UNIVERSITY OF CALIFORNIA, SAN DIEGO

**Genome Assembly and Comparison**

A dissertation submitted in partial satisfaction of the
requirements for the degree of Doctor of Philosophy

in

Computer Science

by

Kim Son Pham

Committee in charge:

Professor Pavel Pevzner, Chair
Professor Vineet Bafna
Professor Ronald Graham
Professor Ramamohan Paturi
Professor Glenn Tesler

2013

The Dissertation of Kim Son Pham is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

---

---

---

---

---

Chair

University of California, San Diego

2013

DEDICATION

*To my family*

EPIGRAPH

I never regretted it, although for some time I had to supplement my income at NII-GENETIKA by gathering empty bottles at Moscow railway stations, one of the very few legal ways to make extra money in pre-perestroika Moscow.

*Pavel A. Pevzner*

Happiness is here and now.

*Thich Nhat Hanh*

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

In March 2008, holding 5 offer letters from prestigious universities, I was confused and did not know which one would be best for my PhD studies. But a single sentence from a bioinformatics textbook that I was reading immediately resolved the confusion.

*"I never regretted it, although for some time I had to supplement my income at NII-GENETIKA by gathering empty bottles at Moscow railway stations, one of the very few legal ways to make extra money in pre-perestroika Moscow"*.

I've never regretted the choice of going to UCSD and becoming a student of Pavel Pevzner. I've learned from him not only how to write a paper, to look at the problem from many angles, to turn the solution into the simplest form possible, but also how to enjoy the academic life. I would like to express my deep gratitude to Pavel for his inspiration, guidance and support during my PhD years.

I would like to sincerely thank professor Glenn Tesler for much help and advice during the last five years. Glenn has always been generous in giving me help and guidance every time I need him. I have learned a lot from him, not only his technical expertise, but also in scientific writing. I would also like to thank the rest of my thesis committee: Professor Vineet Bafna, Professor Ron Graham, Professor Mohan Paturi for their thoughtful comments and guidance throughout this journey.

During my graduate studies at UCSD, I have had the fortune to collaborate with many smart and wonderful people: Max Alekseyev, Paul Medvedev, Dmitry Antipov, Nikolay Vyahhi, Shay Zakov, Irina Vasilinetc, Nitin Udpa, Roy Ronen, Boyko Kakaradov, Nitin Gupta. I always remember many nights walking with Paul Medvedev around La Jolla (San Diego) thinking about paired de Bruijn graphs; a long working night submitting a RECOMB paper at 4 pm before going to Ralphs to buy a cake celebrating Paul's birthday; a night working on rectangle graphs with Ira and being forced to get out of the AU building at midnight; and many more beautiful memories with colleagues that I prefer to keep for myself.

Finally, I choose NOT to mention my parents, my sisters, my wife, and my daughter in this thesis. Describing what they mean to me by words is impossible, let alone acknowledging them.

Chapter 2, in full, is a reprint of the material as it appears in Paul Medvedev*, Son Pham*, Mark Chaisson, Glenn Tesler and Pavel Pevzner, "Paired de Bruijn graphs: a novel approach for incorporating mate pair information into genome assemblers". RECOMB 2011, pp. 238-251. (Co-first author). The dissertation author was the primary investigator and author of this paper.

Chapter 3, in part, is a reprint of the following: Nikolay Vyahhi, Alex Pyshkin, Son Pham and Pavel Pevzner, "From de Bruijn Graphs to Rectangle Graphs for Genome Assembly". WABI 2012, pp.

249-261. (Corresponding author).

Nikolay Vyahhi, Irina Vasilinetc, Son Pham and Pavel Pevzner, "From de Bruijn Graphs to Rectangle Graphs for Genome Assembly". WABI 2012, pp. 249-261. (Corresponding author). The dissertation author was the corresponding author of these papers.

Chapter 4, in part, is a reprint of the material as it appears in Son Pham*, Dmitry Antipov*, Alexander Sirotkin, Glenn Tesler, Pavel Pevzner, and Max Alekseyev, "Pathset Graphs: A Novel Approach for Comprehensive Utilization of Mate-Pairs in Genome Assembly". RECOMB 2012, pp. 200-212. (Co-first author). The dissertation author was the primary investigator and author of this paper.

Chapter 5, in part, is a reprint of the material as it appears in Son Pham and Pavel Pevzner, DRIMM-Synteny, "Decomposing Genomes into Evolutionary Conserved Segments". Bioinformatics 2010, pp. 2509-2516. The dissertation author was the primary investigator and author of this paper.

## VITA

| | |
|---|---|
| 2007 | B. S. in Applied Mathematics, Saint Petersburg State University, Russia. |
| 2010 | M. S. in Computer Science, University of California San Diego, San Diego, California, United States. |
| 2008–2013 | Graduate Student Researcher, University of California, San Diego, California, United States. |
| 2013 | Ph. D. in Computer Science, University of California, San Diego, California, United States. |

## PUBLICATIONS

Ilya Minkin, Anand Patel, Mikhail Kolmogorov, Nikolay Vyahhi, and Son Pham, "Sibelia: A scalable and comprehensive synteny block generation tool for closely related microbial genomes". Accepted to WABI 2013. (Corresponding author).

Viraj Deshpande, Eric Fung, Son Pham and Vineet Bafna, "Cerulean: A hybrid assembly using high throughput short and long reads" Accepted to WABI 2013.

Nikolay Vyahhi, Alex Pyshkin, Son Pham and Pavel Pevzner, "From de Bruijn Graphs to Rectangle Graphs for Genome Assembly". WABI 2012, pp. 249-261. (Corresponding author).

Son Pham and Paul Medvedev, " Mate-pair Consistency and Generating Problems". Accepted to RECOMB AB 2012.

Son Pham*, Dmitry Antipov*, Alexander Sirotkin, Glenn Tesler, Pavel Pevzner, and Max Alekseyev, "Pathset Graphs: A Novel Approach for Comprehensive Utilization of Mate-Pairs in Genome Assembly". RECOMB 2012, pp. 200-212. (Co-first author).

Paul Medvedev*, Son Pham*, Mark Chaisson, Glenn Tesler and Pavel Pevzner, "Paired de Bruijn graphs: a novel approach for incorporating mate pair information into genome assemblers". RECOMB 2011, pp. 238-251. (Co-first author).

Konstantin Avrachenkov, Vladimir Dobrynin, Danil Nemirovsky, Son Pham and Elena Smirnova, "PageRank Based Clustering of Hypertext Document Collections". Proceedings of ACM SIGIR 2008, pp. 873-874.

Konstantin Avrachenkov, Nelly Litvak, and Son Pham, "Distribution of PageRank Mass Among Principle Components of the Web". Proceedings of the 5th Workshop on Algorithms and Models for the Web-Graph, WAW2007, pp. 16-28.

Konstantin Avrachenkov, Danil Nemirovsky and Son Pham, "A survey on distributed approaches to graph based reputation measures". Proceedings of International Workshop on Tools for Solving Structured Markov Chains: SMCtools 2007, p. 82.

Vladimir Dobrynin, Son Pham, David Patterson, Niall Rooney, Mykola Galushka, "SOPHIA in Enterprise Track".The Fifteenth Text REtrieval Conference Proceedings 2006 (TREC2006).

**Journal Papers**

Son Pham*, Dmitry Antipov*, Alexander Sirotkin, Glenn Tesler, Pavel Pevzner, and Max Alekseyev, "Pathset Graphs: A Novel Approach for Comprehensive Utilization of Mate-Pairs in Genome Assembly". Journal of Computational Biology 2012, pp. 359-371. (Co-first author).

Anton Bankevich, Sergey Nurk, Dmitry Antipov, Alexey Gurevich, Mikhail Dvorkin, Alexander Kulikov, Valery Lesin, Sergey Nikolenko, Son Pham, Andrey Prjibelski, Alexey Pyshkin, Alexander Sirotkin, Nikolay Vyahhi, Glenn Tesler, Max Alekseyev, and Pavel Pevzner, "SPAdes: a New Genome Assembly Algorithm and its Applications to Single-Cell Sequencing". Journal of Computational Biology, 2012, pp. 455-477.

Paul Medvedev*, Son Pham*, Mark Chaisson, Glenn Tesler and Pavel Pevzner, "Paired de Bruijn graphs: a novel approach for incorporating mate pair information into genome assemblers". Journal of Computational Biology 2011, pp. 1625-1634. (Co-first author).

Son Pham and Pavel Pevzner, DRIMM-Synteny, “Decomposing Genomes into Evolutionary Conserved Segments”. Bioinformatics 2010, pp. 2509-2516.
Konstantin Avrachenkov, Nelly Litvak, Son Pham, “A Singular Pertubation Approach for Choosing the PageRank Damping Factor”. Internet Mathematics 2009, pp. 47-69.
Konstantin Avrachenkov, Danil Nemirovsky, Son Pham, Roberto Cascella, Roberto Battiti, Mauro Brunato, “Eigenvector based reputation measures”. Paradigms for Biologically–Inspired Autonomic Networks and Services 2008, pp. 260-278.

ABSTRACT OF THE DISSERTATION

**Genome Assembly and Comparison**

by

Kim Son Pham

Doctor of Philosophy in Computer Science

University of California, San Diego, 2013

Professor Pavel Pevzner, Chair

The recent proliferation of next generation sequencing with short reads has enabled many new experimental opportunities but, at the same time, has raised formidable computational challenges in genome assembly. One of the key advances that has led to an improvement in contig lengths has been read-pairs, which facilitate the assembly of repeating regions. The shortcomings of current read-pairs algorithm stem from the fact that they are heuristics approaches that are applied after the de Bruijn graphs have been constructed. First, we introduce the paired de Bruijn graph, a generalization of the de Bruijn graph that incorporates mate pair information into the graph structure itself instead of analyzing mate pairs at a post-processing step. While Paired de Bruijn Graphs provide an elegant solution to the read-pair analysis in theory, they are impractical in real sequencing data. Next, we introduce rectangles graphs and pathset graphs, which addressed additional challenges encountered in real data. In the final chapter of the thesis, we introduce an A-Bruijn graph algorithm for finding synteny blocks in highly duplicated genomes.

# Chapter 1

# Introduction

Current High Throughput Sequencing (HTS) technologies have reduced the time and costs of genome sequencing and have enabled new experimental opportunities in a variety of applications. At the same time, the short read length and sheer demand for powerful assemblers has raised formidable computational challenges. Thus, genome assembly continues to represent one of the most difficult and important algorithmic problems in bioinformatics.

Fortunately, all current sequencing platforms are able to produce mate-pairs — pairs of reads whose separation in the genome (called the insert size) is approximately known. Because insert sizes may be much longer than the read length, mate-pairs may span over long repeats and match up their flanking regions.

Mate-pair information has played an important role in most genome projects and many HTS assemblers incorporate mate-pair information to increase the contigs length in a post-processing step [67, 25, 95, 19, 83, 24, 53]. Most of these assemblers rely on the same basic observation [67]: if there is a unique path of suitable length in the assembly graph that connects the left and right reads of a mate-pair, the gap in the mate-pair can be filled in with the nucleotides spelled by this path (*mate-pair transformation*). However, when multiple paths exist between the left and right reads within a mate-pair, it remains ambiguous which path should be used to fill in the gap. Unfortunately, mate-pairs generated by existing HTS protocols are characterized by rather large variations in insert sizes, leading to multiple paths for a significant fraction of mate-pairs (since the range of suitable path lengths is wide), making it difficult to utilize such mate-pairs.

The challenge of algorithmically incorporating mate pair information into de Bruijn graph assemblers was first addressed by [71], which proposed a heuristic to look for a path between the two reads of a mate pair with a length of the insert size. If exactly one such path was found, then a *mate*

*pair transformation* could be applied to "unwind" this path in the graph. Essentially, this amounted to transforming two mated reads into one long read where the gap between the mates was filled in with the nucleotide sequence representing the found path, thus potentially connecting the surrounding regions of a repeat. Several other heuristic approaches for utilizing mate pair information in the de Bruijn graph were developed [94, 20, 58].

Such methods had a great impact on genome assembly, allowing the construction of much longer contigs; however, they could still fail in complex repeat-rich regions, where there are multiple paths between the read pairs. Many current technologies, including Complete Genomics [36] and Helicos [41], still generate very short reads (around 25 nt) for which the resulting de Bruijn graph is very tangled (even for bacterial genomes). In such cases, mate pair transformations often fail because of multiple paths. Additionally, the percentage of mate pairs that can be successfully transformed deteriorates when the insert size is high [22], and the search for paths between mates becomes prohibitively time-consuming. Unfortunately, these difficulties result in shorter contigs in complex repeat-rich regions. The limitations of the existing heuristics for analyzing mate pairs is thus a major hurdle towards assembling large contigs with short reads.

Three chapters (2, 3, 4) in this thesis address the challenges of utilizing mate-pair reads for improvements in genome assembly.

When genomic sequences are available, an important question to ask is how these genomes have evolved. The evidence in favor of the Whole Genome Duplication (WGD) in *S. cerevisiae* was discovered by [92] but was heavily contested (e.g., see [52]). [50] and [34] used preduplicated genomes of *K. waltii* and *A. gossypii* to settle this controversy (see [56] for a recent study contesting the WGD in yeast). Starting from [92] and [80] all WGD studies essentially amounted to constructing synteny blocks of certain type (e.g., *sister blocks* in [80] or *doubly conserved synteny (DCS)* blocks in [50]) and demonstrating that these blocks cover a large portion of the genome. Remarkably, there is still no general purpose tool that would automate such analysis and reduce various WGD studies to simply computing a "duplicativity coverage" of the genome. For example, software from [50] is not applicable to finding the synteny blocks from [80] and vice versa. Indeed, most WGD studies [87, 50, 34, 45, 55, 29, 5, 78] developed new software for WGD analysis instead of using some previously developed tools!

We argue that the lack of tools for automated WGD analysis is the result of the lack of tools for *synteny block* identification in highly duplicated genomes. Many genomes have undergone extensive duplications followed by gene losses and rearrangements, making decoding of genomic architecture (syn-

teny block reconstruction) in such genomes difficult. For example, duplications account for $\approx 70\%$ of the *Arabidopsis thaliana* genome [13] making synteny block reconstruction in this and other plant genomes challenging. Fig. 5.1(a) shows a highly duplicated "genome" $G$ along with its decomposition into *overlapping* (left) and *non-overlapping* (right) synteny blocks. The non-overlapping decompositions are more desirable since they are required for the follow-up rearrangement and duplication studies (e.g., the existing genome rearrangement algorithms are unable to analyze overlapping decompositions). Chapter 5 presents the DRIMM-Synteny algorithm for finding synteny blocks in highly duplicated genomes.

Chapter 2, in full, is a reprint of the material as it appears in Paul Medvedev*, Son Pham*, Mark Chaisson, Glenn Tesler and Pavel Pevzner, "Paired de Bruijn graphs: a novel approach for incorporating mate pair information into genome assemblers". RECOMB 2011, pp. 238-251. (Co-first author). The dissertation author was the primary investigator and author of this paper.

Chapter 3, in part, is a reprint of the following: Nikolay Vyahhi, Alex Pyshkin, Son Pham and Pavel Pevzner, "From de Bruijn Graphs to Rectangle Graphs for Genome Assembly". WABI 2012, pp. 249-261. (Corresponding author).
Nikolay Vyahhi, Irina Vasilinetc, Son Pham and Pavel Pevzner, "From de Bruijn Graphs to Rectangle Graphs for Genome Assembly". WABI 2012, pp. 249-261. (Corresponding author). The dissertation author was the corresponding author of these papers.

Chapter 4, in part, is a reprint of the material as it appears in Son Pham*, Dmitry Antipov*, Alexander Sirotkin, Glenn Tesler, Pavel Pevzner, and Max Alekseyev, "Pathset Graphs: A Novel Approach for Comprehensive Utilization of Mate-Pairs in Genome Assembly". RECOMB 2012, pp. 200-212. (Co-first author). The dissertation author was the primary investigator and author of this paper.

Chapter 5, in part, is a reprint of the material as it appears in Son Pham and Pavel Pevzner, DRIMM-Synteny, "Decomposing Genomes into Evolutionary Conserved Segments". Bioinformatics 2010, pp. 2509-2516. The dissertation author was the primary investigator and author of this paper.

# Chapter 2

# Paired de Bruijn graphs: a novel approach for incorporating mate pair information into genome assemblers

## 2.1 Introduction

The recent proliferation of next generation sequencing with short reads has enabled new experimental opportunities, such as the 10K genomes project, which aims to sequence and assemble the genomes of approximately one species in every vertebrate genus [42]. At the same time, the short read length and sheer demand for powerful assemblers has raised formidable computational challenges. Thus, genome assembly continues to represent one of the most difficult and important algorithmic problems in bioinformatics.

The first generation of assemblers followed the overlap-layout-consensus paradigm, where overlaps were heuristically used to join reads together into contigs [62, 11]. Later, the introduction of de Bruijn graphs led to significant improvements in assembly [70, 44, 73]. In contrast to the overlap-layout-consensus approach, these assemblers first constructed a graph where the original genome is spelled by a series of walks through the graph, and non-branching walks correspond to substrings (contigs) of the genome. Compared to the earlier heuristic approaches, de Bruijn graphs produced longer contigs and gave rise to more powerful techniques for correcting errors and resolving repeats — identical, or nearly identical, stretches of DNA [79]. Their success led to the development of other types of graphs for sequence assembly: *A*-Bruijn graphs [72] and closely related string graphs [61], which together have become an essential part of most modern assembly tools, including EULER-SR [23], Velvet [94], ALLPATHS [20], ABySS [82], and others.

Despite these advances, the challenge of resolving repeats remains. When the length of a repeat

is longer than twice the read length, it becomes difficult to correctly match its upstream and downstream regions. In order to alleviate this problem, sequencing technologies were extended to produce *mate pairs* [90] — pairs of reads between which the genomic distance (called the *insert size*) is well estimated. Because insert sizes could be much longer than the read length, mate pairs were able to span long repeats and could potentially match up the regions surrounding a repeat.

The challenge of algorithmically incorporating mate pair information into de Bruijn graph assemblers was first addressed by [71], who proposed a heuristic to look for a path between the two reads of a mate pair with a length of the insert size. If exactly one such path was found, then a *mate pair transformation* could be applied to "unwind" this path in the graph. Essentially, this amounted to transforming two mated reads into one long read where the gap between the mates was filled in with the nucleotide sequence representing the found path, thus potentially connecting the surrounding regions of a repeat. Several other heuristic approaches for utilizing mate pair information in the de Bruijn graph were developed [94, 20, 58].

Such methods had a great impact on genome assembly, allowing the construction of much longer contigs; however, they could still fail in complex repeat-rich regions, where there are multiple paths between the read pairs. Many current technologies, including Complete Genomics [36] and Helicos [41], still generate very short reads (around 25 nt) for which the resulting de Bruijn graph is very tangled (even for bacterial genomes). In such cases, mate pair transformations often fail because of multiple paths. Additionally, the percentage of mate pairs that can be successfully transformed deteriorates when the insert size is high [22], and the search for paths between mates becomes prohibitively time-consuming. Unfortunately, these difficulties result in shorter contigs in complex repeat-rich regions. The limitations of the existing heuristics for analyzing mate pairs is thus a major hurdle towards assembling large contigs with short reads.

We believe that the shortcomings of current mate pair algorithms stem from the fact that they are heuristic approaches that are applied *after* the construction of the de Bruijn graph. The de Bruijn graph does an excellent job of incorporating the sequence information from the single reads; however, it ignores any mate pair information that is available. This information has to be recovered after the graph construction, and only then applied in a heuristic manner. In this paper, we propose the *paired de Bruijn graph*, a generalization of the de Bruijn graph that incorporates the mate pair information into the structure of the graph itself, as opposed to a post-processing step. Just as moving from the heuristic overlap-layout-consensus paradigm to the de Bruijn graph paradigm resulted in better assemblies, we believe that moving

from heuristic mate pair algorithms to paired de Bruijn graphs could result in a more effective use of mate pair information. The paired de Bruijn graph is a potential replacement of the de Bruijn graph in existing de Bruijn graph based assemblers; existing assembly stages, including error correction and scaffolding, would not need to be substantially modified.

Through assembly results on simulated perfect data, we argue that when mate pair information is used in this manner, the read length (once above a small threshold) becomes much less relevant [22]. We find that the contig sizes in an assembly are largely dictated by the average insert size — when it exceeds 6000 nt, we can assemble all of *E. coli* into one contig and most of the human chromosome 22 into 15 contigs. Though this paper falls short of analyzing real data, we believe that, similar to how early error-free studies of de Bruijn graphs laid the foundation for their use in assembly [70], the paired de Bruijn graph can become the basis of practical assemblers.

## 2.2 From de Bruijn Graphs to Paired de Bruijn Graphs

### 2.2.1 Preliminaries

To simplify the presentation, we assume that the genome is a circular string, i.e., one circular, single-stranded chromosome, and that all reads have the same length $\ell$; extending our approach for multiple linear chromosomes or varying read length is straightforward. Moreover, we assume that reads are error-free (see Section 2.5 for a discussion). In this setting, a mate pair is an ordered pair of strings of length $\ell$ drawn from the genome at positions $i$ and $j$, respectively. Normally, the relative distance between reads is expressed in terms of the *insert size*, the number of nucleotides from the first nucleotide of $a$ to the last nucleotide of $b$: $j-i+\ell$. However, for the purposes of our construction, it is more convenient to express it in terms of $d=j-i$, the difference in their leftmost coordinates. Note that $d$ is the insert size minus one read length (see Fig. 2.1(a)).

As with any de Bruijn graph based approach, our algorithms have a parameter $k$ that dictates the size of the substrings into which the reads are chopped up. Thus, though our input is a set of mate pairs of reads of any length, we immediately chop them up into smaller pieces. Formally, each mate pair of reads is replaced by its constituent $\ell-k$ (sub-)mate pairs, where the reads of each (sub-)mate pair now have length $k+1$. Therefore, for the remainder of the paper, we will assume without loss of generality that the reads are immediately given with length $k+1$. We now give some definitions.

**A-Bruijn graphs:** Let $G$ be a directed graph on $m$ vertices. The *gluing* of vertices $v$ and $w$ is defined by substituting $v$ and $w$ by a single vertex (called the *successor* of $v$ and $w$) and retaining all

edges incident to either $v$ or $w$ as edges incident to their successor. Let $A$ be a boolean $m \times m$ matrix representing "glues" [72]. The *A-Bruijn graph* $A(G)$ is obtained by gluing all vertices $v$ and $w$ of $G$ for which $A_{v,w} = 1$. One can execute these glues in an arbitrary order under the assumption that each gluing instruction $A_{v,w} = 1$ is applied to the successors of vertices $v$ and $w$ in the graph resulting from the previous gluing instructions.

Below we describe three $A$-Bruijn graphs: de Bruijn graphs (for unpaired reads); paired de Bruijn graphs (for mate pairs with an exact distance), and approximate paired de Bruijn graphs (for mate pairs with an approximate distance).

