, Fereydoun Hormozdiari, Lilia M. Iakoucheva, Zamin Iqbal, Shuli Kang, Jeffrey M. Kidd, Miriam K. Konkel, Joshua Korn, Ekta Khurana, Deniz Kural, Hugo Y. K. Lam, Jing Leng, Ruiqiang Li, Yingrui Li, Chang-Yun Lin, Ruibang Luo, et al., and 1000 genomes project. Mapping copy number variation by population-scale genome sequencing. *Nature*, 470(7332):59–65, 2011.

[59] Mouse genome sequencing consortium. Initial sequencing and comparative analysis of the mouse genome. *Nature*, 420:520–562, 2002.

[60] Eugene W. Myers. The fragment assembly string graph. *Bioinformatics*, 21:ii79–ii85, 2005.

[61] Niranjan Nagarajan and Mihai Pop. Parametric complexity of sequence assembly: Theory and applications to next generation sequencing. *J Comput Biol*, 16(7):897–908, 2009.

[62] Gonzalo Navarro and Veli Mäkinen. Compressed full-text indexes. *ACM Comput Surv*, 39(1):Article No. 2, 2007.

[63] Nicholas Navin, Jude Kendall, Jennifer Troge, Peter Andrews, Linda Rodgers, Jeanne McIndoo, Kerry Cook, Asya Stepansky, Dan Levy, Diane Esposito, Lakshmi Muthuswamy, Alex Krasnitz, W. Richard McCombie, James Hicks, and Michael Wigler. Tumour evolution inferred by single-cell sequencing. *Nature*, 472(7341):90–94, 2011.

[64] SB Needleman and CD Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Molecular Biology*, 48(3):443–453, 1970.

[65] J. Néraud. Elementariness of a finite set of words is co-NP-complete. *Theoretical Informatics and Applications*, 24(5):459–470, 1990.

[66] Layla Oesper, Anna Ritz, Sarah Aerni, Ryan Drebin, and Benjamin Raphael. Reconstructing cancer genomes from paired-end sequencing data. *BMC Bioinformatics*, 13(Suppl 6):S10, 2012.

[67] Pavel A. Pevzner, Haixu Tang, and Michael S. Waterman. An Eulerian path approach to DNA fragment assembly. *Proc. Natl. Acad. Sci.*, 98:9748–9753, 2001.

[68] Mihai Pop. Genome assembly reborn: recent computational challenges. *Brief Bioinform*, 10(4):354–366, 2009.

[69] Pruitt, K.D., Tatusova, T. and Maglott D.R. The Reference Sequence (RefSeq) Project. In *The NCBI Handbook*, chapter 18. NCBI, 2002.

[70] A. R. Quinlan, R. A. Clark, S. Sokolova, M. L. Leibowitz, Y. Zhang, M. E. Hurles, J. C. Mell, and I. M. Hall. Genome-wide mapping and assembly of structural variant breakpoints in the mouse genome. *Genome Res*, 20(5):623–635, 2010.

[71] A. R. Quinlan and I. M. Hall. BEDTools: a flexible suite of utilities for comparing genomic features. *Bioinformatics*, 26(6):841–842, 2010.

[72] Eric W. Sayers, Tanya Barrett, Dennis A. Benson, Evan Bolton, Stephen H. Bryant, Kathi Canese, Vyacheslav Chetvernin, Deanna M. Church, Michael DiCuccio, Scott Federhen, Michael Feolo, Ian M. Fingerman, Lewis Y. Geer, Wolfgang Helmberg, Yuri Kapustin, David Landsman, David J. Lipman, Zhiyong Lu, Thomas L. Madden, Tom Madej, Donna R. Maglott, Aron Marchler-Bauer, Vadim Miller, Ilene Mizrachi, James Ostell, Anna Panchenko, Lon Phan, Kim D. Pruitt, Gregory D. Schuler, Edwin Sequeira, and et al. Database resources of the national center for biotechnology information. *Nuc Acids Res*, 39(Suppl 1):D38–D51, 2011.

[73] Michael M. Shen. Chromoplexy: A new category of complex rearrangements in the cancer genome. *Cancer Cell*, 23(5):567–569, 2013.

[74] Jared T. Simpson, Kim Wong, Shaun D. Jackman, Jacqueline E. Schein, Steven J.M. Jones, and İnanç Birol. ABySS: A parallel assembler for short read sequence data. *Genome Res*, 19(6):1117–1123, 2009.

[75] T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195 – 197, 1981.

[76] TF Smith and MS Waterman. Identification of common molecular subsequences. *J. Molecular Biology*, 147:195–197, 1981.

[77] Manuel Sorge, René van Bevern, Rolf Niedermeier, and Mathias Weller. A new view on rural postman based on Eulerian extension and matching. *Journal of Discrete Algorithms*, 16:12–33, 2012.

[78] Philip J. Stephens, Chris D. Greenman, Beiyuan Fu, Fengtang Yang, Graham R. Bignell, Laura J. Mudie, Erin D. Pleasance, King Wai Lau, David Beare, Lucy A. Stebbings, et al. Massive genomic rearrangement acquired in a single catastrophic event during cancer development. *Cell*, 144(1):27–40, 2011.

[79] The 1000 genomes project consortium. A map of human genome variation from population-scale sequencing. *Nature*, 467:1061–1073, 2010.

[80] Harold Timbleby. The directed chinese postman problem. *Software – Practice & Experience*, 33(11):1081–1096, 2003.

[81] L. Wang and T. Jiang. On the complexity of multiple sequence alignment. *J Comput Biol*, 1(4):337–348, 1994.

[82] Joachim Weischenfeldt, Orsolya Symmons, François Spitz, and Jan O. Korbel. Phenotypic impact of genomic structural variation: insights from and for human disease. *Nat Rev Genet*, 14(2):125–138, 2013.

[83] Tomohiro Yasuda. A linear time algorithm that infers hidden strings from their concatenations. *IPSJ Trans. Bioinformatics*, 1:13–22, 2008.

[84] Tomohiro Yasuda and Asako Koike. Genome Mapping by a 60-core Processor. In *Proceedings of the Fifth International Conference on Bioinformatics models, methods, and algorithms*, pages 227–232, 2014.

[85] Tomohiro Yasuda and Satoru Miyano. Inferrng global structure of chromosomes from structural variations, 2015. In press.

[86] Tomohiro Yasuda, Shin Suzuki, Masao Nagasaki, and Satoru Miyano. Chopsticks: High-resolution analysis of homozygous deletions by exploiting concordant read pairs. *BMC Bioinformatics*, 13(1):279, 2012.

[87] K. Ye, M. H. Schulz, Q. Long, R. Apweiler, and Z. Ning. Pindel: a pattern growth approach to detect break points of large deletions and medium sized insertions from paired-end short reads. *Bioinformatics*, 25(21):2865–2871, 2009.

[88] Daniel R. Zerbino and Ewan Birney. Velvet: Algorithms for de novo short read assembly using de bruijn graphs. *Genome Res*, 18:821–829, 2008.

[89] J. Zhang and Y. Wu. SVseq: an approach for detecting exact breakpoints of deletions with low-coverage sequence data. *Bioinformatics*, 27(23):3228–3234, 2011.

[90] Z. Zhang, S. Schwartz, L. Wagner, and W. Miller. A greedy algorithm for aligning DNA sequences. *J. Comp Biol*, 7:203–214, 2000.