**$k$-mers and labels:** Define a *$k$-mer* as a string of length $k$. Below we assume that the parameter $k$ is fixed. Given a circular string $S = s_1 \dots s_n$, let $S_k(i)$ be the $k$-mer $s_i \dots s_{i+k-1}$ (where the index is taken modulo $n$). The set of all $k$-mers $S_k(i)$ (for $1 \leq i \leq n$) is called the *$k$-spectrum* of $S$. For a $k$-mer $a = a_1 \dots a_k$, we define two $(k-1)$-mers, $\text{prefix}(a) = a_1 \dots a_{k-1}$ (remove last character) and $\text{suffix}(a) = a_2 \dots a_k$ (remove first character). We say that $k$-mer *$a$ aligns at position $i$* if $a = S_k(i)$.

**$(k,d)$-mers and bilabels:** A *bilabel* $(a|b)$ is a pair of strings, $a$ and $b$, of equal length. Define $\text{left}(a|b) = a$ and $\text{right}(a|b) = b$. A *$k$-mer bilabel* indicates both $a$ and $b$ have length $k$. Define $\text{prefix}(a|b) = (a_1 \dots a_{k-1}|b_1 \dots b_{k-1})$ and $\text{suffix}(a|b) = (a_2 \dots a_k|b_2 \dots b_k)$. Given an integer $d$ (usually $d \geq k$), a $(k,d)$-mer of $S$ is a pair of $k$-mers $S_k(i)$ and $S_k(i+d)$ that start exactly $d$ nucleotides apart. We use the bilabel notation $(S_k(i)|S_k(i+d))$ for $(k,d)$-mers. For a string $S$ and parameters $d$ and $\Delta$, we say $k$-mer bilabel $(a|b)$ *aligns at position $i$* if $a = S_k(i)$ and $b = S_k(i+d+x)$ for some $-\Delta \leq x \leq \Delta$. A *$(k,d,\Delta)$-mer* of $S$ is a bilabel $(a|b)$ that aligns somewhere to $S$.

### 2.2.2 De Bruijn Graphs (Modelling Unpaired Reads)

Let $C$ be a set of $(k+1)$-mers from a circular string $S$. We construct an $A$-Bruijn graph based on $C$ as follows.

- First we define an initial graph $G_0$ consisting of $m = 2|C|$ vertices and $|C|$ isolated edges. For each $(k+1)$-mer $a \in C$, introduce two new vertices $u, v$ and form an edge $u \to v$. Label the edge by the $(k+1)$-mer $a$; label $u$ by the $k$-mer $\text{prefix}(a)$; and label $v$ by the $k$-mer $\text{suffix}(a)$.

- Second, we glue certain vertices of $G_0$ together, by forming an $m \times m$ binary matrix $A$ and setting $A_{i,j} = 1$ to indicate that vertices $i$ and $j$ should be glued together. For this construction, we set $A_{i,j} = 1$ when vertices $i$ and $j$ have the same label.

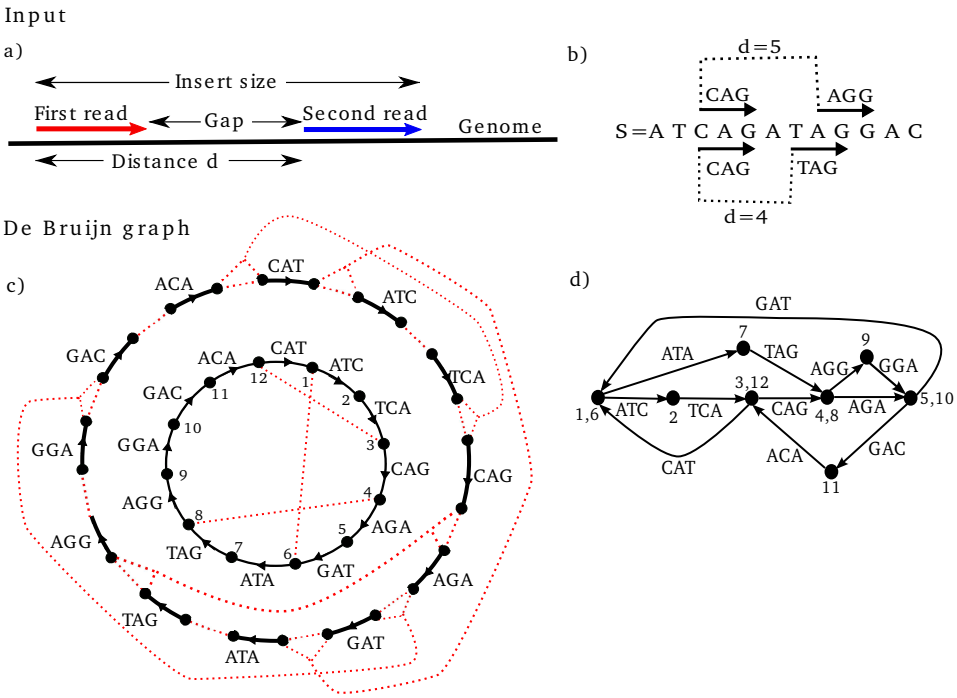
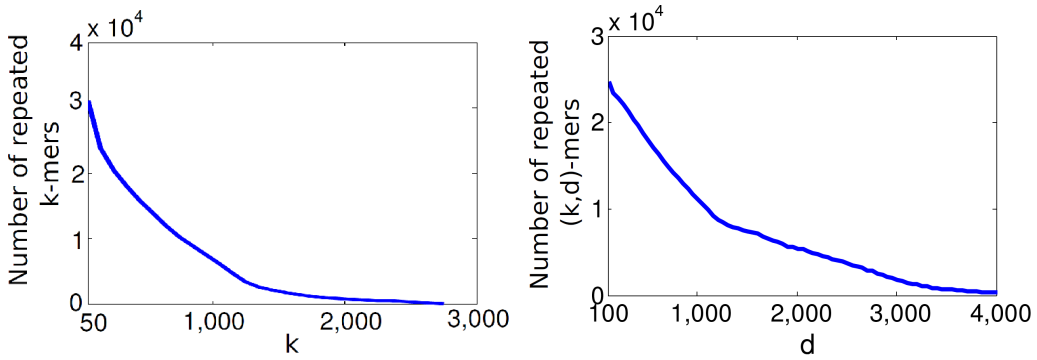



Figure 2.1: Mate pairs and the de Bruijn graph: (a) A mate pair is a pair of reads with a distance of $d$ between their start positions. (b) A circular genome $S$ and two mate pairs, with $d = 4$ and $d = 5$. (c–d) The de Bruijn graph construction for $k = 2$. In (c), the outside circle shows a separate black edge for each 3-mer (equivalently, each element of the 3-spectrum). The dotted red lines indicate vertices that will be glued. The inner circle shows the result of applying some of the glues. Note that this is an intermediate step of the construction in which we only show the gluings of vertices arising from the same position of $S$. (d) The final de Bruijn graph, resulting from all the glues.

The labeled directed graph $G = \mathrm{DB}(C,k)$ obtained from these gluings is the de Bruijn graph of $C$ [72] (for an illustration, see Fig. 2.1(b,c,d)). It may be considered as either a simple graph (without parallel edges but with loops), or as a multigraph where the multiplicity of each edge is determined by the number of times the $(k+1)$-mer it represents is present in $C$. Consider a walk through edge/label sequence $e_1,\ldots,e_r$. The labels satisfy $\mathrm{suffix}(e_i) = \mathrm{prefix}(e_{i+1})$, and we may define the string of length $r+k$ spelled by this walk as $\mathrm{walkword}(e_1,\ldots,e_r)$ by successively overlapping the labels with a shift of one character at a time.

Traditionally, the de Bruijn graph is also defined on a string $S$ by setting the vertex set equal to the $k$-spectrum of $S$. For every $(k+1)$-mer $a$ of $S$, define an edge $\mathrm{prefix}(a) \to \mathrm{suffix}(a)$ labeled by $a$. Explicitly, for each $S_{k+1}(i)$ of $S$, define an edge $S_k(i) \to S_k(i+1)$ labeled by $S_{k+1}(i)$ (for $i = 1,\ldots,n$).

In the case that $C$ is the $(k+1)$-spectrum of $S$, the de Bruijn graph built on $C$ using the gluing approach is identical to the one built directly on the genome $S$. Moreover, there is a covering cycle that spells $S$, where a *covering cycle* is a cyclical walk that visits every edge at least once. In this graph, the




(a) (b)

Figure 2.2: The effect of increasing $k$ and $d$: (a) The number of repeated $k$-mers in the *E. coli* genome, for various values of $k$. (b) The number of repeated $(k,d)$-mers, for various values of $d$ with $k = 50$.

cycle is the sequence of edges $S_{k+1}(1),\ldots,S_{k+1}(n)$. The covering cycle property is crucial for assembly because it implies that all walks whose interior vertices have just one out-neighbor must spell substrings in $S$ (contigs).

### 2.2.3 Graph Complexity

The usefulness of a graph representation of a genome can vary widely. In general, the number of vertices can serve as a rough indicator of how useful the graph is — as the number of vertices grows (and the number of edges stays the same), the graph is likely to become less entangled, and the contigs are likely to become longer. Fig. 2.2(a) shows that in the de Bruijn graph, the number of repeated $k$-mers in *E. coli* drops as $k$ increases, indicating that the de Bruijn graph has more vertices and likely becomes less entangled. Alternatively, consider pairs of $k$-mers, i.e., $(k,d)$-mers. Fig. 2.2(b) shows that, after fixing $k = 50$, the number of repeated $(k,d)$-mers drops as $d$ increases. This is not surprising due to the repeat structure of genomes — the bigger the $d$, the less common it is to have pairs of repeats spaced a distance of $d$ apart. Figs. 2.2(a) and (b) illustrate alternatives for improving contig lengths: increasing the read length (pursued by companies such as Pacific Biosciences) versus increasing the insert size (advocated by Chaisson et al. [22]). While the increase in the read length remains a difficult technological challenge, increasing the insert size (up to tens of thousands of nucleotides) is already within the power of current technologies. Thus, if we could build a graph whose vertices represent $(k,d)$-mers instead of $k$-mers, then the length of the contigs is likely to increase as the insert size grows. This is the basic motivation for the paired de Bruijn graph, and, as we will show in Section 2.3, the contig lengths in the paired de Bruijn graph do in fact increase with $d$.

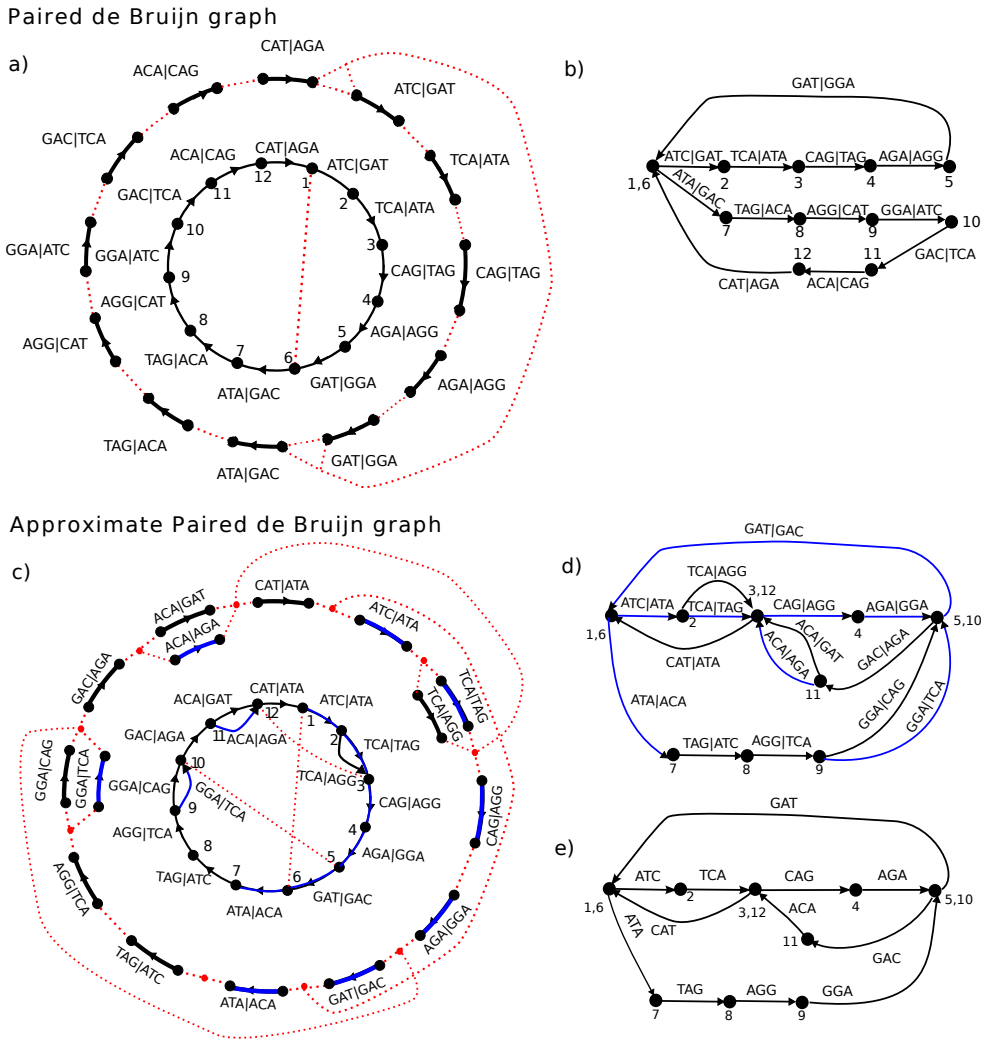



Figure 2.3: The (approximate) paired de Bruijn graph: (a–b) The paired de Bruijn construction for $k = 2$, $d = 4$ from the same string $S$ as in Fig. 2.1. In (a), the outer circle has an edge from every element of the $(3,4)$-spectrum. (b) The paired de Bruijn graph after all the gluings; notice that it has only one branching vertex, versus four in the de Bruijn graph (Fig. 2.1(d)). (c–e) The construction of the approximate paired de Bruijn graph for $k = 2$, $d = 5$, $\Delta = 1$. In (c), one possible covering spectrum is shown in the outside circle, with black edges for elements with mate pair distance 6 and blue edges for distance 5. Since $\Delta = 1$, we glue vertices if they have equal left labels and their right labels are a distance at most 2 apart from each other in the de Bruijn graph (Fig. 2.1(d)). The final multigraph after all vertex gluings is shown in (d), and the resulting simple graph, used to spell the contigs, is shown in (e). Notice that this graph now has three branching vertices.

### 2.2.4 Paired de Bruijn Graphs (Modelling Paired Reads with Exact Distance)

We now define a graph modelling mate pairs in the special case that all pairs are exactly the same distance $d$ apart. This is an idealized case unachievable with current sequencing technologies, but the next section will generalize the construction to varying distances. Given a set of $(k+1,d)$-mers $C$ (modelling mate pairs), construct an A-Bruijn graph as follows:

- Define an initial graph $G_0$ on $m = 2|C|$ vertices. For each bilabel $(a|b) \in C$ (representing a $(k+1,d)$-mer), introduce two new vertices $u, v$ and form an edge $u \to v$. Label the edge by $(a|b)$; label $u$ by prefix$(a|b)$; and label $v$ by suffix$(a|b)$.
- Glue vertices of $G_0$ together when they have the same label. The graph $G$ so obtained is called the *paired de Bruijn graph* of $C$.

This procedure is illustrated in Fig. 2.3(a,b), and Fig. 2.4 gives another example of the graph. An alternate construction of the paired de Bruijn graph is to define the vertex set as the $(k,d)$-mers present in $C$, and the edges as connecting prefix$(a|b)$ to suffix$(a|b)$ for every element of $C$.

As with the regular de Bruijn graph, in this construction, every vertex of $G$ inherits the label common to all the vertices of $G_0$ that were glued together to form it, and this label is unique to that vertex. Any walk through the graph on edge sequence $e_1, \ldots, e_r$ spells out an $(r+k)$-mer bilabel $(L|R)$ where $L$ is formed from the left labels, $L = \text{walkword}(\text{left}(e_1), \ldots, \text{left}(e_r))$, and $R$ is formed from the right labels, $R = \text{walkword}(\text{right}(e_1), \ldots, \text{right}(e_r))$.

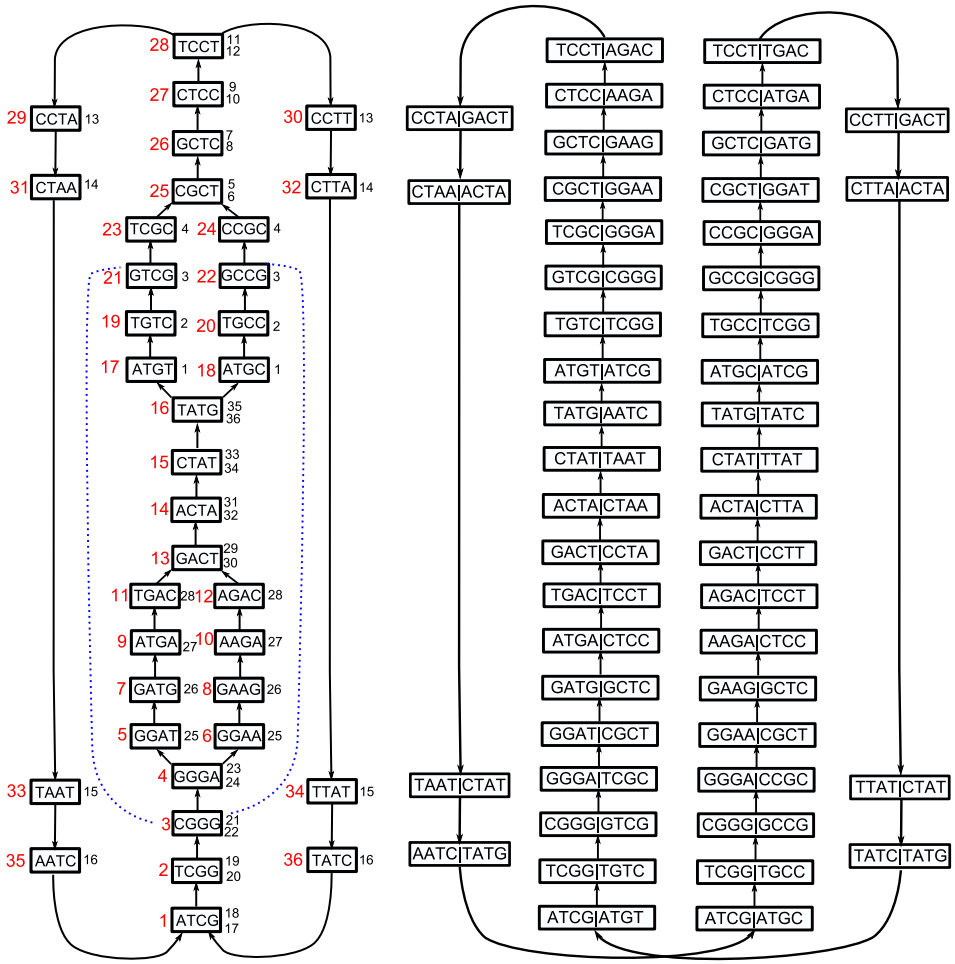
The $(k,d)$-spectrum of a string $S$ is $\{(S_k(i)|S_k(i+d)) : i = 1, \ldots, n\}$. When $C$ is the $(k+1,d)$-mer spectrum of $S$, there is a covering cycle whose left labels spell $S$ in $G$. The cycle consists of consecutive edges

$$(S_k(i)|S_k(i+d)) \to (S_k(i+1)|S_k(i+d+1)) \quad \text{for } i = 1, \ldots, n.$$

Just as with the de Bruijn graph, this is a key property that makes the paired graph useful for spelling contigs.

### 2.2.5 Approximate Paired de Bruijn Graphs (Modelling Inexact Distance)

We now define a graph modelling mate pairs where the distance between the two reads in each pair is only known to lie within some range $d \pm \Delta$. The parameter $\Delta$ can be estimated based on the mate pair generation protocol.




(a) De Bruijn graph (b) Paired de Bruijn graph

Figure 2.4: Example of the standard and paired de Bruijn graphs: The reads are the (5,12)-spectrum generated from the cyclic sequence $ATCGGGATGACTATGTCGCTCCTAATCGGGAAGACTATGCCGCTCCTT$. (a) The de Bruijn graph with edges constructed from the set of 5-mers in the (5,12) spectrum. Each node is a rectangle labeled by a 4-mer with the node ID shown as a large red number on the left of the node. The mate pair information is also presented in the graph: for each node, the node IDs of its corresponding right 4-mers are shown as small numbers on the right of the rectangle. For instance, the right 4-mers (blue dotted lines) of CGGG (node 3) are GTCG (node 21) and GCCG (node 22) and we write 21 and 22 on the right side of node 3. Note that there is not a single mate pair with a unique path between the mates, making mate pair transformations impossible. (b) The paired de Bruijn graph from the (5,12) spectrum is a cycle, representing a single contig. In this example, the paired approach allows for longer contigs than would mate pair transformations (though there are also examples when the opposite is true).

Let $C$ be an arbitrary set of $(k+1,d,\Delta)$-mers, representing the input data. The key insight is that if two $(k,d,\Delta)$-mers $(a|b)$ and $(a|b')$ both arise from the same instance of $a$ in $S$, then in the de Bruijn graph of $S$, there is a directed path from $b$ to $b'$, or vice-versa, with distance at most $2\Delta$. This insight was used for repeat resolution in [58], albeit as a post-construction modification step. We construct an $A$-Bruijn graph from $C$ as follows:

- The initial graph $G_0$ consists of $|C|$ isolated edges on $2|C|$ vertices. For each $(a|b) \in C$, introduce an edge $u \to v$ on two new vertices. Label the edge by the $(k+1)$-mer bilabel $(a|b)$. Label $u$ by prefix$(a|b)$ and $v$ by suffix$(a|b)$.
- For each $k$-mer $\alpha$, glue together all vertices with labels $(\alpha|\beta),(\alpha|\beta')$ if there exists a directed path from $\beta$ to $\beta'$ (or vice-versa) in the de Bruijn graph $D = \mathrm{DB}(C,k)$ of length at most $2\Delta$. Here, we assume that the construction of $D$ implicitly breaks the $(k+1)$-mer bilabels of $C$ into independent $(k+1)$-mers.

The graph $G = \mathrm{APDB}(C,k,d,\Delta)$ so obtained is the *approximate paired de Bruijn Graph* of $C$ (Fig. 2.3(c,d,e)). The effect of this gluing is to merge all vertices ($k$-mer bilabels) that might align to the same position in the genome; vertices that align to the same position are thus guaranteed to be merged. However, the converse does not hold; vertices aligning to different positions in the genome are sometimes merged, either due to repeats that are not resolved by the given parameters, or due to chance short paths in $D$.

In the case that $k > 2\Delta$, we observed that if there is a directed path between $\beta$ and $\beta'$ in the de Bruijn graph $D$ of length at most $2\Delta$, then $\beta$ and $\beta'$ should share an overlap of at least $k-2\Delta$ characters. This observation leads to an alternate rule to glue vertices of $G_0$: for each $k$-mer $\alpha$, glue together all vertices with labels $(\alpha|\beta),(\alpha|\beta')$ if $\beta$ and $\beta'$ share an overlap of at least $k-2\Delta$ characters. Note that this rule can only be used if $k > 2\Delta$ and may lead to a different graph; however, it is easier to implement.

Unlike our earlier constructions of the de Bruijn and paired de Bruijn graphs, the vertices of $G$ do not inherit a single label from $G_0$; the vertices glued together have the same left label, but may have different right labels. In an edge walk $e_1,\ldots,e_r$ on $G$, the left labels spell walkword$(\mathrm{left}(e_1),\ldots,\mathrm{left}(e_r))$. However, the right labels typically do not successively overlap by $k-1$ characters as they did for the paired de Bruijn graph. Though we currently ignore these after gluing, we recognize that there is a potentially untapped benefit to using the right labels to later improve the assembly (see Section 2.5).

A set $C$ of $(k+1)$-mer bilabels is a *covering spectrum of* $S$ if for every position $i = 1,\ldots,n$, we

have $(S_{k+1}(i)|S_{k+1}(i+d+x)) \in C$ for at least one $x$ in the range $-\Delta \leq x \leq \Delta$. For each position $i$, there are $2\Delta+1$ choices of $x$. Note that there are many different covering spectra, and different choices of $C$ may lead to different graphs. However, the graph will satisfy the key property of having a covering cycle that spells out $S$.

**Theorem 2.1.** *Let $S$ be a circular string, and $C$ a set of $(k,d,\Delta)$-mers that is a covering spectrum of $S$. Then there is a covering cycle through the graph $G = \mathrm{APDB}(C,k,d,\Delta)$ that spells out $S$.*

*Proof.* For $i = 1,\ldots,n$, let $e_i \in C$ be any $(k+1)$-mer bilabel in $C$ aligning to position $i$ in $S$. To prove $e_1,\ldots,e_n$ is a cycle in $G$, we need to show that consecutive edges $e_i = u_i \rightarrow v_i$ with label $(a|b)$, and $e_{i+1} = u_{i+1} \rightarrow v_{i+1}$ with label $(a'|b')$, share the connecting vertex, $v_i = u_{i+1}$. (Indices are taken modulo $n$.) Since $C$ is a covering spectrum of $S$, the graph $D$ is the ordinary de Bruijn graph of $S$. In $G_0$, $v_i$ has label $(S_k(i+1), \mathrm{suffix}(b))$ and $u_i$ has label $(S_k(i+1), \mathrm{prefix}(b'))$. Since these both align to position $i+1$ in $S$, the distance between the start of $b$ and $b'$ in $S$ is at most $2\Delta$. Thus in $D$, the directed distance from $b$ to $b'$ (or vice-versa) is at most $2\Delta$, so these vertices were glued together when forming $G$. □

## 2.3 Results

We implemented a prototype assembly algorithm to test the effectiveness of the (approximate) paired de Bruijn graph approach under the ideal conditions of perfect coverage and error-free reads. We experimented with with *E. coli* (4.6 Mbp) and *Human* chromosome 22 (35 Mbp after removal of ambigious bases). The reads were generated with perfect coverage, meaning for every position in the genome we generated a single $(k,d,\Delta)$-mer aligning to it. The insert size was picked uniformly at random from the specified range. We report as contigs the (left) words spelled by all maximal walks of the graph whose interior vertices have just one out-neighbor. We validated that any generated contigs mapped perfectly back to the original genome — this was the case for all the contigs.

Constructing the de Bruijn graph and finding all its non-branching paths takes time $O(n \log n)$, where $n$ is the number of $k$-mers. The construction of the approximate paired de Bruijn graph has an additional cost of searching all neighbors within a distance $2\Delta$ of each node. Therefore, the running time of the algorithms is $O(n \log n + n \min\{2^{\Delta}, n\})$, where $n$ is the number of $(k,d,\Delta)$-mers. However, since de Bruijn graphs are sparse, the searches in the graph are usually very fast, and in practice, even the run on chr22 with $\Delta = 200$ took less than 2 hours on a 8 core processor with 16G RAM. Moreover, the algorithm could be easily distributed over a large cluster to deal with larger $\Delta$.

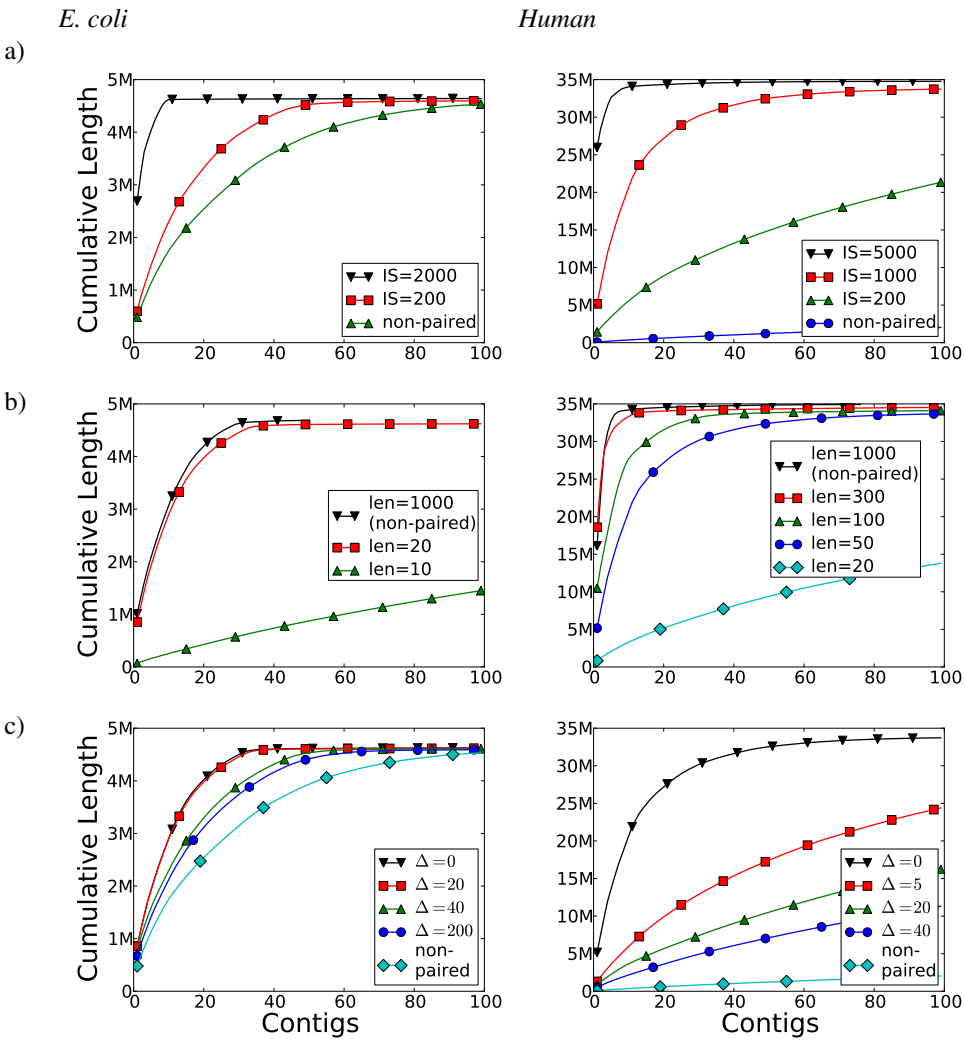



Figure 2.5: Contig Lengths: Cumulative contig lengths (for standard and paired de Bruijn graphs) on simulated data with perfect coverage. Contigs are sorted in order from largest to smallest. Point $(x, y)$ means the largest $x$ contigs have cumulative length $y$. (a) To analyze the effect of the insert size (IS) on the assembly, we kept the read length fixed at 50, but varied the insert size. We also generated non-paired reads of length 50. For *E. coli*, the curve for insert size 6000 is not shown because there was only one contig, representing the whole genome. (b) To analyze the effect of read length on contig lengths, we fixed the insert size to 1000 but varied the read length. We also generated non-paired reads of length 1000, giving an upper bound on how good the assembly can be in this case. (c) To analyze the effect of variations in the insert size ($\Delta$), we fixed the mean insert size (1000) and read length (50). We also show the baseline contig lengths in a non-paired dataset, with read length 50 and perfect coverage.

Our motivation for the paired de Bruijn graph approach was that the number of repeated $(k,d)$-mers quickly drops as $d$ increases (Fig. 2.2(b)), and hence the contigs of the paired de Bruijn graph based on these $(k,d)$-mers could be longer. To test this hypothesis, we generated a set of mate pairs with varying insert sizes and plotted the length of the obtained contigs (Fig. 2.5(a)). To isolate the effect of the insert size, the coverage of the data was perfect (the $(k,d)$-spectrum), the insert sizes were perfect ($\Delta = 0$), and the read length was fixed to 50. We observed that contig lengths improved dramatically as the insert size increased. With an insert size of 6000 nt, all of *E. coli* was covered with just one contig, while for chr22, an insert size of 5000 nt enabled us to cover 98% of the chromosome with the 15 largest contigs. We thus believe that properly using mate pairs has a strong potential to increase contig lengths.

To explore the role that read length plays relative to the insert size, we generated sets of mate pairs with varying read lengths but with a fixed insert size (1000 nt). To isolate the effect of the read length, we had perfect coverage and no variation in the insert size. For *E. coli*, we found that, for an insert size of 1000 nt, once the read length grew over a small threshold of 10–20 nt, the contig lengths nearly reached the theoretical optimum that could be achieved by simply generating reads of length equal to the insert size (Fig. 2.5(b)). For *Human*, we needed to increase the read length to 300 nt in order to reach the optimum with 1000 nt insert size (Fig. 2.5(b)). However, for a longer insert size (5000 nt), a read length of 50 came close (Fig. 2.5(a)) to achieving the optimum (which, with 5000 nt reads, was a single contig). Therefore, by properly using mate pairs with large enough insert size, one can significantly reduce the limitations caused by short read length.

We measure the effect of increasing variability in the insert size ($\Delta$) on the assembly. We fix the insert size to be 1000 nt and generate 50-long reads with perfect coverage, while varying $\Delta$ (Fig. 2.5(c)). We found that the assembly deteriorates with increasing $\Delta$, especially for the *Human* genome. When $\Delta$ is large, the chance of two vertices of the de Bruijn graph being connected increases, and, hence, the number of vertices (bilabels) that do not align but nevertheless get glued together increases. In this situation, the read length is still important in determining the complexity of the (non-paired) de Bruijn graph. Some recent datasets achieve a small $\Delta$, such as the [12] human dataset with a mean insert size of 208 nt and a standard deviation of 13 nt. Nevertheless, we see robustness with respect to $\Delta$ as an important direction for improving the practical usefulness of our method.

## 2.4 Towards a Practical Paired de Bruijn Graph Assembler

We believe that, similarly to early studies of idealized fragment assembly with error-free $k$-mers [70], the (approximate) paired de Bruijn graphs can be of use in practical assemblers that utilize paired reads. Though this paper falls short of analyzing real data, we present here potential ways to remove our simplifications, and to move from the current de Bruijn graph assemblers to (approximate) paired de Bruijn graphs.

**Base calling errors in reads.** As with regular assembly, reads with base-calling errors may perturb the graph. Error correction algorithms for single reads may be used to improve the accuracy of the reads, while future error correction algorithms may also incorporate the mate pair information. Graph correction algorithms employed by current de Bruijn based assemblers [23, 94] may also be applied to (approximate) paired de Bruijn graphs.

**Insert size outliers.** If a small percentage of read pairs are spaced outside the range $d \pm \Delta$, they will likely form isolated edges or terminal branches, which can be detected and discarded.

**Double strandedness.** The approximate paired de Bruijn graph is asymmetric in its treatment of the two reads $(a|b)$, and in the reverse complement, these are switched to $(b'|a')$ (where $a', b'$ are the reverse complements of $a$ and $b$). This makes existing methods [48, 59, 94] for accounting for double-strandedness difficult to apply. However, we may explicitly introduce the reverse complement of every read; perform assembly; match up reverse complement contigs after assembly; and reconcile any differences through a consensus stage.

## 2.5 Conclusion

In this paper, we introduced the paired de Bruijn graph and motivated its use in genome assembly. Instead of incorporating mate pairs into a post-graph-construction step, we have used them to construct the graph itself. Any procedures that could be performed on the regular de Bruijn graph (e.g., error correction) can be performed in the same manner on the paired de Bruijn graph. For instance, even when there are repeats that the paired de Bruijn graph does not resolve, mate pair transformations can still be applied to the graph to help resolve the remaining repeats.

By formulating an alternative to mate pair transformations, the paired de Bruijn graph approach provides a potential method for assembly with short read mate pairs, like the ones generated by Complete Genomics [36] and Helicos [41]. By not requiring unique paths between paired reads in the de Bruijn

graph, the paired approach could still resolve repeats despite the short read length (e.g. Fig 2.4). Moreover, the algorithms we describe can be extended to the *strobes* generated by Pacific Biosciences, which extend the notion of the mate pair to a set of multiple (more than two) reads separated by some distances.

A future direction lies in the use of the right labels on edges of the approximate paired de Bruijn graph. Currently, we spell out each contig using only the left label. The positions of the right labels are only known approximately, but this is often sufficient to form a righthand word displaced approximately $d$ from the lefthand word. Moreover, after encountering an edge $(a|b)$ in a walk, we must encounter some edge $(b|c)$ approximately $d$ edges away (unless it is past the end of the walk). This compatibility requirement may help to narrow the choice of valid paths when encountering branching vertices, thereby resolving longer repeats and improving contig lengths.

Chapter 2, in full, is a reprint of the material as it appears in Paul Medvedev*, Son Pham*, Mark Chaisson, Glenn Tesler and Pavel Pevzner, "Paired de Bruijn graphs: a novel approach for incorporating mate pair information into genome assemblers". Journal of Computational Biology 2011, pp. 1625-1634. (Co-first author). The dissertation author was the primary investigator and author of this paper.

# Chapter 3

# From de Bruijn Graphs to Rectangle Graphs for Genome Assembly

## 3.1 Introduction

The recent proliferation of next generation sequencing technologies has enabled new experimental opportunities and, at the same time, raised formidable computational challenges. When the length of a repeat in the genome exceeds the read length, it becomes difficult to "span" the flanking regions of this repeat in the assembly. To alleviate this problem, sequencing technologies were extended to produce *read-pairs*, pairs of reads separated by an estimated *insert length*. Because insert length is longer than the read length, read-pairs span longer repeats and could potentially result in better assemblies. However, while assembling *single* reads can be elegantly modeled by *de Bruijn graphs* [44], equally elegant models for assembling *read-pairs* remain unknown [32].

Pevzner and Tang, 2001 [67] addressed this challenge by constructing the de Bruijn graph and further checking if a path between two reads within a read-pair satisfies the constraint imposed by the insert length. If only one such path exists, the read-pair is transformed into a virtual long read where the gap between reads is filled in with the nucleotide sequence representing the found path.

While this and similar methods [95, 53] had a large impact on genome assembly, they fail in repeat-rich regions, where there are multiple paths between the reads within read-pairs. Recently, Medvedev et al., 2011 [57] introduced *paired de Bruijn graphs* that directly incorporate read-pairs into the graph structure and bypass the problem of multiple paths in previous approaches. However, the paired de Bruijn graph concept was introduced as a theoretical framework and is mainly aimed at an unrealistic case when the distance between reads within read-pairs is exactly $d$ for all read-pairs. To address this bottleneck, Bankevich et al., 2012 [9] introduced the *rectangle graph* by generalizing the problem of a string reconstruction from its paired substrings to a variation of a jigsaw puzzle problem. While fragment as-

sembly is usually modeled as a 1-dimensional *overlapping* puzzle (pieces correspond to individual reads), Bankevich et al. [9] modeled assembly as a 2-dimensional *non-overlapping* puzzle (pieces correspond to pairs of paths in the de Bruijn graph). However, while Bankevich et al. [9] sketched the rectangle graph idea, the various questions arising in applications of rectangle graphs to fragment assembly remained unaddressed.

The jigsaw puzzles were originally constructed by painting a picture on a rectangular piece of wood and further cutting it into smaller pieces with a jigsaw. The *Jigsaw Puzzle Problem* is to find an arrangement of these pieces that fills up the rectangle in such a way that neighboring pieces have "matching" boundaries with respect to color and texture. This paper extends the previous algorithmic studies of the Jigsaw Puzzle Problem (that were motivated by the restoration of archaeological artifacts [47]) to the problem of genome assembly from read-pairs. In section 2, we describe a class of simple jigsaw puzzles (called *rectangle puzzles*) and define the rectangle graph to assemble such puzzles. In section 3, we establish the relation between the rectangle puzzle and genome assembly from read-pairs and address the algorithmic challenges of "missing rectangles" (that often arise in genome assembly) in the rectangle puzzle problem. In Section 4, we apply the rectangle graph to bacterial genome assembly for both standard (multicell) and more difficult single cell datasets.

## 3.2 Rectangle Puzzles

Consider $n+1$ points $x_0 = 0 < x_1 < \ldots < x_n$ on $x$-axis and $m+1$ points $y_0 = 0 < y_1 < \ldots < y_m$ on $y$-axis. Points $(x_i, y_j)$ form a 2-dimensional grid consisting of $n \cdot m$ rectangles filling up the grid with corners $(0,0), (0,y_m), (x_n,0)$ and $(x_n,y_m)$. By analogy with the jigsaw puzzle assembly, we assume that the grid is "painted" and the goal is to assemble small rectangles into the painted grid in such a way that rectangles fully fill up the grid and that the colors at the sides of neighboring rectangles match (*valid assembly*). To simplify the matters, we will assume that the "orientation" of each rectangle is known.

Assembling the Ha Long Bay puzzle in Fig. 3.1a is trivial (for every rectangle, there exists an unambiguous choice of neighboring rectangles). Fig. 3.1b shows 9 rectangles from this puzzle with 12 blue dotted lines connecting the *unique* matching sides of these rectangles. Fig. 3.1c shows a more difficult "frogs and butterflies" puzzle (for every rectangle, there are *multiple* choices of neighboring rectangles). Fig. 3.1d shows 9 rectangles from this puzzle with 6 dotted connections showing all rectangles that match the upper side of the lower-left rectangle. Even a seasoned puzzle enthusiast may have difficult time assembling this puzzle and may end up with a wrong assembly shown in Fig. 3.1e [1]. Below we introduce

[1] Polynomial algorithms for assembling such puzzles remain unknown.

a simpler type of puzzles (shown in Fig. 3.1f and called *rectangle puzzles with traversing curves*) and discuss algorithms for their assembly.

Consider a continuous non-self-intersecting curve from $(0,0)$ to $(x_n, y_m)$ in the grid that crosses the sides of the rectangles at points $p_0 = (0,0), p_1, \ldots, p_N = (x_n, y_m)$ (in their order along the curve). For convenience, we assume that points on this curve are painted "red" and no other points in the puzzle is painted red. The curve is called *traversing* [2] if $p_i$ and $p_{i+1}$ belong to different sides of a rectangle (for $0 \leq i \leq N-1$) and if there are exactly two red points on the sides of each rectangle. For simplicity we assume that the direction of the traversing curve within each rectangle is known and that the curve does not pass through the corners of rectangles except for the points $p_0 = (0,0)$ and $p_N = (x_n, y_m)$.

In the Rectangle Puzzle Problem we assume that red points in the grid form a traversing curve and the goal is to assemble the grid from rectangles. More precisely, we want to generate all valid assemblies of the rectangles into the grid. Fig. 3.1g shows 9 rectangles from this puzzle with blue dotted lines connecting all matching sides of these rectangles and illustrates that the number of matching sides is reduced as compared to Fig 3.1d.

The red curve enters and leaves each rectangle $R$ through sides that we call $source(R)$ and $sink(R)$ correspondingly. We assume that every side $S$ of a rectangle is assigned a label $label(S)$ (that encodes the painting of this side) and that differently painted sides are assigned different labels. We represent a rectangle $R$ by a directed edge $edge(R)$ from vertex $label(source(R))$ to vertex $label(sink(R))$. For convenience, we assign identical and unique labels to the sides of rectangles containing the first and the last points $(0,0)$ and $(x_n, y_m)$ on the red curve. It corresponds to closing the traversing curve as shown in Fig. 3.1g.

The concept of the traversing curve turns out to be useful for bringing the de Bruijn graph concept in the domain of puzzle assembly. Below, we describe an application of this concept for assembling rectangle puzzles.

**Rectangle Graphs**. The concept of rectangle graphs was first described in [9]. Given a rectangle puzzle, its *rectangle graph* is constructed as follows:

- Define a graph $G$ with $n \cdot m$ isolated edges (on $2 \cdot n \cdot m$ vertices) by introducing a directed edge $edge(R)$ for each rectangle $R$. Starting and ending vertices of $edge(R)$ are labeled as $label(source(R))$ and $label(sink(R))$, correspondingly.
- The rectangle graph is formed by gluing identically labeled vertices in $G$ (Fig. 3.1h).

[2] Intuitively, a traversing curve is a curve that "visits" every rectangle exactly once.

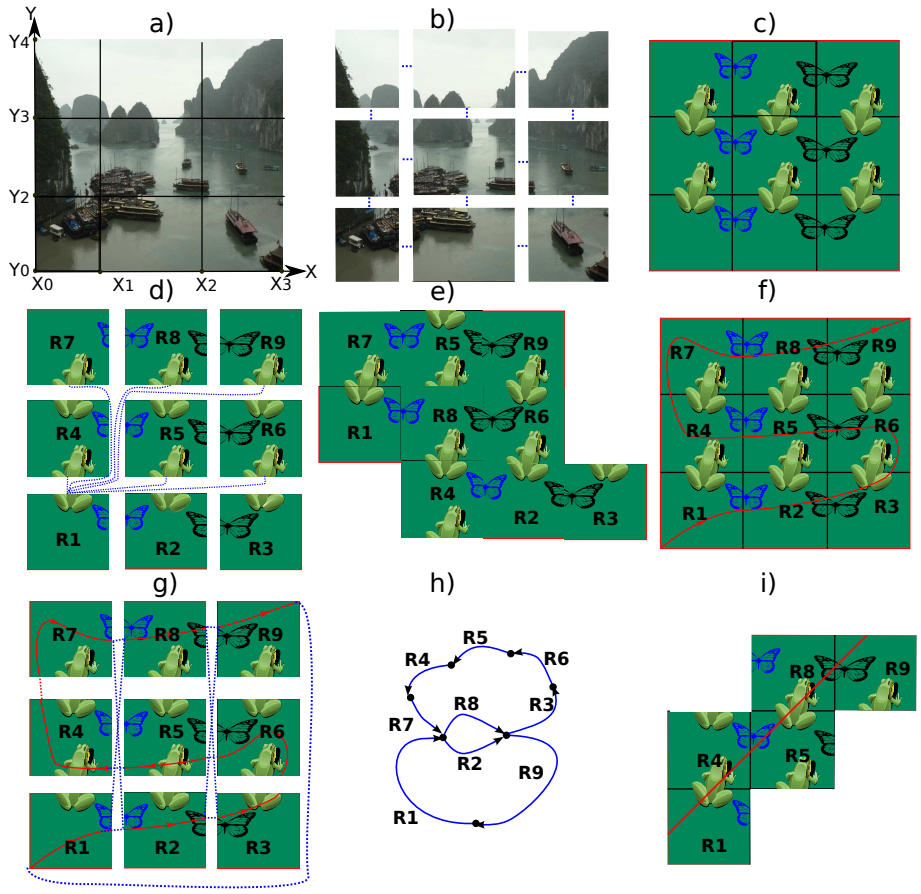



Figure 3.1: Rectangle puzzles and rectangle graphs. a) A simple puzzle on the background image of Ha Long Bay. Points $x_0, \ldots, x_3$ on $x$-axis and $y_0, \ldots, y_3$ on $y$-axis form a $3 \times 3$ grid. (b) Nine rectangles with twelve matching sides in the Ha Long bay puzzle (shown by blue dotted connections) illustrate that there exists an unambiguous choice of matching sides. (c) A more difficult "frogs and butterflies" puzzle with multiple ambiguous choices of matching sides. (d) Nine rectangles with multiple matching sides (only some of them are shown) illustrate ambiguities in the selection of matching sides. (e) Failed attempt at the "frogs and butterflies" puzzle assembly. (f) The traversing curve in the "frogs and butterflies" puzzle makes the assembly easier. (g) Gluing sides of the rectangles in the "frogs and butterflies" puzzle. (h) The rectangle graph with 2 Eulerian cycles: $R_1R_2R_3R_6R_5R_4R_7R_8R_9$ and $R_1R_8R_3R_6R_5R_4R_7R_2R_9$ where only the first represents a valid solution. (i) Traversing line and subgrid assembly. The red traversing curve is replaced by a line. We are interested in assembling the subgrid formed by all rectangles crossed by the red line.

Obviously, the rectangle graph is the de Bruijn graph on strings of length 2 in the alphabet of labels (each label encodes a side of a rectangle). Each rectangle assembly corresponds to an Eulerian cycle in the rectangle graph. All Eulerian cycles can be generated using the BEST [1] theorem thus reducing the rectangle puzzle assembly to enumerating Eulerian cycles in the rectangle graph [2]. However, not every Eulerian cycle corresponds to a valid solution of the rectangle puzzle since some solutions may correspond to: (i) an assembly where rectangles overlap, (ii) an assembly that does not form a rectangular grid, (iii) an assembly where some sides of rectangles do not match. While the number of Eulerian cycles may be large, it is easy to check if a given Eulerian cycle corresponds to a valid rectangle puzzle assembly in linear time.

Below we limit attention to traversing *lines* (rather than *curves*) and relax the condition of *visiting* all rectangles (Fig. 3.1i): The traversing line $y = x + d$ visits *some* (not necessary all) rectangles. In this case we are only interested in assembling rectangles into a *subgrid* formed by rectangles crossed by the red line, rather than assembling all rectangles into the full grid. For $d \neq 0$, the traversing line does not necessarily starts at $(0,0)$ or ends at $(x_n, y_m)$. In this case, every Eulerian cycle corresponds to a valid subgrid assembly. It is easy to see that in the case of the traversing *line* (in difference from the traversing *curve*) no additional checks are needed to verify that the assembly (given by an Eulerian cycle) is valid. Below we continue using the term "rectangle puzzle" while referring to the case of traversing *lines* (rather than traversing *curves*).

## 3.3 Rectangle Puzzles and Genome Assembly

**Generating a Rectangle Puzzle from a Genome**.

We represent a genome as a circular string over the alphabet of nucleotides $\{A,T,C,G\}$. A *k-mer* is a string of length $k$ in the alphabet of nucleotides.

Given a $k$-mer $s = s_1 \dots s_k$, we define $prefix(s) = s_1 \dots s_{k-1}$ and $suffix(s) = s_2 \dots s_k$. Given a *Genome*, the de Bruijn graph $DB(Genome, k)$ is defined on the set of vertices representing all $(k-1)$-mers from *Genome* and has a directed edge $(prefix(s), suffix(s))$ for each $k$-mer $s$ appearing in *Genome*. It is easy to see that *Genome* defines an Eulerian cycle in its de Bruijn graph.

A vertex $v$ in a graph precedes (follows) a vertex $w$ if there exists an edge from $v$ to $w$ (from $w$ to $v$). The indegree (outdegree) of a vertex is the number of vertices preceding (following) it. A vertex is called a branching vertex if either its indegree or its outdegree is larger than 1. A path in a graph is called a *non-branching path* if all vertices in this path (with exception of the first and the last ones) have

indegree and outdegree both equal to 1.

The de Bruijn graph $DB(Genome,k)$ partitions $(k-1)$-mers from *Genome* into branching (if they correspond to branching vertices in $DB(Genome,k)$) and non-branching. Similarly, all positions in *Genome* are partitioned into branching (if the $(k-1)$-mer starting at this position is branching) and non-branching. For example, ACG, CGT, GTT, and TCT are the branching 3-mers in *Genome* (shown as red points in Fig. 3.2a) while 0, 1, 7, 9, 13, 14, 19, 21, 24 are branching positions (shown as red points in Fig. 3.2b). For convenience, we assume that the circular genome "starts" at a branching position 0 and "ends" at the branching position $N$.[3]

We denote the branching positions in *Genome* as $x_0 = 0 < x_1 < \ldots x_n = N$ and define the grid consisting of $n \cdot n$ rectangles formed by points $x_0 = 0 < x_1 < \ldots < x_n = N$ on $x$ axis and the same list of points on $y$-axis. The segment of *Genome* between positions $x_i$ and $x_{i+1}$ corresponds to a non-branching path in the de Bruijn graph. Thus, every rectangle corresponds to a pair of non-branching paths. The red line is defined by the equation $y = x + d$ [4]. Given an integer $d$, we define $Puzzle(Genome,k,d)$ as a set of rectangles crossed by the red line. Each rectangle in this set is uniquely defined by a pair of non-branching paths and the position of a red line segment within the rectangle.

Given a position $x$ in *Genome* we define $(\bar{x})$ as the $(k-1)$-mer starting at this position. Thus, each integer 2D coordinate $(x,y)$ defines a *paired* $(k-1)$*-mer* $(a|b)$, where $a = (\bar{x})$ and $b = (\bar{y})$. The label ("paint") of position $(x,y)$ in the grid is defined as $((\bar{x}),(\bar{y}),color)$. where *color* is "red" or "white" depending on whether $(x,y)$ is located on the red line or not. The label of a side of a rectangle is defined as an ordered list of all labels of (integer) points located on this side. It is easy to see that there exists an alternative simpler representation of this label, i.e., by a paired $(k-1)$-mer corresponding to the red position on the corresponding side [5]. A rectangle formed by a pair of non-branching paths $(p,p')$ together with the red line segment on it can be represented as a triple $(p,p',t)$ where $t$ is the position of the red line in the rectangle relative to the low left corner of the rectangle. See Fig. 3.2b for an example of rectangle puzzle constructed from a genome.

We mention that since the labels along the red line completely define the genome, assembling the red line from the rectangles results in assembling the genome. However, this puzzle may appear useless for genome assembly tasks since the puzzle itself was originally created from the genome that we

[3] Since the genome is circular, these two positions represent the same site in the genome and the same vertex in the de Bruijn graph.

[4] Below we will define $d$ as the median distance between reads within a read-pair.

[5] Since it uniquely defines the label of all other points on the side.

are trying to assemble in the first place! Below we show that the rectangle puzzle can be created from read-pairs without knowing the genome.

**Generating a Rectangle Puzzle from Exact-Distance Read-Pairs**.

A $(k,d)$-mer is a pair of $k$-mers separated by $d$ in the genome. If two reads $r'_1 \dots r'_n$ and $r''_1 \dots r''_n$ within a read-pair are separated by an exact distance $d$, one can extract $(k,d)$-mers $(r'_i \dots r'_{i+k-1} | r''_i \dots r''_{i+k-1})$ from them (for $1 \leq i \leq n-k+1$). Iterating over all read-pairs results in a large set of $(k,d)$-mers. For simplicity, we unrealistically assume that the resulting set contains all and only $(k,d)$-mers from the genome. Before showing that the $Puzzle(Genome,k,d)$ can be constructed only from the $(k,d)$-mers set without knowing the genome, we introduce *multirectangle* — a different but equivalent representation of rectangle pieces in the rectangle puzzle [6] (Fig. 3.3a).

Given $r$ rectangles: $(p,p',t_1),\dots,(p,p',t_r)$ formed by the same pair of non-branching paths $(p,p')$ but having different positions of their red line segments, we define a multirectangle $R^*$ as a rectangle that is formed by the same pair of non-branching paths $(p,p')$ (with horizontal edges $p$ and vertical edges $p'$) but with $r$ red line segments and represent the multirectangle as $(p,p',\{t_1,\dots,t_r\})$.

Let $Puzzle^*(Genome,k,d)$ denote a set of multirectangles by transforming rectangles

in $Puzzle(Genome,k,d)$ that are formed by the same pairs of non-branching paths into single multirectangles. Within each multirectangle $R^* = (p,p',\{t_1,\dots t_r\}) \in Puzzle^*(Genome,k,d)$, the red (integer) points on these $r$ line segments represent *all* $(k-1,d)$-mers $(a|b)$ of the genomes such that $a \in p$ and $b \in p'$ [7]. Additionally, each $(k-1,d)$-mer $(a|b)$ such that $a \in p$ and $b \in p'$, corresponds to a *unique* position in the multirectangle. These two observations lead to a simple approach for constructing the $Puzzle^*(Genome,k,d)$ from the $(k,d)$-mers set: (1) Construct the de Bruijn graph $DB(ReadPairs,k)$ from individual reads in read-pairs; (2) For each pair of non-branching paths $(p,p')$ in the de Bruijn graph that are *connected* by $(k,d)$-mers (i.e., there exists at least one $(k-1,d)$-mer $(a|b)$ such that $a \in p$ and $b \in p'$), we draw a multirectangle with horizontal edges $p$ and vertical edges $p'$, together with red points corresponding to $(k-1,d)$-mer that connect $p$ and $p'$. These points form a single or multiple red line segments within the multirectangle.

From the set of multirectangles, we further transform it into the rectangle puzzle by replacing each multirectangle by separated rectangles, each containing a *single* red line segment (see Fig. 3.3a).

[6] While introducing the multirectangle concept does not have any analogy to the jigsaw puzzle assembly, it simplifies the proof that the $Puzzle(Genome,k,d)$ can be constructed only from the set of all $(k,d)$-mers of the unknown genome.

[7] With a minor exception for points that lie on the edges of the multirectangles.

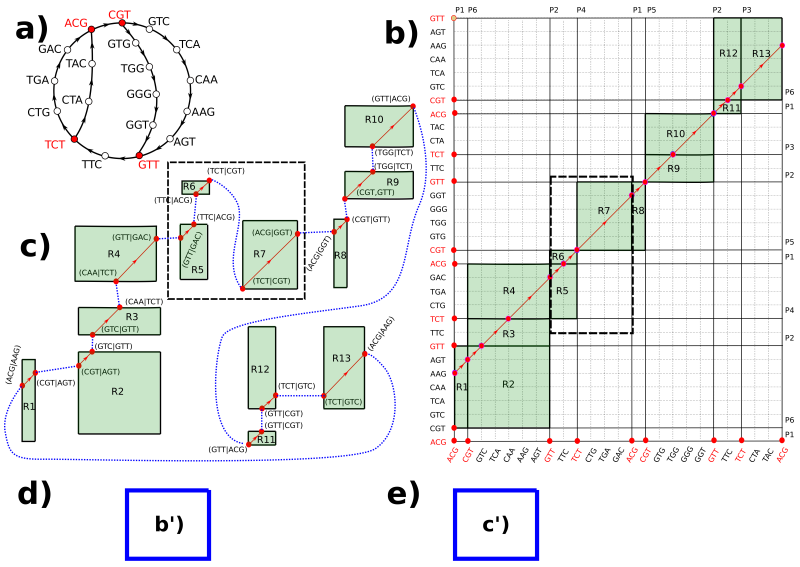



Figure 3.2: The rectangle puzzle and the rectangle graph of the genome $Genome = ACGTCAAG\,TTCTGACGTGGGTTCT$. (a) De Bruijn graph $DB(Genome,k)$ with $k = 4$ can be constructed from *Genome* or individual reads generated from *Genome*. The graph has 4 branching vertices (ACG, CGT, GTT, TCT) colored red that correspond to 8 branching positions in *Genome*. (b) Generating rectangle puzzle when *Genome* is known. *Genome* is represented as a sequence of 3-mers in both vertical and horizontal axes. The 3-mers corresponding to the branching vertices in the de Bruijn graph are colored red. The set of all $(3,5)$-mers (pair of 3-mers separated by 5 nucleotides in the genome) forms a line $(d) : y = x+5$ on the grid. (c) Rectangle graph is obtained by gluing sides of rectangle with the same labels. (d) The same as figure (b) but with rectangles in the dash box $(R_5, R_6, R_7)$ removed. $R_5, R_6, R_7$ represent 3 missing rectangles. (e) The same as figure (c) but with rectangles in the dash box $(R_5, R_6, R_7)$ missing. This results in two dead-ends vertices (sides of $R_4$ and $R_8$) in the rectangle graph.

**Generating a Rectangle Puzzle from Inexact-Distance Read-Pairs**.

We now show how to construct the rectangle puzzle in a more realistic case of read-pairs with inexact distances between reads. Given integers $d$ and $\Delta$,[8] a pair of $k$-mers $(a|b)$ is called a $(k,d,\Delta)$-mer in *Genome* if it is a $(k,d_0)$-mer of *Genome* for some $d_0 \in [d-\Delta, d+\Delta]$. While the set of all $(k-1,d)$-mer of *Genome* forms a line $(d) : y = x+d$ in the 2D grid, a set of $(k-1,d,\Delta)$-mers fills up a band of width $\Delta$ around $(d)$, called a $\Delta$-*cloud*. In this case, the rectangle puzzle needs to be redefined since: (1) red line segments in rectangles are substituted by red $\Delta$-clouds, making it difficult to infer the position of the red line segments within the rectangles; (2) some new rectangles with red points are added into the original set of rectangles crossed by the red line (false rectangles); (3) some rectangles crossed by the red line are now missing (missing rectangles). Below we address these complications.

For each pair of non-branching path $(p,p')$ in the de Bruijn graph that is connected by $(k-1,d,\Delta)$-mers, we form a *multirectangle* with horizontal edge $p$ and vertical edge $p'$ together with red

[8] Integer $d$ refers to the median distance between reads within a read-pair while integer $\Delta$ refers to the maximum deviation of the distance between the reads within a read-pair.

points corresponding to the $(k-1,d,\Delta)$-mers $(a,b)$ where $a \in p$ and $b \in p'$. While in the case of the exact distance and perfect coverage, red points define a collection of line segments in the multirectangle, in the case of inexact distance these points *fall into* a band of width $\Delta$ around these (unknown) red line segments. Thus, red points in each multirectangle should be somehow transformed into the red line segments, a difficult task. Below, we introduce the notion of $(k-1,d)$-tuple, which enables us to draw all possible positions of the red line segments, and later, using the red points in the multirectangle to classify these segments into correct/incorrect red line segments.

Given the de Bruijn graph *DB*, we define a $(k-1,d)$-*tuple* as a pair of $(k-1)$-mers $(a|b)$ such that there exists a path of length $d$ between vertex $a$ and vertex $b$ in *DB*. Obviously, every $(k-1,d)$-mer in *Genome* corresponds to a $(k-1,d)$-tuple in *DB* (but not vice versa).

Given a multirectangle formed by a pair of paths $(p,p')$ and a collection of red points within it, we generate all possible [9] $(k-1,d)$-tuples $(a|b)$ such that $a \in p$ and $b \in p'$. These $(k-1,d)$-tuples define 45 degree line segments within this multirectangle. Those $(k-1,d)$-tuples that are also $(k,d)$-mers, form correct red line segments, while tuples that are not $(k,d)$-mers, form incorrect line segments. However, such a classification is unknown and we attempt to infer the correct/incorrect red line segments by the red points (corresponding to the $(k-1,d,\Delta$-mers) in the multirectangle.

Intuitively, correct line segments usually lie close to the "center" of red $\Delta$-clouds, while the incorrect ones have few red points surrounding them. However, correctly classifying these segment into correct/incorrect segments still remains a difficult problem[10], since in the case of closely located red line segments, it is difficult to rule out which of them is correct (or whether they both are correct) and often forces us to combine such segments into a *a cluster* (see Fig. 3.3b) within an assumption that at least one of red line segments in the cluster represents a correct red segment. Below we describe the rectangle graph approach in the case when we deal with clusters of red segments.

In this case, we still represent each rectangle $R$ as a single edge $edge(R)$ but use *multiple* labels for its starting and ending vertices (in the past we labeled these vertices by a single label). Specifically, we label its starting (ending) vertex by a multiset of all starting (ending) points of red segments. The *multilabeled rectangle graph* is defined as follows:

- Form a directed edge $edge(R)$ for each rectangle $R$. Starting (ending) vertices of $edge(R)$ are labeled by a *set* of labels of all starting (ending) points of the red segment within this rectangle.

[9] If no such $(k-1,d)$-tuple exists, we remove the multirectangle.
[10] A similar problem was addressed in [9, 74].

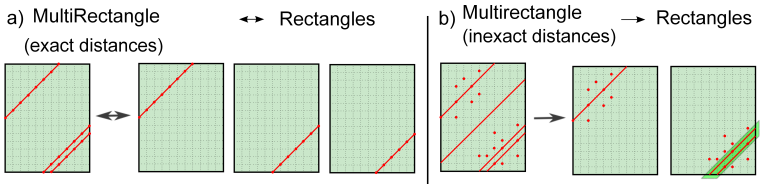



Figure 3.3: Rectangles and red line segments. a) Multirectangle: an equivalent representation of multiple rectangles that are formed by the same non-branching paths but have different positions of the red line segments. b) A multirectangle is transformed into 2 rectangles (one of them represents a cluster of two closely positioned red line segments). Note that one segment (the longest) in the multirectangle was classified as incorrect (there is no red point around this segment) and further being removed.

- The rectangle graph is formed by gluing vertices in $G$ if their sets of labels overlap.

Given a multirectangle $R$, SPAdes+ identifies $T$ *clusters* of line segments that are supported by *ReadPairs* using a variation of the approach from [9]. It further generates $T$ rectangles (each rectangle with a single cluster of red segments as in Fig. 3.3b) and applies the multilabeled rectangle graph to assemble the resulted rectangles.

**Missing Rectangles.** We now consider the case when some rectangles are missing and ask whether the missing rectangles can be somehow reconstructed to complete the puzzle. Fig. 3.2d illustrates the case of 3 missing rectangles (R5, R6, and R7) resulting in a "gap" in the rectangle graph between vertices $(GTT|GAG)$ and $(ACG|GGT)$ in Fig. 3.2e. These *dead-ends* vertices (i.e., vertices with indegree or outdegree zero) provide a clue that some rectangles are missing and, as we show below, often allow one to recover the missing rectangles.

Consider two points with integer coordinates $(x, x+d)$ and $(x+t, x+d+t)$ in the grid located at the intersection of the red line with the sides of the rectangles. We refer to labels of these points as $(a|b)$ and $(a'|b')$, correspondingly. For example, points (7,11) and (13,17) in Fig 3.2d correspond to paired $(k-1)$-mers $(GTT|GAG)$ and $(ACG|GGT)$. Given *Genome*, we define $Rectangles((x,y) \to (x',y'))$ as the set of all rectangles crossed by the segment of the red line between points $(x, x+d)$ and $(x+t, x+d+t)$. For example, $Rectangles((7,11) \to (13,17)) = \{R5, R6, R7\}$.

Fig. 3.2c presents an idealized case when *Genome* as well as the points (7,11) and (13,17) (that contain vertices from some missing rectangles) are known. In reality, this information is not available in genome assembly projects. However, one knows the de Bruijn graph and can guess the labels of the points (7,11) and (13,17) (as labels of the dead-ends vertices in the rectangle graph in Fig. 3.2e). This raises the question whether the missing rectangles $Rectangles((7,11) \to (13,17))$ can be inferred from the paired $(k-1)$-mers $(GTT|GAG)$ and $(ACG|GGT)$ (that represent labels of points (7,11) and (13,17)) *without*

knowing the coordinates of these points. Given paired $(k-1)$-mers $(a|b)$ and $(a'|b')$, below we define the set of rectangles $Rectangles((a|b) \to (a'|b'))$ that often approximates $Rectangles((x,y) \to (x',y'))$ well.

Given an integer $t$, paired $(k-1)$-mers $(a|b)$ and $(a'|b')$ are called *$t$-tied* if there exist instances of $a, b, a', b'$ located, respectively, at positions $x, y, x+t, y+t$ in *Genome*. Labels of every two red points in the grid represent $t$-tied paired $(k-1)$-mers. Below we relax the definition of $t$-tied paired $(k-1)$-mers for the case when *Genome* is unknown and only the de Bruijn graph of *Genome* is given.

Given an integer $t$ and a de Bruijn graph *DB*, paired $(k-1)$-tuples $(a|b)$ and $(a'|b')$ are called *$t$-linked* if there exists a path $p = p_0 \dots p_t$ of length $t$ between $a$ and $a'$ and a path $q = q_0 \dots q_t$ of the same length between $b$ and $b'$ in the de Bruijn graph *DB*. Obviously, every $t$-tied paired $k$-mers is also $t$-linked, but not vice versa. Paths $p$ and $q$ define $t+1$ paired $(k-1)$-tuples $(p_i|q_i)$ that may potentially belong to the red line (since the notion of "$t$-linked" is a relaxation of the notion of "$t$-tied"). We define $Rectangles_{p,q}((a|b) \to (a'|b'))$ as the set of all rectangles that contain at least one point $(p_i|q_i)$ (for $0 < i < t$). We will often refer to $Rectangles_{p,q}((a|b) \to (a'|b'))$ as simply $Rectangles((a|b) \to (a'|b'))$ when it does not cause a confusion.

For example, $(GTT|GAG)$ and $(ACG|GGT)$ are 6-linked since there exists a path $p$ ($q$) of length 6 from GTT to ACG (from GAG to GGT) in the de Bruijn graph in Fig. 3.2a. The vertices of the path $p(q)$ are located on 2 (3) non-branching paths in the de Bruijn graph thus contributing to $2 \times 3 = 6$ rectangles. Only 3 of these 6 rectangles ($R5, R6$, and $R7$) contain red points implying that $Rectangles((GTT|GAG) \to (ACG|GGT)) = \{R5, R6, R7\}$.

This example illustrates that one can close the gap between the dead-ends vertices $(a|b)$ and $(a'|b')$ in the rectangle graph by simply finding $t$-linked dead-ends in the rectangle graph (for small values of $t$), generating the set of missing rectangles $Rectangles((a|b) \to (a'|b'))$, and adding these missing rectangles to the pool of previously generated rectangles. Finally, one can construct the rectangle graph from the resulting enlarged set of rectangles.

## 3.4 Results

**Assembly Datasets**. To evaluate rectangle graph algorithm for genome assembly, we assembled two paired-end datasets from [28]. All these datasets are Illumina short reads with 100bp read length, 600× coverage. The first dataset is the multiple cell *E.coli* dataset with average insert size 215 bp, and denoted as ECOLI-MC. The second dataset is single cell *E.coli* dataset with average insert size 266 bp.

**Benchmarking**. We compare our SPAdes+ algorithm with EULER-SR [25], SOAPdenovo [53], Velvet [95], Velvet-SC [28], E+V-SC [28] and SPAdes [9]. See Table 3.1. Our rectangle graph algorithm

Table 3.1: Comparison of assemblies for single-cell (ECOLI-SC) and two standard (ECOLI-MC) datasets. Assembly quality assessment was made by QUAST 1.3. The best assembler by each criteria is indicated in bold. EULER-SR 2.0.1, Velvet 0.7.60, and E+V-SC were run with vertex size 55. SOAPdenovo 1.0.4 was run with vertex size 27–31. SPAdes-single refers to SPAdes without using read-pair information. SPAdes iterated over vertex sizes $k-1 = 21, 33, 55$. Only contigs of length longer than 500 bp are reported.

| Assembler | # contigs | NG50 (bp) | Largest (bp) | Total (bp) | Covered (%) | #Misassemblies | # Complete genes |
|---|---|---|---|---|---|---|---|
| **MC215: A multicell *E. coli* dataset** | | | | | | | |
| EULER-SR | **100** | 110153 | 221409 | 4574240 | 99.0 | 7 | 4220 |
| SOAPdenovo | 141 | 62512 | 172567 | 4519621 | 97.6 | **0** | 4157 |
| Velvet | 120 | 82776 | 242032 | 4554702 | 99.5 | 3 | 4231 |
| IDBA-UD | 107 | 111789 | 236470 | 4562955 | 99.6 | 1 | 4222 |
| SPAdes-single | 158 | 60600 | 166117 | 4544186 | 99.3 | **0** | 4190 |
| SPAdes | 102 | 119880 | 265405 | 4639596 | **99.9** | 3 | 4278 |
| Rectangles | 104 | **133918** | **285115** | 4633639 | **99.9** | 1 | **4285** |
| **MC480: A multicell *E. coli* dataset** | | | | | | | |
| EULER-SR | 784 | 11793 | 60466 | 3966931 | 85.7 | 6 | 3223 |
| SOAPdenovo | 984 | 57167 | 173976 | 4527198 | 98.4 | **0** | 4131 |
| Velvet | 169 | 105637 | 209338 | 4531649 | 99.1 | 4 | 4225 |
| IDBA-UD | 185 | 112430 | 224018 | 4556916 | 99.5 | **0** | 4223 |
| SPAdes-single | 493 | 60600 | 173976 | 4536189 | 99.0 | **0** | 4159 |
| SPAdes | **90** | 130544 | 224341 | 4615709 | **99.8** | 3 | 4251 |
| Rectangles | 101 | **132556** | **225631** | 4633124 | **99.8** | 1 | **4274** |
| **SC266: A single-cell *E. coli* dataset** | | | | | | | |
| EULER-SR | 429 | 26662 | 140518 | 4248713 | 85.4 | 18 | 3431 |
| SOAPdenovo | 569 | 18468 | 87533 | 4098032 | 80.2 | 6 | 3055 |
| Velvet | 261 | 22648 | 132865 | 3501984 | 74.7 | 2 | 3105 |
| IDBA-UD | 250 | 96947 | 224018 | 4791744 | 96.4 | 10 | 4054 |
| SPAdes-single | 354 | 55617 | 166064 | 4786498 | 96.0 | **0** | 3999 |
| SPAdes | 279 | 109876 | 268093 | 4960147 | **97.0** | 3 | 4084 |
| Rectangles | **238** | **118981** | **284497** | 4884705 | 96.5 | 2 | **4103** |

outperforms other assemblers in most metrics. Improvement is more significant for single cell dataset. On ECOLI-SC dataset, SPAdes+ produces contigs with higher N50 (56,842 bp vs 49,623 by SPAdes), with higher largest contig (209,690 bp vs 177,944 by SPAdes) and no misassemblies, also captures 64 additional *E. coli* genes (3975 vs 3911 by SPAdes).

For both *E. coli* datasets, the rectangle graph module works for less than 10 seconds and using less than 100 MB RAM given (1) the de Bruijn graph has been already constructed and (2) mapping information of all paired-end reads to the de Bruijn graph has been calculated (using other modules in [9].

## 3.5 Conclusion

In this paper, we modeled the problem of genome assembly using read-pairs as a simple jigsaw puzzle and reintroduced the notion of rectangle graph [9] in a more intuitive way. We further addressed algorithmic challenges that arise in the application of rectangle graphs that have not been addressed in [9]. We demonstrated that by addressing these algorithmic challenges, the quality of the assembly significantly improves for both single cell and multicell bacterial datasets.

Chapter 3, in part, is a reprint of the material as it appears in Nikolay Vyahhi, Alex Pyshkin, Son Pham and Pavel Pevzner, “From de Bruijn Graphs to Rectangle Graphs for Genome Assembly”. WABI 2012, pp. 249-261. The dissertation author was the corresponding author of these paper.

# Chapter 4

# Pathset Graphs: A New Approach for Comprehensive Utilization of Mate-Pairs in Genome Assembly

## 4.1 Introduction

Current High Throughput Sequencing (HTS) technologies have reduced the time and costs of genome sequencing and have enabled new experimental opportunities in a variety of applications. However, as sequencing technologies improve, the challenges in designing software to assemble genomes get harder. Current genome assemblers face the challenge of assembling billions of short reads. When the length of a repeat is longer than the read length, correctly matching up its upstream and downstream flanking regions is difficult. Fortunately, all current sequencing platforms are able to produce mate-pairs — pairs of reads whose separation in the genome (called the insert size) is approximately known. Because insert sizes may be much longer than the read length, mate-pairs may span over long repeats and match up their flanking regions.

Mate-pair information has played an important role in most genome projects and many HTS assemblers incorporate mate-pair information to increase the contigs length in a post-processing step [67, 25, 95, 19, 83, 24, 53]. Most of these assemblers rely on the same basic observation [67]: if there is a unique path of suitable length in the assembly graph that connects the left and right reads of a mate-pair, the gap in the mate-pair can be filled in with the nucleotides spelled by this path (*mate-pair transformation*). However, when multiple paths exist between the left and right reads within a mate-pair, it remains ambiguous which path should be used to fill in the gap. Unfortunately, mate-pairs generated by existing HTS protocols are characterized by rather large variations in insert sizes, leading to multiple paths for a significant fraction of mate-pairs (since the range of suitable path lengths is wide), making it difficult to

utilize such mate-pairs.

Recently, [57] introduced the notion of the *paired de Bruijn graph*, which incorporates mate-pair information in the graph structure rather than using it for mate-pair transformations in a post-processing step. Similar methods were also developed independently [27, 35]. Unfortunately, the performance of these methods deteriorates as the variation in the insert size increases.

Below we show that while reducing variation in the insert size remains a difficult experimental problem, it can be addressed computationally by aggregating the mate-pair information for all mate-pairs linking a pair of edges in the condensed de Bruijn graph. This transforms mate-pairs into *edge-pair histograms*, consisting of pairs of edges together with distances aggregated from mate-pair information.

Using the collection of edge-pair histograms, we further combine the ideas of mate-pair transformations and paired de Bruijn graphs into new data structures for genome assembly, called *pathsets* and *pathset graphs*. We further compare the performance of our assembler (based on the pathsets approach) to other assemblers on various bacterial datasets.

## 4.2 Methods

### 4.2.1 Assumptions

There are many sources of error and variation in Next Generation Sequencing technologies. This paper focuses on issues arising from paired reads. For theoretical development of our methods, in this section we will assume all reads are perfect, with no local errors like point mutations, insertions, or deletions. We will also assume perfect coverage, with a $k$-mer starting at every position in the genome. Our focus is on the complexities of using paired reads in assembly: (1) insert size variation; (2) repeats for which a read pair maps as a pair to two or more places in the genome, resulting in multiple ways to fill in the region between those two reads; and (3) chimeric read pairs.

In Sec. 4.2.9 and Results, we demonstrate that our approach can be adapted to work well with various bacterial datasets where read errors and imperfect coverage are present.

### 4.2.2 De Bruijn graphs

We represent genomes as circular strings over $\{A,T,G,C\}$, the alphabet of nucleotides. An $n$-mer is a string of length $n$. Given a $n$-mer $s = s_1 \dots s_n$, we define $\textsc{prefix}(s) = s_1 \dots s_{n-1}$ and $\textsc{suffix}(s) = s_2 \dots s_n$.

Let $k$ be a fixed parameter. Given a set $A$ of $k$-mers, the standard de Bruijn graph has a directed

edge $(\text{PREFIX}(s), \text{SUFFIX}(s))$ for each $k$-mer $s \in A$. The condensed de Bruijn graph $\text{CG}(A,k)$ is obtained from the standard de Bruijn graph by replacing every maximal non-branching path[1] $P$ by a single edge $e$ of length $\ell(e)$ equal to the number of edges in $P$ (see Fig. 4.1a,b). Below we work with condensed (rather than standard) de Bruijn graphs and refer to them simply as de Bruijn graphs.

Each $k$-mer $a \in A$ maps to an edge $e = \text{EDGE}(a)$ in $\text{CG}(A,k)$ at position $\text{OFFSET}(a)$ ($1 \leq \text{OFFSET}(a) \leq \ell(e)$). For a path $P = e_1 \dots e_n$ in a graph $\text{CG}(A,k)$, we define $d_P(e_i, e_j) = \Sigma_{t=i}^{j-1} \ell(e_t)$. We further define the length of $P$ as $d_P(e_1, e_n)$, i.e., the total length of its edges, excluding the last edge.

Given a parameter $d$, a pair of $k$-mers $a$, $b$ form a *$(k,d)$-mer* $(a|b)$ of string $S = s_1 \dots s_n$ if there are instances of $a = s_i \dots s_{i+k-1}$ and $b = s_{i+d} \dots s_{i+d+k-1}$ in $S$ whose starting positions differ by $d$ nucleotides. Given parameters $d$ and $\Delta$, a pair of $k$-mers $(a|b)$ is called a $(k,d,\Delta)$-mer of $S$ if it is a $(k,d_0)$-mer of $S$ for some $d_0 \in [d-\Delta, d+\Delta]$.

We call $a$ and $b$ the *left* and *right* $k$-mers of the pair $(a|b)$, respectively, and we refer to the distance between them as the *distance* of $(a|b)$. In particular, error-free pairs of reads of length $l$ with exact insert size $s$ form $(l, s-l)$-mers (Fig. 4.2). For a set $B$ of $k$-mer pairs, we let $\text{LEFT}(B)$ (resp., $\text{RIGHT}(B)$) be the set of left (resp., right) $k$-mers appearing in $B$.

We transform each pair of reads of length $l$ into a sequence of $l-k+1$ consecutive $k$-mer pairs. For the set $B$ of resulting $k$-mer pairs, we define the *de Bruijn graph of reads* as $\text{CG}(\text{LEFT}(B) \cup \text{RIGHT}(B), k)$.

### 4.2.3 From $k$-mer pairs to edge-pair histograms

We assume for simplicity that the genome defines a *genomic walk* $W$ that passes through all edges in the de Bruijn graph of reads.[2] Since some edges may appear multiple times in the genomic walk, we distinguish between the edge $e$ and its instance $\tilde{e}$. Every two instances $\tilde{e}_1$ and $\tilde{e}_2$ of edges $e_1$ and $e_2$ define a *genomic distance* $d_W(\tilde{e}_1, \tilde{e}_2)$ between edges $e_1$ and $e_2$. This in turn yields a triple $(e_1, e_2, d_W(\tilde{e}_1, \tilde{e}_2))$ called a *genomic edge-pair*. Note that a pair of edges may have multiple genomic distances if one of these edges appears multiple times in the genomic walk.

When the genomic walk $W$ is unknown, genomic edge-pairs can be computed from the set of

[1] A path is *non-branching* if all its vertices (except possibly the first and last) have indegree and outdegree both equal to 1. The condensed de Bruijn graph may contain loops and multiple edges between the same pair of vertices.

[2] While this assumption is unrealistic (e.g., gaps in coverage often make the de Bruijn graphs of reads disconnected), the analysis of fragment assembly in this idealized setting can be extended to real sequencing data; see the Results section.

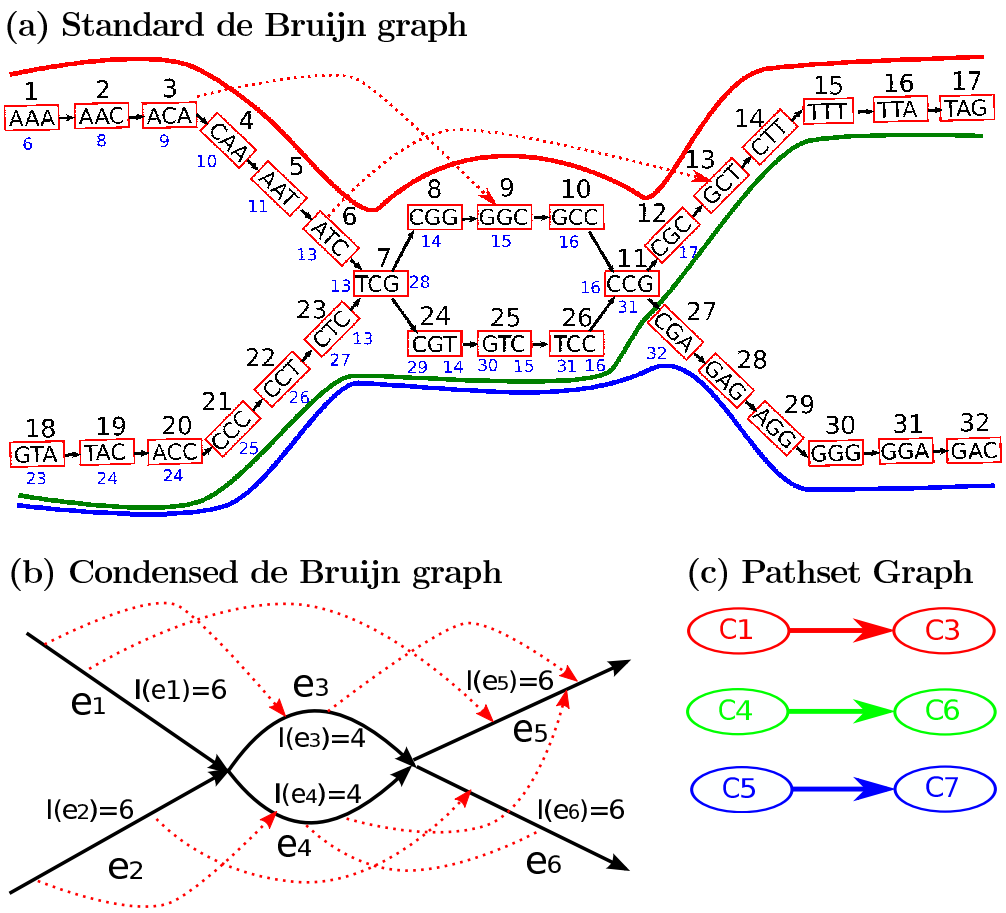



Figure 4.1: From de Bruijn graph to pathset graph. (a) A standard de Bruijn graph and the corresponding mapping of mate-pairs. The number on top of each node is the node ID. The smaller blue numbers below/beside each node are the IDs of the corresponding paired right nodes. The bold red, blue and green paths show how the genome traverses the graph. (b) The condensed de Bruijn Graph with edges corresponding to non-branching paths in the standard de Bruijn graph. The dotted red lines indicate edge-pairs. (c) Pathset graph. Initially there are eight pathsets: $C_1 = \{e_1e_3e_5, \mathbf{e_1e_4e_5}\}$, $C_2 = \{e_1e_3\}$, $C_3 = \{e_3e_5\}$, $C_4 = \{\mathbf{e_2e_3e_5}, e_2e_4e_5\}$, $C_5 = \{\mathbf{e_2e_3e_6}, e_2e_4e_6\}$, $C_6 = \{e_4e_5\}$, $C_7 = \{e_4e_6\}$, and $C_8 = \{e_2e_4\}$. Using the edge-pair information, we find phantom paths (indicated in boldface) and remove them. After removal of all prefix pathsets ($C_2$ and $C_8$), the pathset graph has six nodes and consists of three edges: $C_1 \to C_3$ (red path), $C_4 \to C_6$ (green path), and $C_5 \to C_7$ (blue path). Each edge in the pathset graph corresponds to a contig; e.g., $C_1 \to C_3$ spells out the red path (AAACAATCGGCCGCTTTAG).

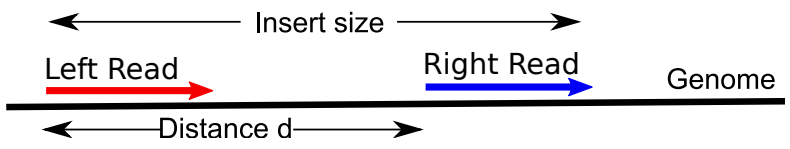



Figure 4.2: A mate-pair is a pair of reads with distance $d$ between their starting positions. The insert size is the distance from the start of the left read to the end of the right read. Each platform has reads oriented a particular way, but for presentation purposes, we canonically re-orient them as indicated.

$k$-mer pairs $B$ when the genomic distance between $k$-mers in each $k$-mer pair of $B$ is known exactly. In practice, such distances are estimated rather than exact. Below, we define *edge-pair histograms* and use them for more accurate approximation of genomic edge-pairs.

For a set $B$ of $k$-mer pairs, a pair of edges $(e_1, e_2)$ in the de Bruijn graph $\mathrm{CG}(\textsc{left}(B) \cup \textsc{right}(B), k)$ is called *B-bounded* if there exists at least one $k$-mer pair in $B$ whose left (resp., right) $k$-mer maps to the edge $e_1$ (resp., $e_2$).

Below we assume that parameters $d_0$ (estimated distance between reads within read-pairs) and $\Delta$ (maximum error in distance estimates) are fixed. Given a set of read pairs whose distances fall in the range $d_0 \pm \Delta$, one can generate a set $B$ of all $(k, d_0, \Delta)$-mers (extracted from all read-pairs), and then compute the set of $B$-bounded edge-pairs. For a $B$-bounded edge-pair $(e_1, e_2)$, we define the *edge-pair histogram* $(e_1, e_2, \mathbf{h})$, where $\mathbf{h}$ is a histogram with $\mathbf{h}(x)$ equal to the number of $(k, d_0, \Delta)$-mers in $B$ that support *genomic distance* $x$ between $e_1$ and $e_2$:

$$\mathbf{h}(x) = \#\{(a|b) \in B \mid \textsc{edge}(a) = e_1,\ \textsc{edge}(b) = e_2, \\ \text{and } d_0 + \textsc{offset}(a) - \textsc{offset}(b) = x\}.$$

While HTS machines produce large numbers of read pairs (e.g., over $10^7$ for the *E. coli* datasets we analyze below), the de Bruijn graph of reads contains a small number of edges (several thousand for our *E. coli* datasets). Thus, edge-pair histograms are typically supported by many $k$-mer pairs, which allows one to accurately estimate the genomic distance(s) between $e_1$ and $e_2$.

Since insert sizes typically follow a Gaussian distribution [26], the histogram $(e_1, e_2, \mathbf{h})$ either represents a sample from a single Gaussian distribution (if $e_1$ and $e_2$ appear once in the genomic walk) or a mixture of Gaussian distribution (if $e_1$ and $e_2$ appear multiple times in the genomic walk). Inferring each individual Guassian distribution from a sample of mixture distributions represents a challenging problem [60]. Here we sketch a simple approach to address this problem (see [8] for details of a more advanced analysis). We smooth the histogram, choose a threshold, and focus on the regions where the values of the histogram exceed the threshold (above the red line in Fig. 4.3b). The edge-pair histogram is thus transformed into one or more non-overlapping *edge-pair intervals*, each corresponding to one or more genomic edge-pairs with similar distances.

An edge-pair interval $(e_1, e_2, [a, b])$ is called *correct* if there exists a genomic edge-pair $(e_1, e_1, D)$ such that $a \leq D \leq b$. A properly chosen threshold should maximize the number of correct edge-pair intervals, while still separating the genomic edge-pairs with different distances.

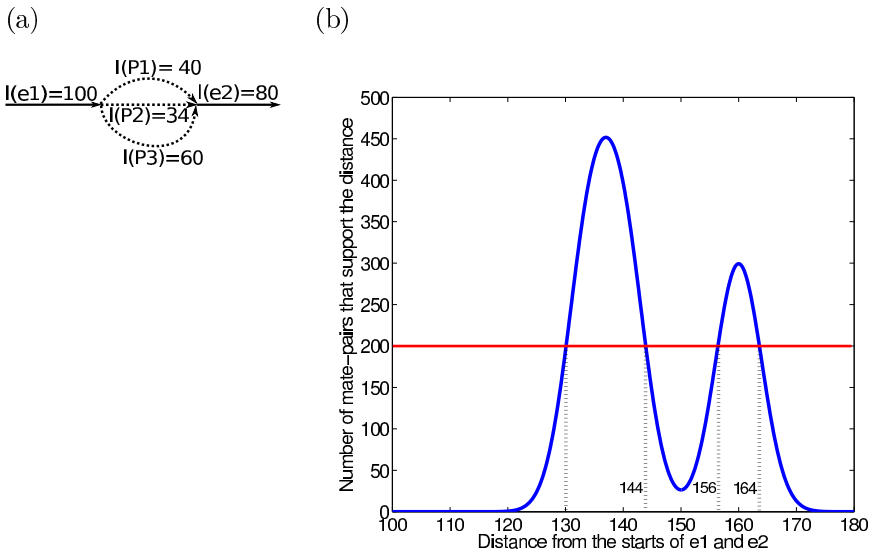



Figure 4.3: Estimating genomic distances between edges within edge-pairs using the edge-pair histogram. (a). The genome traverses the graph from $e_1$ to $e_2$ through $P_1$, $P_2$, and $P_3$. (b) The smoothed edge-pair histogram (inferred from mate-pairs mapping to $e_1$ and $e_2$) supports various genomic distances between $e_1$ and $e_2$. Setting the threshold to 200 (red line) splits the histogram into two edge-pair intervals $(e_1, e_2, [130, 144])$ (supporting the traversal via paths $P_1$ and $P_2$) and $(e_1, e_2, [156, 164])$ (supporting the traversal through path $P_3$).

Figure 4.3a shows a pair of edges $(e_1, e_2)$ that is traversed three times through the paths $P_1$, $P_2$, and $P_3$. The first two traversals, $e_1P_1e_2$ and $e_1P_2e_2$, have similar lengths while the third, $e_1P_3e_2$, has significantly larger length. Setting the threshold to 200 results in two edge-pair intervals: $(e_1, e_2, [130, 144])$ and $(e_1, e_2, [156, 164])$. The first interval supports two genomic edge-pairs, $(e_1, e_2, 134)$ and $(e_1, e_2, 140)$, while the second interval supports a single genomic edge-pair, $(e_1, e_2, 160)$.

Edge-pair intervals have two advantages over mate-pairs:

- With proper choice of thresholds, estimates of distances between edges in edge-pair intervals are more accurate than estimates of distances between individual mate-pairs. Better estimates may result in better performance of existing methods for resolving repeats, including mate-pair transformations [68], paired de Bruijn graphs [57], and mate-pair graphs [35].

- Since the edge-pair intervals compactly represent the mate-pair data, many redundant operations can be avoided; for instance, multiple function calls to perform mate-pair transformations for all mate-pairs corresponding to the same edge-pair interval (e.g., in EULER-SR [25]) can be replaced by a single mate-pair transformation of the corresponding edge-pair interval.

Below, we use edge-pair intervals to define the notions of *pathset* and *pathset graph*.

### 4.2.4 From edge-pair intervals to pathsets

We define a *pathset* as any set of paths between a fixed pair of edges.[3] Every edge-pair interval $(e_1, e_2, [a,b])$ corresponds to a pathset $\text{PATHSET}(e_1, e_2, [a,b])$ formed by all paths starting at $e_1$, ending at $e_2$, and having lengths in the interval $[a,b]$. For example, in Fig. 4.1b, $\text{PATHSET}(e_1, e_5, [9,11]) = \{e_1e_3e_5, e_1e_4e_5\}$.

As a by-product of transforming edge-pair intervals to pathsets, the set of possible genomic distances between edges in each edge-pair interval can be further reduced. Namely, the interval $[a,b]$ in an edge-pair interval $(e_1, e_2, [a,b])$ can be replaced by a list of lengths of paths in the corresponding pathset. This is referred to as an *edge-pair distance set*, or simply an *edge-pair*, and denoted $(e_1, e_2, \mathbf{d})$. If all paths in a pathset have the same length, this length reveals the genomic distance between $e_1$ and $e_2$.

A path in the de Bruijn graph is called a *genomic path* if it corresponds to a substring of the genome (i.e., is a subpath of the genomic walk), and a *phantom path* otherwise. In general, a pathset may contain multiple genomic and phantom paths. Given a collection of pathsets obtained from edge-pairs, our goal is to remove the phantom paths in each pathset and to further split pathsets with $t$ genomic paths into $t$ pathsets consisting of singleton paths. While it is not always possible to accomplish this task (since it is not clear how to separate phantom and genomic paths), below we describe some steps towards this goal.

We note that all edge-pairs are *disjoint*, i.e., for every two distinct edge-pairs $(e_1, e_2, \mathbf{d})$ and $(e_1, e_2, \mathbf{d}')$ formed by the same two edges $e_1$ and $e_2$, the sets $\mathbf{d}$ and $\mathbf{d}'$ are disjoint. A pair of edges $e_1$ and $e_2$ in the genomic walk is called $d_0$-*bounded* if at least one of its genomic distances falls in the range $[d_0 - \ell(e_2), d_0 + \ell(e_1)]$. A set of edge-pairs is *representative* if for each instance of a $d_0$-bounded pair of edges $e_1$ and $e_2$ (at genomic distance $D$), there exists a *supporting* edge-pair $(e_1, e_2, \mathbf{d})$ with $D \in \mathbf{d}$. For the sake of simplicity, we assume that a set of edge-pairs is representative and correct. Then the corresponding pathsets are also (a) disjoint (i.e., do not overlap as sets), (b) correct (i.e., each contains a genomic path), and (c) representative (i.e., every genomic path between $d_0$-bounded instances of two edges belongs to some pathset). These conditions are important for constructing the pathset graph.[4] Below we show how to remove phantom paths and split the pathsets into smaller pathsets while preserving conditions (a–c).

[3] The term "pathset" is also used in [35] to describe the construction of a different graph that represents mate-pairs.

[4] In the Results section, we demonstrate that conditions (a–c) are satisfied for the vast majority of pathsets constructed for our bacterial assembly datasets.

### 4.2.5 Removing phantom paths from pathsets.

Since our pathsets are representative (condition (c)), for each instance of a $d_0$-bounded pair of edges, there exists a supporting edge-pair. Therefore, if a path in a pathset does not have a supporting edge-pair, it represents a phantom path. Below we describe how to identify some (but not necessarily all) phantom paths.

A path $P = e_1 e_2 \dots e_n$ is *supported* by a set of edge-pairs $EP$ if for every pair of $d_0$-bounded edges $e$ and $e'$ in $P$, there exists an edge-pair $(e, e', \mathbf{d}) \in EP$ such that $d_P(e, e') \in \mathbf{d}$. The path $P$ is *strongly supported* by a set of edge-pairs $EP$ if it can be extended from both ends into a longer path $P' = u \dots e_1 \dots e_n \dots v$ such that (i) $P'$ is supported by $EP$, (ii) the pair of edges $u$ and $e_1$ is $d_0$-bounded, and (iii) the pair of edges $e_n$ and $v$ is $d_0$-bounded. Paths in a pathset that are not strongly supported are classified as phantom paths and removed.[5]

Indeed, if $P$ is a genomic path, then it is a subpath of the genomic walk. In the genomic walk, there exists an edge $u$ preceding $P$ and an edge $v$ succeeding $P$ that satisfy the properties (ii) and (iii). The subpath starting at $u$ and ending at $v$ (denoted $P'$ above) clearly satisfies property (i).

### 4.2.6 Splitting pathsets

A pathset *contains* an edge $e$ if there is a path in this pathset that contains $e$. For a pathset $PS$ and sets of edges $U = \{u_1, \dots, u_m\}$ and $V = \{v_1, \dots, v_n\}$, we define $PS_{u_1,\dots,u_m,\overline{v_1},\dots,\overline{v_n}}$ as the set of all paths in $PS$ that contain all edges from $U$ and no edges from $V$.

Two edges contained in a pathset are called *independent* if no path in this pathset contains them both. An edge $e$ is *essential* for a pathset $PS$ if there exists a genomic path in $PS$ containing $e$. A set of essential edges in a pathset is called *independent* if every two edges in this set are independent. An independent set $A = \{a_1, \dots, a_t\}$ of essential edges contained in $PS$ defines a *split* of $PS$ into $t$ disjoint pathsets:[6] $PS_{a_1}, \dots, PS_{a_{t-1}}, PS_{\overline{a_1},\dots,\overline{a_{t-1}}}$. We remark that each of these pathsets contains an essential edge from $A$ and thus is correct. It is easy to check that the split operation preserves conditions (a–c).

For example, for the pathset defined by edges $e_1$ and $e_7$ in Fig. 4.4, all edges are essential. Edges $e_2$ and $e_3$ (as well as $e_5$ and $e_6$) form an independent set.

[5] In Sec. 4.2.9, we describe how to adapt this approach for real datasets where the *representative* condition can be violated.

[6] We remark that the resulting pathsets depend on ordering of elements in $A$ (in particular, on the choice of "last" element $a_t$); different orderings may result in different splits.

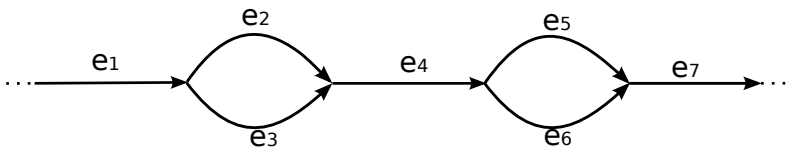



Figure 4.4: The pathset $PS = \text{PATHSET}(e_1, e_7, \mathbf{d})$ corresponding to the edge-pair $(e_1, e_7, \mathbf{d})$ contains 4 paths: $PS = \{e_1e_2e_4e_5e_7, e_1e_3e_4e_5e_7, e_1e_2e_4e_6e_7, e_1e_3e_4e_6e_7\}$. Edges $e_2$ and $e_3$ (as well as edges $e_5$ and $e_6$) form an independent set. Therefore $PS$ can be split into two pathsets: $PS_{e_2} = \{e_1e_2e_4e_5e_7, e_1e_2e_4e_6e_7\}$ and $PS_{\overline{e_2}} = \{e_1e_3e_4e_5e_7, e_1e_3e_4e_6e_7\}$ containing edges $e_2$ and $e_3$ respectively.

### 4.2.7 Identifying essential edges

To split a pathset, one has to identify essential edges. Consider an instance $\tilde{e}$ of an edge $e$ in the genomic walk. A subpath of this walk $e_1 \dots \tilde{e} \dots e_2$ is called an *$\tilde{e}$-subpath* if (i) the pair of edges $e_1$ and $\tilde{e}$ is $d_0$-bounded, and (ii) the pair of edges $\tilde{e}$ and $e_2$ is $d_0$-bounded. A maximal $\tilde{e}$-subpath (i.e., a subpath containing all other $\tilde{e}$-subpaths) is called a *span* of $\tilde{e}$. For an edge $e$, we define $\text{SPAN}(e)$ as a set of the spans of all instances $\tilde{e}$ of the edge $e$.

We refer to a pathset as $PS(a,b)$ if all paths in this pathset start at edge $a$ and end at edge $b$. Given paths $a \dots e$ and $e \dots b$ (i.e., starting and ending at the same edge $e$), we define their *concatenation* as the path $a \dots e \dots b$. Given a collection of pathsets and a fixed edge $e$, consider all pathsets $PS(a,e)$ and $PS(e,b)$ and all concatenations of paths from $PS(a,e)$ with paths from $PS(e,b)$ (for all possible choices of $a$ and $b$). Define $\text{SPREAD}(e)$ as the set of all supported paths in this set. Obviously, $\text{SPAN}(e) \subseteq \text{SPREAD}(e)$.

An edge $e$ is called $(e_1, e_2)$-*constrained* if all paths in $\text{SPREAD}(e)$ have the form $\dots e_1 \dots e \dots e_2 \dots$ and edges $e_1$ and $e_2$ in all such paths are $d_0$-bounded. It is easy to see that every $(e_1, e_2)$-constrained edge $e$ is essential in some pathset from $PS(e_1, e_2)$. Indeed, since the genomic walk contains each edge $e$ in the graph, there exists a genomic path $P$ in $\text{SPAN}(e)$. Since $\text{SPAN}(e) \subseteq \text{SPREAD}(e)$ and since $e$ is $(e_1, e_2)$-constrained, $P$ has the form $\dots e_1 \dots e \dots e_2 \dots$, where edges $e_1$ and $e_2$ are $d_0$-bounded. Therefore, the subpath $e_1 \dots e \dots e_2$ of $P$ belongs to some pathset from $PS(e_1, e_2)$, implying that $e$ is an essential edge.

### 4.2.8 Pathset graph

Given a collection of pathsets satisfying conditions (a–c), the genome assembly problem becomes similar to traditional genome assembly with each pathset playing a role of a single (long) read. Below, we define the *pathset graph* with nodes corresponding to pathsets and non-branching paths corresponding to assembly contigs.

Path $p$ is a *prefix* (resp., *suffix*) of path $q$ if $q$ can be obtained from $p$ by concatenating some non-empty path to the end (resp., start) of $p$. Pathset $PS$ is called a *prefix* of pathset $PS'$ if each path of $PS$ is a prefix of a path in $PS'$. Path $q$ follows path $p$ if they have the following form: $p = e_1 e_2 \dots e_k$ and $q = e_2 \dots e_k \dots e_{k+t}$, where $t \geq 0$. Pathset $PS'$ *follows* pathset $PS$ if there exists paths $q \in PS'$ and $p \in PS$ such that $q$ follows $p$.

A collection of pathsets is called *prefix-free* if no pathset in this collection is a prefix of another pathset. Given a collection of pathsets, we remove phantom paths, perform splits, and remove prefix pathsets to obtain a prefix-free collection of pathsets. We then construct the *pathset graph* by representing each remaining pathset as a node and forming a directed edge $PS \rightarrow PS'$ if $PS'$ follows $PS$ (similarly to the classical overlap-layout-consensus approach to fragment assembly).

Fig. 4.1c illustrates the pathset graph for a toy example. After removing phantom paths, we obtain six singleton pathsets. The pathset graph consists of six vertices and three edges (non-branching paths), corresponding to three contigs.

The pathset graph approach can be summarized as follows:

**Input:** Set of mate-pairs

1: Construct the de Bruijn graph from individual reads in mate-pairs
2: Transform read-pairs into a set of edge-pair histograms
3: Transform edge-pair histograms into edge-pair intervals
4: Transform edge-pairs intervals into pathsets
5: **for each** pathset
6: Remove phantom paths
7: Construct an independent edge-set in each pathset
8: Split the pathset over the independent edge-set
9: Remove prefix pathsets from the resulting collection of pathsets
10: Construct the pathset graph on the resulting prefix-free collection of pathsets
11: Output contigs as non-branching paths in the pathset graph

### 4.2.9 Adaptations for imperfect coverage and read errors

In contrast to the paired de Bruijn graph approach [57], which requires us to have a pair of $k$-mers starting at each position of the genome, the pathset approach only requires us to have a pathset starting at each condensed edge. In the two bacterial datasets in our Results section, we observed very

few cases where there is no such pathset. In genomes with complicated repeat structures, the number of such cases may increase. This raises two obstacles for the current approach: (1) some genomic paths may be removed; (2) some pathsets may not be extended. To address the first obstacle, we do not remove paths in singleton pathsets, and we retain the *most supported* path in each pathset if all of its paths are identified as phantom paths. For the second obstacle, if a pathset can not be extended, the algorithm finds the best possible extension pathset and extends the contig.

Additionally, some mate-pairs can have aberrant insert sizes or may contain errors that are not corrected by error correction programs. Therefore, some pathsets may not contain any genomic path. Our implementation has a procedure for removing pathsets that are not connected to any other pathsets and which have very few mapped mate-pairs.

## 4.3 Results

We implemented Pathset as a module in the new assembler SPAdes [8]. The source code is released under the GNU General Public License and is available at http://bioinf.spbau.ru/spades/pathset. To evaluate the performance of Pathset, we compared it with various assemblers on two Illumina *E. coli* datasets. The first dataset (EMBL-EBI Sequence Read Archive ERA000206, which we refer to as EC215) consists of 28 million paired reads of length 100 bp and mean insert size $\approx$ 215 bp, while the second (EMBL-EBI Sequence Read Archive ERR022075, which we refer to as EC500) consists of 44 million paired reads of length 100 bp and mean insert size $\approx$ 500 bp. The reads in each dataset were error-corrected with Quake [49].

### 4.3.1 From mate-pairs to edge-pair intervals

For each dataset, we constructed the de Bruijn graph for $k = 55$ and transformed the set of input mate-pairs into a set $EP$ of edge-pair intervals. To evaluate this transformation, we further extracted from the *E. coli* genome a set of $k$-mer pairs at fixed distance $d_0$. The mapping of these $k$-mer pairs to edges of the graph defines a set of genomic edge-pairs, $EP_0$. Table 4.1 demonstrates that for insert size $d_0 = 215$, most genomic edge-pairs in $EP_0$ (97%) are also present[7] in $EP$ and very few (0.8%) of the edge-pair intervals in $EP$ are incorrect. We also observed that for $d_0 = 500$, the proportion of incorrect edge-pairs in EC500 only slightly increases as compared to EC215.

[7] A genomic edge-pair $(e_i, e_j, d_g)$ is *present (resp., missing)* in $EP$ if there exists (resp., does not exist) an edge-pair interval $(e_i, e_j, [a,b]) \in EP$, such that $a \leq d_g \leq b$.

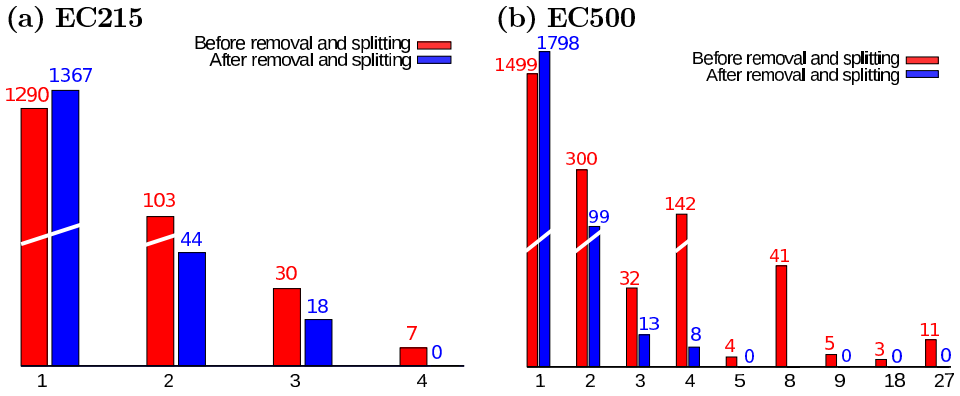



Figure 4.5: The number of pathsets of each size in datasets (a) EC215 and (b) EC500. The red columns column counts initially constructed pathsets. The blue columns count pathsets after phantom path removal and splitting.

### 4.3.2 From edge-pair intervals to pathsets

For each edge-pair interval $(e_1, e_2, [a,b]) \in EP$, we constructed its pathset and further applied phantom path removal and pathset splitting. For EC215 (resp., EC500), we generated 6178 (resp., 18571) pathsets before removing prefix pathsets and 1430 (resp., 2037) pathsets after removing prefix pathsets. Approximately 90% (resp., 74%) of all pathsets that remained represented singletons.

Fig. 4.5 shows the number of pathsets of each multiplicity before (red) and after (blue) phantom path removal and splitting. Before removing phantom paths and splitting, the largest pathset contained only 4 (resp., 27) paths for the EC215 (resp., EC500) dataset. As Fig. 4.5 illustrates, most pathsets with multiple paths turn into singletons after phantom path removal and splitting.

### 4.3.3 Comparing Pathset with other genome assemblers

We compared Pathset to Velvet [95], SOAPdenovo [53], SPAdes [8], and IDBA [65] assemblers using Plantagora [93], an assembly evaluation tool; see Table 4.2. For dataset EC215, Pathset improved on other assemblers in N50, N75, number of misassemblies,[8] and the number of captured complete genes. For dataset EC500, Pathset outperformed the other assemblers in N75 and the number of captured genes. While Velvet had a larger N50 for the EC500 dataset, Velvet's assembly was compromised by the largest number of errors (5). SOAPdenovo produced the most accurate assembly (no assembly errors) but the smallest N50 (57167 as compared to 97971 for Pathset and 105637 for Velvet).

[8] A *misassembly* is formed by concatenation of two sequences $A$ and $B$ that both align to the reference genome but with a gap between them larger than 1000 bases.

## 4.4 Discussion

In this paper, we presented the pathset data structure for assembling genomes using mate-pair data. Instead of using mate-pair transformations on a set of mate-pairs directly, which is computationally expensive and susceptible to failure when the insert size variation is high, we first transform mate-pairs into edge-pairs. We aggregate the distance estimates from all mate-pairs mapping to each pair of edges to make distance estimates more accurate.

As compared with the traditional mate-pair transformation approach, where it is required to have a unique path between paired reads in the de Bruijn graph, our approach stores all suitable paths in a pathset data structure and later uses the paired information to remove phantom paths and further split pathsets. We also introduce the pathset graph, which allows one to construct contigs from the pathsets. Multiple libraries with different insert sizes can be utilized in the pathset data structure. The paired information in different libraries can be used to remove invalid paths in each pathset.

One should note that the pathset algorithms were designed and tested for Illumina paired-end reads with short insert sizes (typically 200–500 bp). Without further developments, the current pathset algorithms will not perform well on Illumina mate-pair (jumping) libraries with insert size in the thousands (typically 2–5 kb). These long libraries, while able to span longer repeats, possess multiple properties that makes it difficult to use the current pathset algorithms: a) high variation in the insert size; b) low coverage; c) high rate of chimeric reads and read pairs. The high variation of insert size together with its long range result in pathsets containing a very large number of paths. The low coverage violates the *representative* property of pathsets. The high rate of chimeric reads and read pairs introduces false paths in the graph. Adapting the pathset algorithm to jumping libraries faces many algorithmic challenges and requires further investigation.

## 4.5 Appendix: Compact representation of pathsets

### 4.5.1 Gapped pathsets

The number of paths of length $d$ between two given edges may be exponential in $d$, so working with pathsets given explicitly may lead to computational difficulties in the case of large insert size, especially in highly repetitive regions. Below, we describe an implicit representation of pathsets to compactly represent such large sets. This has not yet been implemented, but we expect it will prove valuable for dealing with jumping libraries.

Everywhere below, we assume that a directed weighted graph $G(V,E,\ell)$ (where the weighting function $\ell()$ measures the length of edges) is fixed.

A *gapped path* in $G$ is a tuple $p = (e_1, e_2, \ldots, e_n, q)$ where $e_1, e_2, \ldots, e_n \in E$ and $q$ an $(n-1)$-dimensional integer vector. The edges $e_1, e_2, \ldots, e_n$ are called *solid edges* of $p$. A gapped path $p = (e_1, e_2, \ldots, e_n, (q_1, q_2, \ldots, q_{n-1}))$ encodes a pathset consisting of all paths $P$ that pass through the solid edges $e_1, e_2, \ldots, e_n$ in order such that $d_P(e_i, e_{i+1}) = q_i$, $i = 1, 2, \ldots, n-1$. We denote the corresponding pathset by $\textsc{PathSet}(p)$.

A *gapped pathset* is a set of gapped paths. For a gapped pathset $g$, we let $\textsc{PathSet}(g) = \bigcup_{p \in g} \textsc{PathSet}(p)$.

Initially we transform given edge-pairs into gapped pathsets with two solid edges. Namely, an edge-pair $(e_1, e_2, \mathbf{q})$ is transformed into a gapped pathset $\{(e_1, e_2, (q)) \mid q \in \mathbf{q}\}$.

### 4.5.2 Counting paths

For $a, b \in E$ and a nonnegative integer $d$, define $\textsc{NumPaths}(a, b, d)$ as the number of paths of length $d$ starting at $a$ and ending at $b$.

Since we will use this function extensively, for efficiency purposes, we precompute and store its values in a table of size $|V| \times |V| \times d_{\max}$, where $d_{\max}$ is the maximum value of $d$ that we use.

Our dynamic programming algorithm is based on the following formula:

$$\textsc{NumPaths}(a, b, d) = \begin{cases} [a \neq b], & \text{if } d = 0; \\ 0, & \text{if } 0 < d < \ell(a); \\ \sum_{a', \textsc{Start}(a') = \textsc{End}(a)} \textsc{NumPaths}(a', b, d - \ell(a)) & \text{if } d \geq \ell(a). \end{cases}$$

This formula allows one to efficiently fill up the table in $O(|V|^2 \cdot d_{\max})$ time.

We can easily extend $\textsc{NumPaths}()$ to gapped paths:

$$\textsc{NumPaths}((e_1, e_2, \ldots, e_n, (q_1, q_2, \ldots, q_{n-1}))) = \prod_{i=1}^{n-1} \textsc{NumPaths}(e_i, e_{i+1}, q_i)$$

and further to gapped pathsets:

$$\textsc{NumPaths}(g) = \sum_{p \in g} \textsc{NumPaths}(p).$$

In particular, this can be used for detecting empty gapped pathsets (when the number of paths is zero) and gapped pathsets with a unique path (when the number of paths is one).

### 4.5.3 Identifying bridges

An edge $e$ is called a *bridge* for a gapped pathset $g$ if every path in $\textsc{pathset}(g)$ passes through $e$.

We remark that $e$ is a bridge for a gapped path $(e_1, e_2, q)$ if and only if $\textsc{NumPaths}(e_1, e_2, q)$ equals

$$\sum_{j=0}^{q} \textsc{NumPaths}(e_1, e, j) \cdot \textsc{NumPaths}(e, e_2, q-j)$$

which can easily tested. We further can detect that $e$ is a bridge for a gapped path $(e_1, \ldots, e_n, (q_1, q_2, \ldots, q_{n-1}))$ by testing whether $e$ is a bridge for at least one of the gapped subpaths $(e_i, e_{i+1}, q_i)$, $i = 1, 2, \ldots, n-1$.

It is clear that $e$ is a bridge for a gapped pathset $g$ iff $e$ is a bridge for every gapped path in $g$.

### 4.5.4 Prefix testing

To test whether a gapped path $(e_1, e_2, q)$ represents a prefix of a gapped path $(e'_1, e'_2, q')$, we check that $e_1 = e'_2$ and $\textsc{NumPaths}(e_2, e'_2, q' - q) > 0$. This can be further extended to gapped paths with more than two solid edges and gapped pathsets (to be described elsewhere).

### 4.5.5 Finding essential edges

To find essential edges for gapped pathsets in a given set $S$, we first remark that if an edge $e$ is $(e_1, e_2)$-constrained, then $e_1$ represent a bridge in every gapped pathset ending with $e$ and $e_2$ represents a bridge in every gapped pathset starting with $e$. So to determine whether an edge $e$ is essential in some gapped pathset from $S$, we start with searching for such bridges $e_1$ and $e_2$. Then for each possible pair of $(e_1, e_2)$, we check whether it is $d_0$-bounded. In this case, $e$ represents an essential edge for every gapped pathset from $S$ whose elements (i.e., gapped paths) contain $e_1, e_2$ in order as solid edges.

Table 4.1: Comparison of edge-pairs from *E. coli* genome vs. from paired readsin two *E. coli* datasets EC215 and EC500, with insert size 215 and 500, correspondingly.

| Insert size[a] | $\|EP_0\|$[b] | $\|EP\|$[c] | $FPR$[d] | $FNR$[e] |
|---|---|---|---|---|
| 215 | 6321 | 6178 | 0.008 | 0.030 |
| 500 | 18684 | 18571 | 0.017 | 0.023 |

[a]These correspond to *E. coli* datasets EC215 and EC500.

[b]Edge-pairs determined from the reference genome; these are correct by definition.

[c]Edge-pairs determined by mapping mate-pairs to the assembly graph.

[d]The False Positive Rate is the fraction of edge-pair intervals in $EP$ that are incorrect.

[e]The False Negative Rate is the fraction of genomic edge-pairs in $EP_0$ that are missing in $EP$.

### 4.5.6 Splitting pathsets

Assume that we have a set of independent essential edges $A = \{a_1, a_2, \ldots, a_t\}$ in a gapped path $p = (e_1, \ldots, e_n, (q_1, \ldots, q_{n-1}))$. The subset of PATHSET$(p)$ that contains paths passing through $a_1$ is encoded by a gapped pathset $g$ constructed as follows: for every $i = 1, 2, \ldots, n-1$, we find all such $j = 0, 1, \ldots, q_i$ that NUMPATHS$(e_i, a_1, j) \cdot$ NUMPATHS$(a_1, e_{i+1}, q_i - j) > 0$, and add a gapped path

$$(e_1, \ldots, e_i, a_1, e_{i+1}, \ldots, e_n, (q_1, \ldots, q_{i-1}, j, q_i - j, q_{i+1}, \ldots, q_{n-1}))$$

$g$. Gapped pathset representing a subset of PATHSET$(p)$ consisting of pathsets containing $a_i$ $(i = 2, 3, \ldots, t)$ is constructed similarly.

Construction of a gapped pathset representing pathsets not containing any edges from $A$ is to be described elsewhere.

Table 4.2: Comparison of different assemblers. For each column, the best assembler by each criteria is indicated in bold.

| Assembler | # contigs | N50 | N75 | Covered (%)[a] | MA[b] | MM[c] | CG[d] |
|---|---|---|---|---|---|---|---|
| **EC215 dataset:** | | | | | | | |
| Velvet | 198 | 82776 | 42878 | **99.93** | 4 | 1.2 | 4223 |
| SOAPdenovo | 192 | 62512 | 35069 | 97.72 | 1 | 26.1 | 4141 |
| IDBA | 246 | 48825 | 25483 | 99.60 | 3 | 1.3 | 4170 |
| SPAdes | 385 | 86548 | 42441 | 99.53 | 2 | 3.7 | 4223 |
| SPAdes single read | 458 | 54858 | 30309 | 99.56 | **0** | **0.7** | 4239 |
| Pathset | 360 | **91829** | **56830** | 99.56 | 1 | 2.1 | **4249** |
| **EC500 dataset:** | | | | | | | |
| Velvet | 169 | **105637** | 57172 | 99.28 | 5 | 2.5 | 4095 |
| SOAPdenovo | 982 | 57167 | 31582 | **99.88** | **0** | **0.2** | 4196 |
| IDBA | 227 | 57827 | 34421 | 99.26 | 1 | 4.2 | 4158 |
| SPAdes | 215 | 95454 | 46490 | 99.80 | 4 | 1.7 | 4223 |
| SPAdes single read | 493 | 54666 | 35158 | 99.88 | **0** | **0.9** | 4215 |
| Pathset | 320 | 97971 | **58548** | 99.46 | 2 | 2.0 | **4252** |

[a] *Percent of genome covered* is the ratio of total number of aligned bases in the assembly to the genome size.

[b] MA: *Misassemblies* are locations on an assembled contig where the left flanking sequence aligns over 1 kb away from the right flanking sequence on the reference.

[c] MM: Mismatch (substitution) error rate per 100 kbp is measured in the correctly assembled contigs.

[d] *Complete genes* is the number of genes contained completely within assembly contigs (using *E. coli* gene annotations from http://www.ecogene.org).

# Acknowledgements

As members of the SPAdes assembler developers group [8] at the Algorithmic Biology Laboratory (Russian Academy of Science) and UCSD, the authors would like to thank the rest of the group for productive collaboration throughout the SPAdes and Pathset projects. We are especially indebted to Anton Bankevich, Mikhail Dvorkin, Alexey Gurevich, Alexey Pyshkin, Sergey Nikolenko, Sergey Nurk, Nikolay Vyahhi, and Viraj Deshpande for their support of the Pathset project, helpful comments, and thought-provoking discussions. The authors are grateful to Paul Medvedev for his insightful comments.

Chapter 4, in part, is a reprint of the material as it appears in Son Pham, Dmitry Antipov, Alexander Sirotkin, Glenn Tesler, Pavel Pevzner, and Max Alekseyev, “Pathset Graphs: A Novel Approach for Comprehensive Utilization of Mate-Pairs in Genome Assembly”. RECOMB 2012, pp. 200-212. The dissertation author was the primary investigator and author of this paper.

# Chapter 5

# DRIMM-Synteny: decomposing genomes into evolutionary conserved segments

## 5.1 Introduction

The evidence in favor of the Whole Genome Duplication (WGD) in *S. cerevisiae* was discovered by 92 but was heavily contested (e.g., see 52). 50 and 34 used preduplicated genomes of *K. waltii* and *A. gossypii* to settle this controversy (see 56 for a recent study contesting the WGD in yeast). Starting from [92] and [80] all WGD studies essentially amounted to constructing synteny blocks of certain type (e.g., *sister blocks* in 80 or *doubly conserved synteny (DCS)* blocks in 50) and demonstrating that these blocks cover a large portion of the genome. Remarkably, there is still no general purpose tool that would automate such analysis and reduce various WGD studies to simply computing a "duplicativity coverage" of the genome. For example, software from [50] is not applicable to finding the synteny blocks from [80] and vice versa. Indeed, most WGD studies [87, 50, 34, 45, 55, 29, 5, 78] developed new software for WGD analysis instead of using some previously developed tools!

We argue that the lack of tools for automated WGD analysis is the result of the lack of tools for *synteny block* identification in highly duplicated genomes. Many genomes have undergone extensive duplications followed by gene losses and rearrangements, making decoding of genomic architecture (synteny block reconstruction) in such genomes difficult. For example, duplications account for $\approx 70\%$ of the *Arabidopsis thaliana* genome [13] making synteny block reconstruction in this and other plant genomes challenging. Fig. 5.1(a) shows a highly duplicated "genome" $G$ along with its decomposition into *overlapping* (left) and *non-overlapping* (right) synteny blocks. The non-overlapping decompositions are more desirable since they are required for the follow-up rearrangement and duplication studies (e.g., the existing genome rearrangement algorithms are unable to analyze overlapping decompositions). However, con-

structing non-overlapping decompositions is more difficult than constructing overlapping decompositions. While it may appear that one can simply sub-partition the overlapping blocks into the non-overlapping ones, 46 and 64 explained that this partitioning does not work for complex genomes.

77 proposed the first algorithm for synteny block generation, which was aimed at comparative mapping data and did not take into account *micro-rearrangements*. The first algorithms for synteny block reconstruction in sequenced genomes (GRIMM-Synteny [69] and Chains-and-Nets [51]) were developed in 2003 when thousands of micro-rearrangements in mammalian genomes were discovered. These and many other synteny block generation algorithms [38, 18, 21, 31, 30, 16, 86, 33, 54] proved to be adequate for small sets of genomes but did not address issues that stem from extensive duplications and deletions. Most previous efforts to generate synteny blocks for highly-duplicated genomes [88, 17, 14, 50, 39, 40, 85, 81] generated overlapping rather than non-overlapping blocks. In contrast, some recently developed tools (e.g., Enredo tool [63] used in Ensembl [43]) aim to generate non-overlapping synteny blocks. The non-overlapping representation has advantages over the traditional pairwise (and overlapping) representation of duplications. Indeed, the pairwise representation (that dominated previous studies of human segmental duplications) left the question of finding ancestral *duplicons* in the human genome unanswered [6], while the non-overlapping representation constructed in [46] resolved it. Also, an overlapping representation can be easily obtained from a non-overlapping representation but not vice versa.

Fig. 5.2 shows an Human-Chimpanzee-Macaque-Rat-Mouse-Opossum-Cow synteny block and illustrates the challenge of constructing synteny blocks in multiple genomes. As the number of analyzed genomes increases, the number of shared genes may substantially decrease. While this block contains 29 genes, only 2 of them are shared between all 7 species. The existing synteny block generation algorithms (like GRIMM-Synteny) are likely to miss such a block with only 2 shared genes or discard it as statistically insignificant.

64 noticed that the problem of constructing non-overlapping synteny blocks is similar to the difficult problem of *de novo* repeat classification [10]. 66 introduced the *A-Bruijn graph* approach to repeat classification, representing all repeats as a mosaic of non-overlapping sub-repeats. Later, the A-Bruijn graphs were found to be useful in diverse applications such as multiple alignment [75], *de novo* protein sequencing [7], analysis of segmental duplications [46], and next generation DNA sequencing [25, 19, 95].

While diverse applications of A-Bruijn graphs use the same algorithmic idea, each application has unique features that need to be addressed for a new research domain. The original A-Bruijn graph

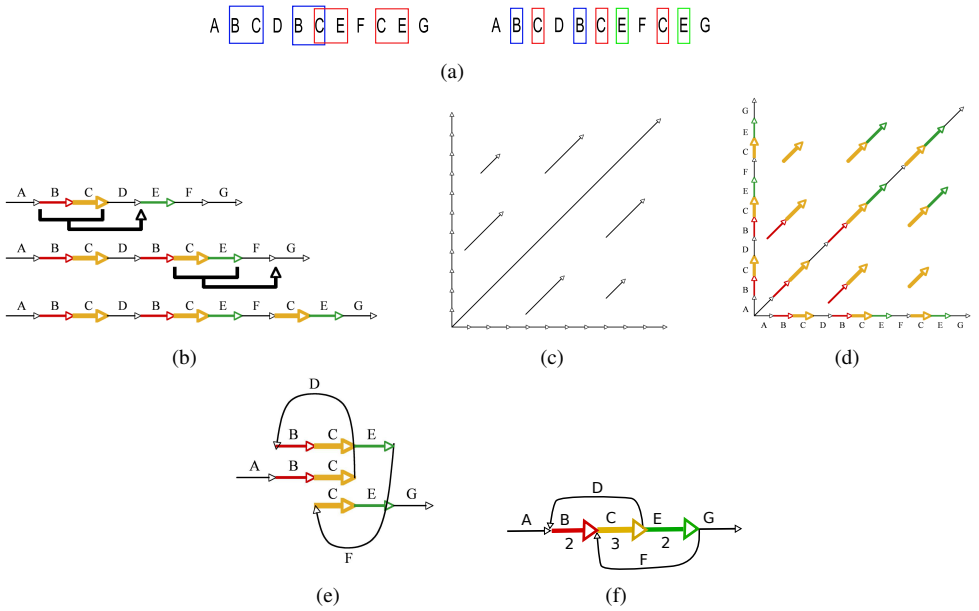



Figure 5.1: (a) Decomposition of a "genome" into overlapping and non-overlapping synteny blocks. A highly duplicated "genome" (b) and its genomic dot-plot (c). (d) Since the diagonals in 2-D representations overlap in 1-D representation, one has to subpartition them into red, yellow, and green sub-diagonals to avoid overlaps. (e) Generating A-Bruijn graph. (f) A-Bruijn graph reveals synteny blocks B, E (each with two copies) and C with three copies. (While this represents anchors as directed edges, all other figures in this paper represent anchors as vertices. We found that the vertex representation of anchors does not significantly affect our results while significantly simplifying the presentation of DRIMM-Synteny.)

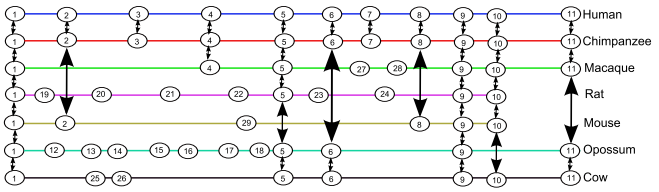



Figure 5.2: A Human-Chimpanzee-Macaque-Rat-Mouse-Opossum-Cow synteny block (in Human chromosome 1) contains 29 genes with only 2 of them shared between all 7 species. While 17 of these red 29 genes appear to be present only in a single genome (like gene 27 in macaque), most of these 17 genes have orthologs in other species (these orthologs are not shown since they are located within other synteny blocks in other species).

approach [66] involves some heuristics that may or may not work for a particular application. For example, the *bulge removal* heuristic was originally designed for fragment assembly of Sanger reads but turned out to work well in various tools for next generation DNA sequencing [95, 19, 25], mass spectrometry [7] and synteny block reconstruction [63, 64]. Another important heuristic that is application specific is the *threading* procedure from [66] that reconstructs how the genome traverses the transformed A-Bruijn graph. While threading was never problematic in sequencing applications, 64 came to the conclusion that it is a major bottleneck in synteny block reconstruction and wrote: *"Optimizing the A-Bruijn graph approach for synteny block generation represents the next challenge in analyzing the genomic architectures."* Our paper addresses this problem by devising the first A-Bruijn graph approach that does not require a threading step and substitutes it with an alternative *genome modification* step implemented in the DRIMM-Synteny (Duplications and Rearrangements In Multiple Mammals) software ( http://bix.ucsd.edu/projects/drimm/). We illustrate applications of DRIMM-Synteny to analyzing yeast, plant, and mammalian genomes and further combine it with rearrangement analysis to reconstruct the ancestral preduplicated yeast genome.

## 5.2 Methods

**Preliminaries**. A typical synteny block generation algorithm takes as an input a set of *anchors* (e.g., local alignments or pairs of similar genes) between two genomes and constructs a set of synteny blocks that cover (without overlaps) most of each genome. As a result, each genome is represented as a shuffled sequence of the synteny blocks. For two genomes, most synteny blocks generation algorithms employ a 2-dimensional genomic dot-plot where two genomes are placed along the axes on the plane and their anchors are represented as dots (Fig. S1(a)). These algorithms further decompose the dot-plot into long diagonal-like segments constituting 2-D synteny blocks. The conventional (1-D) synteny blocks for each genome can be obtained as projections of the 2-D synteny blocks onto a corresponding axis (Fig. S1(b)).

Fig. 5.1(b), 5.1(c) shows a highly duplicated "genome" and its genomic dot-plot. The diagonals in Fig. 5.1(c) are what conventional synteny block reconstruction methods would produce as synteny blocks from the genomic dot-plot of a genome against itself. Since these 2-D blocks overlap along the sequence (in 1-D), the duplication structure is unclear. Ideally, we would like to see diagonal segments that do not overlap along the sequence (Fig. 5.1(d)). The non-overlapping segments are revealed by the A-Bruijn graph (Fig. 5.1(e), 5.1(f)) approach described in [66].

Let $S = (s_1, s_2, \ldots, s_n)$ be a sequence of genes in a genome represented as an undirected path

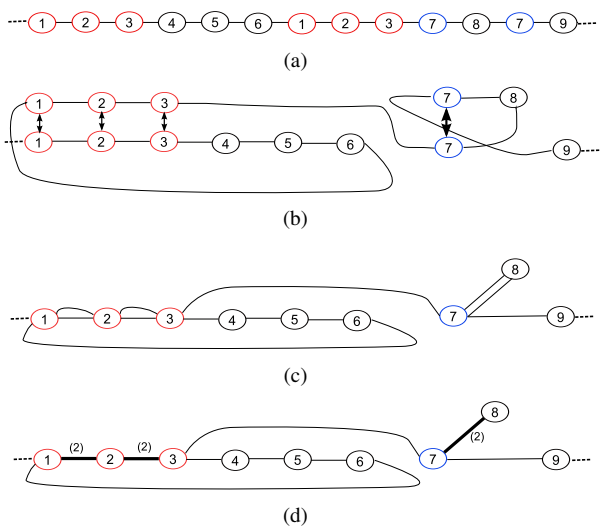



Figure 5.3: (a) A genome $S = (1,2,3,4,5,6,1,2,3,7,8,7,9)$ with 13 genes (9 unique genes) represented as a path. (b) Constructing the A-Bruijn graph by gluing vertices with the same labels. (c) The A-Bruijn graph of genome $S$. (d) The weighted A-Bruijn graph with edge multiplicities shown.

(Fig. 5.3(a)) and let $m$ be the number of *unique* genes in $S$ ($S$ may have repeated genes). While this paper considers genes as anchors, DRIMM-Synteny is applicable to any anchors representing arbitrary regions of similarity. An *A-Bruijn graph* $AB(S)$ is obtained by "gluing" identically labeled vertices of the path $S$ as shown in Fig. 5.3(c) (see 66 for the precise definition of "gluing"). We remark that the A-Bruijn graphs are *Eulerian*, i.e., there exists a path in these graphs visiting every edge exactly once. The A-Bruijn graph can be viewed as both an undirected *multi-graph* (adjacent vertices can be connected by multiple edges) and a *weighted* graph with the multiplicity of an edge $(v,w)$ defined as the number of times genes $v$ and $w$ are consecutive in $S$.

A set of the perfectly repeated regions in $S$ corresponds to a path in the $A$-Bruijn graph (e.g., the path [1,2,3] in Fig. 5.3(d)). The perfectly repeated regions that do not share genes with other regions in $S$ correspond to *non-branching* paths (maximal paths in the graph satisfying the condition that all their internal vertices have only two neighboring vertices), with the multiplicities equal to the number of times these regions appear in the sequence $S$. In the case of the synteny blocks, however, small differences between multiple instances of the same synteny block generate short cycles in the A-Bruijn graphs, while the spurious similarities between different synteny blocks (called *microblocks*) break long non-branching paths into multiple shorter sub-paths. Moreover, short palindromic regions within the conserved blocks generate the so-called *thorns* (like path [7,8,7] in Fig. 5.3(c)). These short cycles, microblocks, and thorns hide the underlying synteny blocks in genomes and make the synteny block generation difficult.

**Synteny blocks in multiple and/or highly duplicated genomes**. From an algorithmic perspective, finding (i) synteny blocks between multiple genomes and (ii) synteny blocks within a single genome are similar problems since (i) can be reduced to (ii) by concatenating (with delimiters) the multiple genomes into a single genome. This illustrates the challenge one faces while reconstructing synteny blocks in *multiple* mammalian genomes that are traditionally viewed as an "easy target" (compared to plant genomes) for synteny block analysis: while duplications account for less than 7% of mammalian genomes, the concatenation of mammalian genomes represents a highly duplicated virtual genome that rivals the complexity of plant genomic architectures. 15 faced this problem while constructing the human-mouse-rat synteny blocks. While their approach (based on anchors shared between *all* genomes) worked for a small number of genomes, it is unsustainable since the number of such anchors decreases with the increase in the number of genomes.

Given a set of chromosomes, one can concatenate them and construct the A-Bruijn graph of the resulting concatenation. Applying this procedure to genomes of *S. cerevisiae* (16 chromosomes with 5616 genes, 5057 are unique) and *K. waltii* (8 chromosomes with 5070 unique genes) results in a complex graph with 6240 vertices and 8976 edges. Fig. 5.4(b) represents a subgraph of this A-Bruijn graph corresponding to a *Doubly Conserved Synteny (DCS) block* [50]. This DCS is formed by a pair of regions:

... **1**, 4, **7**, **12**, **15**, 16, **17**, **19**, 21, ...

... **1**, 3, 22, 5, **7**, 9, 23, **12**, 13, 24, 25, **15**, **17**, **19**, 26, ...

in *S. cerevisiae* and a single region

... **1**, 2, 3, 4, 5, 6, **7**, 8, 9, 10, 11, **12**, 13, 14, **15**, 16, **17**, **19**, 20, ...

in *K. waltii* (six genes shared by all 3 regions are shown in bold). Fig. 5.4(b) reveals many short cycles that "hide" these three syntenic regions. Below we propose a *Sequence Modification* algorithm that transforms the original genome (with cryptic) synteny blocks into a slightly different genome with the well defined synteny blocks. The key idea is to make small changes to the sequence $S$, so that its corresponding A-Bruijn graph is simplified. In contrast, the previous A-Bruijn graph approaches simplify the A-Bruijn graph $AB(S)$ without changing the sequence $S$ and thus faced a difficult challenge of threading $S$ through the simplified graph. The Sequence Modification algorithm transforms the subgraph in Fig. 5.4(b) into a subgraph in Fig. 5.4(c) and transforms each of the 3 (varying) instances of the doubly conserved synteny block into 3 (non-varying) instances ..., **1**, 3, 22, 5, 6, **7**, 9, 23, **12**, 13, 24, 25, **15**, 16, **17**, 18, **19**,....

**Genome Threading Problem**. A cycle in a graph is *short* if it has fewer than *girth* edges, where *girth* is a parameter. Short cycles often aggregate into complex networks and "hide" the underlying structure

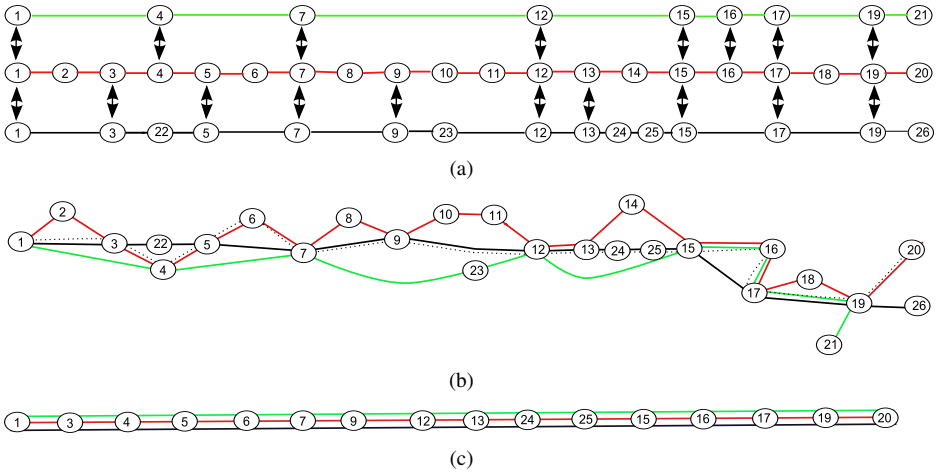



Figure 5.4: (a) A doubly conserved synteny block in *K. waltii* and *S. cerevisiae* where one region in *K. waltii* genome (red) corresponds to 2 regions in *S. cerevisiae* (green and black). (b) An induced subgraph of the A-Bruijn graph corresponding to a DCS block (a) in the genomes of *S. cerevisiae* and *K. waltii* contains many short cycles. (c) The sequence modification algorithm reveals the synteny block as a non-branching path.

of the A-Bruijn graphs. To reveal this hidden structure, 66 formulated the *Maximum Subgraph with Large Girth (MSLG)* problem, which aims to find the maximum weight subgraph of the A-Bruijn graph that does not contain short cycles. 66 proposed constructing the *Maximum Spanning Tree (MST)* as the first step towards finding MSLG, followed by extending MST into an approximate solution (called the *simplified A-Bruijn graph* and denoted $MSLG(S)$) of the MSLG problem, and finally, the genome *threading* procedure. We remark that while the multigraph $AB(S)$ is Eulerian and the genome sequence $S$ represents an Eulerian path in $AB(S)$, $MSLG(S)$ is typically non-Eulerian. The goal of threading is to find a *Chinese Postman* [84] path in $MSLG(S)$ that "mimics" $S$. While the threading heuristic from [66] worked well for fragment assembly, 64 commented that it deteriorates for synteny block construction when missing genes and micro-rearrangements are common.

Therefore, the key complication in the synteny block reconstruction is that, in difference from the (Eulerian) A-Bruijn graph, the simplified A-Bruijn graph is not Eulerian. Since the MSLG algorithm from [66] breaks the Eulerian path into multiple segments, threading the original sequence through the simplified graph, in some cases, becomes impossible. This motivates the *Sequence Modification Problem* (*SMP*) defined below.

**Sequence Modification Problem**. Since the A-Bruijn graphs of real genomes have many short cycles (hiding synteny blocks), the goal of the synteny block reconstruction is to reveal the "hidden" synteny blocks by removing these short cycles. In an A-Bruijn graph without short cycles, synteny blocks are

defined as the non-branching paths in the graph with multiplicity larger than 1. The number of times a synteny block appears in the sequence is the multiplicity of the corresponding non-branching path.

Let $d(S,S')$ be the minimum number of edit operations (e.g. insertions/deletions/substitutions of letters or short substrings) to transform a string $S$ into a string $S'$. We define the Sequence Modification Problem as follows:

Sequence Modification Problem (SMP): Given a string $S$ and a parameter $girth$, find a string $S'$ with minimum $d(S,S')$ among all strings such that $AB(S')$ has no cycles shorter than $girth$.

Since the complexity status of SMP is unknown, we propose a greedy algorithm that produces good results in practice. In brief, the algorithm finds short cycles in $AB(S)$ and further changes $S$ into $S'$ with the goal to *eliminate* short cycles from $AB(S')$. Before describing the strategy for eliminating the short cycles, we classify all cycles in the A-Bruijn graphs into *two-way*, *one-way*, and *composite* cycles.

A cycle $C$ in $AB(S)$ is *formed* by paths $P_1$ and $P_2$ ($P_1$ and $P_2$ are non-overlapping substrings of $S$) if the edge set of $C$ is the union of the edge sets of $P_1$ and $P_2$. A cycle $C$ is called a two-way cycle if it is formed by paths $P_1$ and $P_2$. For example, in Fig. 5.5(a), a two-way cycle on vertices $(1,2,3)$ in the A-Bruijn graph of $S = (\ldots 1,2,3,4,\ldots,1,3,4,\ldots)$, is formed by paths $P_1 = [1,2,3]$ (consisting of two edges) and $P_2 = [1,3]$ (consisting of a single edge).

A cycle $C$ in $AB(S)$ is called a one-way cycle if it is formed by a single path (substring) $P$ of sequence $S$ (i.e., the edge sets of $C$ and $P$ are the same). In Fig. 5.5(b), the tandem repeat [2,3,4,2,3,4] corresponds to a one-way cycle (2,3,4). We also define composite cycles as cycles that are formed by more than 2 paths/substrings (Fig. 5.5(c)).

In practice, the cycles in the A-Bruijn graphs are typically classified in only one of three categories above. However, some cycles are classified into multiple categories, for example, a cycle can be both a one-way cycle and a two-way cycle.

**Cycle rerouting**. Let $C$ be a two-way cycle formed by paths $P_1$ and $P_2$. The string $S$ may contain multiple instances of substrings $P_1$ and $P_2$, with the corresponding multiplicities $n_1 \leq n_2$. $(P_1,P_2)$-*transformation* of $S$ (called DETOUR) is a substitution of all instances of $P_1$ in $S$ by $P_2$. $(P_1,P_2)$-transformation has a simple interpretation: the Eulerian path switches from traversing $P_1$ to traversing $P_2$, thus eliminating an instance of a cycle $C$ from the A-Bruijn graph (Fig. 5.6(a)). We choose $n_1$ substitutions of $P_1$ by $P_2$ (rather than $n_2$ substitutions of $P_2$ by $P_1$) to minimize the number of segmental substitutions in the Sequence Modification algorithm.

Let $C$ be a one-way cycle formed by a path $P = (v_{in},\ldots,u,v_{in},\ldots,v_{out})$, where $v_{in}$ and $v_{out}$ are

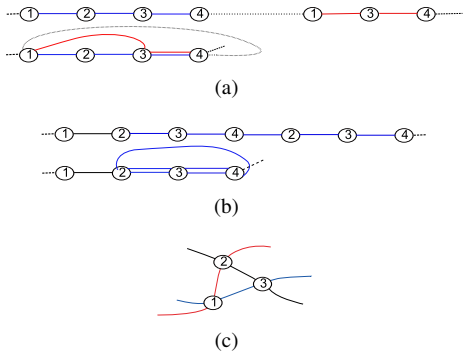



Figure 5.5: (a) A two-way cycle (1,2,3) caused by a small difference between the syntenic regions [1,2,3,4] and [1,3,4]. (b) A one-way cycle (2,3,4) formed by a tandem repeat [2,3,4,2,3,4]. (c) A composite cycle formed by 3 paths: [1,3], [3,2], and [2,1] that share some genes.

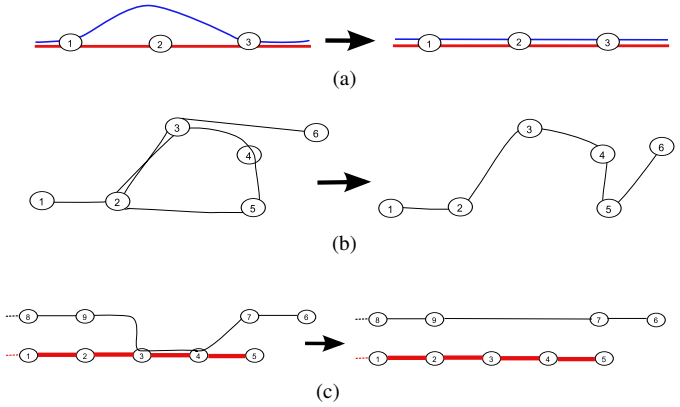



Figure 5.6: (a) Detour defined by a two-way cycle (1,2,3) (that is formed by paths [1,2,3] and [1,3]) eliminates the cycle. (b) Shortcut of a one-way cycle (2,3,4,5) formed by a path [2,3,4,5,2,3] eliminates the cycle. (c) Path splitting eliminates spurious similarities.

the first and the last vertices of $P$. *P-transformation* (called SHORTCUT) substitutes every instance of path $P$ by a shorter path $(v_{in}, \ldots, u)$ (Fig. 5.6(b)).

The REROUTE procedure (Fig. S4) iterates detours and shortcuts on a cycle $C$ until the cycle is eliminated or is neither a two-way nor one-way cycle. DRIMM-Synteny does not have a specific subroutine that removes composite cycle. However, in most cases, composite cycles are removed by the cycle rerouting procedure (on different cycles) or the splitting procedure described below.

**Processing microblocks and thorns**. After REROUTE, the A-Bruijn graph may still be complex. Spurious similarities between different synteny blocks form *microblocks* (non-branching paths shorter than a threshold *pathLength*) and the palindrome-like substrings in the genomic sequences form *thorns* (like path [7,8,7] in Fig. 5.3(c)). Both microblocks and thorns break long synteny blocks into shorter blocks and need to be processed to avoid unnecessary synteny blocks miniaturization.

GRIMM-Synteny [69] simply removes microblocks (defined as "small" synteny blocks) and may occasionally "destroy" biological synteny blocks formed by multiple microblocks. DRIMM-Synteny instead *splits* blocks that share a microblock (Fig. 5.6(c)). The palindrome-like substrings in $S$ form non-branching paths called thorns. Long palindromes are valuable synteny blocks while short ones form thorns that break long synteny blocks into shorter ones. We process short thorns (shorter than *thornLength*) by finding all short palindromes and removing the second halves of these palindromes. Similarly, tandem repeats are reported as synteny blocks (of multiplicity 2) if they from long cycles.

**Identification of syntenic regions: an alternative to genome threading**. DRIMM-Synteny (Fig. 5.7) is an approximation algorithm for the Sequence Modification Problem that first finds a maximum spanning tree $T$ of the graph $AB(S)$ and iteratively analyzes all edges that are not present in $T$ (*outside edges*). We limit our attention to the outside edges forming short cycles, identify a shortest cycle containing an outside edge, and further change $S$ into $S'$ with the goal to *eliminate* this cycle from $AB(S')$. Application of DRIMM-Synteny to the graph in Fig. 5.4(b) results in a simple graph in Fig. 5.4(c) that reveals the DCS block. We remark that while any spanning tree (rather than MST) would work for detecting short cycles, DRIMM-Synteny selects MST since it proved to work well in other applications of A-Bruijn graphs. DRIMM-Synteny is fast in practice, taking less than a minute even for the largest dataset we analyzed ($\approx$ 20000 genes per each of seven mammalian genomes).

DRIMM-Synteny transforms the original sequence $S$ (genome) into a new sequence $S'$ with well-defined synteny blocks (each synteny block in $S'$ corresponds to a non-branching path in $AB(S')$). The only remaining task is to identify the *positions* of all synteny blocks in the original sequence $S$. If we assume

```
DRIMM-SYNTENY (sequence S, cycle length threshold girth, path length
      threshold pathLength, thorn length threshold thornLength)

   Construct the A-Bruijn graph AB(S)
   Find the Maximum Spanning Tree MST(S) in AB(S)
   for each edge e outside MST(S) (in increasing order of multiplicities)
      Identify the shortest one or two-way cycle C that contains edge e
      if (|C| ≤ girth)
         REROUTE(S, C)
   Process palindromes in S that are shorter than thornLength
   Process microblocks (nonbranching paths shorter than pathLength)
   Return all non-branching paths in AB(S) as synteny blocks
```

Figure 5.7: The pseudo-code of DRIMM-Synteny algorithm. The last *color propagation* step is not shown. See section 2 of the Supplement for details of the algorithm.

that each synteny block in the modified sequence $S'$ is painted with its own color, then the problem is to transform colors from $S'$ back to $S$. While the threading step from [66] often results in poor-quality synteny block reconstruction [64], our sequence modification approach bypasses the genome threading step.

## 5.3 Results

**Datasets and parameters**. The yeast gene orders were extracted from [50] and [78]. The mammalian gene orders were generated using MSOAR program [37]. The gene order of *Arabidopsis thaliana* was extracted from [17].

Although every synteny block reconstruction algorithm is parameters independent, we are not aware of tools for automatic derivation of the optimal parameters. The parameters' choice for these tools (and DRIMM-Synteny) relies on an expert analysis. In this paper, we use the default parameters ($girth = 20, pathLength = 3, thornLength = 3$) for all datasets.

**Synteny blocks in 7 mammalian genomes**. To benchmark DRIMM-Synteny on multiple (but not highly duplicated) genomes, we analyzed 7 mammalian genomes: Human (H), Chimpanzee (C), Macaque (Q), Rat (R), Mouse (M), Opossum (O) and Cow (W). As the number of genomes increases, the number of genes that are shared between all genomes decreases and methods relying on the genes shared by all genomes (e.g., GRIMM-Synteny) deteriorate. Fig. 5.2 shows a 7-way synteny block that would most likely be missed by such tools.

The concatenation of 7 mammalian genomes results in a virtual genome with 144149 genes (53245 unique genes). The simplified A-Bruijn graph of this concatenation (with the default parameters) has 31282 vertices and 35773 edges. Substituting non-branching paths in this graph by single edges

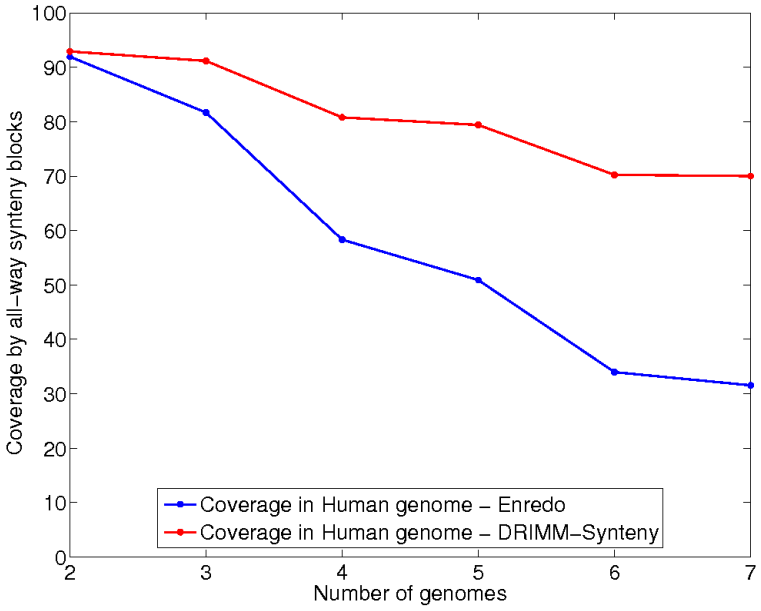
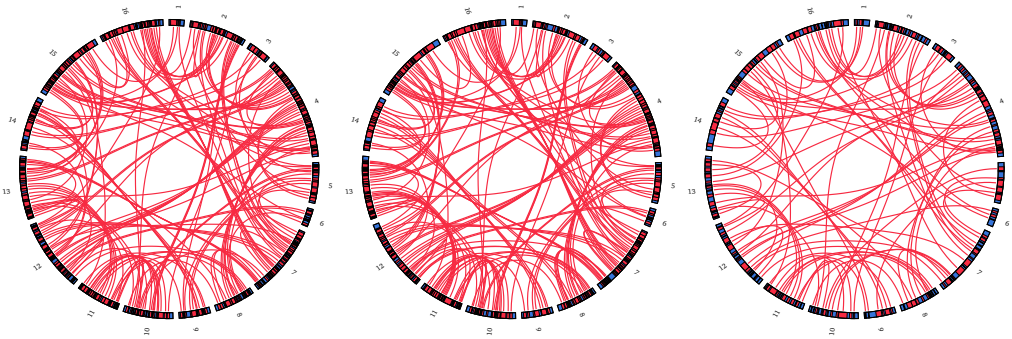



Figure 5.8: Coverage of Human genome by $k$-way synteny blocks for $2 \leq k \leq 7$. While Enredo [63] and DRIMM-Synteny produce blocks with similar coverage for small $k$, the $k$-way synteny blocks generated by Enredo have lower coverage for larger $k$.

results in a graph on 2212 vertices and 3514 edges. DRIMM-Synteny still finds many synteny blocks with good coverage ($\approx 70\%$) in this highly duplicated virtual genome. Enredo [63], an advanced synteny block generation tool used in Ensembl [43], generated 7-way blocks with a significantly lower coverage ($\approx 32\%$, table S1 b).

To further compare Enredo [63] and DRIMM-Synteny, we ran both program on the dataset initially containing only Human and Chimpanzee genomes where these tools generated nearly identical results. Then at each step, we added one more genome to the dataset, generated $k$-way synteny blocks ($k$ is the number of genomes), computed the genome coverage by these blocks and repeated the process for $k = 3, \ldots, 7$. Fig. 5.8 shows that, as more genomes are added to the dataset, DRIMM-Synteny continues generating synteny blocks with high coverage ($\approx 70\%$ for 7-way blocks), while the 7-way synteny blocks generated by Enredo cover only $\approx 32\%$ of the genome.

**Synteny blocks in *K. waltii and S. cerevisiae*** . The concatenation of *S. cerevisiae* (S) and *K. waltii* (K) results in a genome with 10686 genes (6240 unique genes). The simplified A-Bruijn graph of this concatenation has 5844 vertices and 6221 edges. Substituting non-branching paths in this graph by single edges results in a graph on 653 vertices and 997 edges. DRIMM-Synteny finds nearly all doubly conserved



(a) (b) (c)

Figure 5.9: Positions of DCS blocks in *S. cerevisiae*. Sixteen chromosomes of *S. cerevisiae* are presented on the circle. Each arc joins two segments in *S. cerevisiae* forming a DCS block. (a): 152 DCS blocks for *K. waltii* and *S. cerevisiae* from [50], (b): 151 DCS blocks generated by DRIMM-Synteny for *K. waltii* and *S. cerevisiae* (with the default parameters). (c): 87 DCS blocks generated by DRIMM-Synteny for *S. cerevisiae alone*.

synteny blocks identified in [50] as well as 231 singly conserved synteny blocks (Table 5.1).

Table 5.1: Synteny blocks of *K. waltii (K)* and *S. cerevisiae (S)*. K-S-S blocks represent blocks of multiplicity 3 that have one instance in *K waltii* and two instances in *S. cerevisiae* (DCS blocks from [50]). K-S blocks represent blocks of multiplicity 2 that have one instance in *K waltii* and one instances in *S. cerevisiae*. The average size of the K-S-S and K-S blocks is 18 and 8 genes respectively (before removing short blocks).

| Mult. | Type | # of blocks | Span on *K* | Span on *S* |
|---|---|---|---|---|
| 3 | K-S-S | 246 | 77% | 78% |
| 2 | K-S | 231 | 13% | 10% |

Since most studies of genomic architectures ignore very short synteny blocks, we delete all short synteny blocks (with fewer than $\Delta$ genes in each species) in an iterative fashion as described in [3].

Fig. 5.9(a,b) shows the position of DCS blocks (on *S. cerevisiae* genome) generated by DRIMM-Synteny and in [50] and illustrates that they produced nearly identical results. The statistics of synteny blocks generated by DRIMM-Synteny is given in Table 5.1. Enredo [63], on the other hand, missed many synteny blocks found in [50]. This raises a question why a rather sophisticated Enredo algorithm failed to reveal synteny blocks constructed using a simple approach from [50]. We emphasize that Enredo is a general synteny block generation tool while the approach in [50] has many limitations: it is only applicable to pairs of pre-WGD and after-WGD genomes with small number of additional segmental duplications.

We also ran DRIMM-Synteny on the *S. cerevisiae* genome *alone* with the default parameters. DRIMM-Synteny generated 87 synteny blocks (Fig. 5.9(c)), which cover about 51% the genome and

reveal a pattern similar to the one shown in Fig. 5.9(a,b). This result is consistent with the analysis in [80] (84 blocks with $\approx 50\%$ coverage). While 80 indeed revealed sister blocks in *S. cerevisiae*, we are not aware of any (general purpose) synteny block generation tool that can automatically construct synteny blocks in highly duplicated genomes. As Fig. 5.9 illustrates, if such a tool was available in 2004 when [50] was published, it would provide a solid evidence for WGD in *S. cerevisiae* even without additional analysis in [50]. Moreover, the analysis in [50] would be largely reduced to merely running DRIMM-Synteny.

**How many Whole Genome Duplications have shaped evolution of *Arabidopsis thaliana*?** Although the *Arabidopsis thaliana* genome has been shaped by large duplications, the number and extent of these duplications have been controversial [76, 91]. On the one hand, *Arabidopsis'* genomic architecture may be explained by multiple independent segmental duplications. On the other hand, it may originate from a single WGD (or a few rounds of WGDs) followed by genomic rearrangements that split up the original duplicated sequences. The initial *Arabidopsis* studies hypothesized that its ancestor underwent a single WGD [4, 13]. However, 89 argued that *Arabidopsis thaliana* underwent multiple segmental duplications at different times (rather than WGD). 14 (see also 17) refuted [89], confirmed WGD and further found evidence for a second older WGD that has been partly obscured by other segmental duplications. A good way to resolve this controversy would be to construct synteny blocks and to analyze coverage by blocks of multiplicity larger than 2. However, to the best of our knowledge, the high-coverage non-overlapping decompositions of *Arabidopsis thaliana* into synteny blocks has not been constructed yet.

The genome of *Arabidopsis thaliana* contains 28170 genes (23129 unique genes). The simplified A-Bruijn graph of this genome has 20288 vertices and 21486 edges. Substituting non-branching paths in this graph by single edges results in a graph on 782 vertices and 1224 edges. We further remove short (and potentially spurious) synteny blocks (Table S2). While the synteny blocks with multiplicity 2 (supporting one round WGD) span 50% of the genome, the synteny blocks of multiplicity 4 (supporting evidence for two rounds of WGDs) cover only 8% of the genome. If 50% coverage by 2-way blocks in *S. cerevisiae* established by 80 was criticized as a proof of WGD in yeast (and required an additional study [50] to establish WGD), why 8% coverage by 4-way synteny blocks is a definite proof of two rounds of WGD in *Arabidopsis*. If one counts both 3-way and 4-way synteny blocks, the coverage increases to 16% but in retrospect (see 80, 50) it remains unclear why 16% coverage represents a definite proof of two rounds of WGD.

## 5.4 Discussion

The rapidly increasing set of sequenced genomes highlights the importance of identifying the synteny blocks in multiple and/or highly duplicated genomes. As the number of analyzed genomes increases, the number of shared genes may decrease substantially. The synteny block generation algorithms based on pairwise comparisons are often limited, since in some cases, the synteny blocks can only be reconstructed by multi-way comparison. We proposed the DRIMM-Synteny algorithm for identifying the non-overlapping synteny blocks and bypassed the difficult threading problem (a bottleneck in [64]) by developing a new A-Bruijn graph approach for solving the Sequence Modification Problem.

## 5.5 Acknowledgements

We are indebted to Glenn Tesler for many helpful suggestions that significantly improved the paper and to Javier Herrero for help with analyzing Enredo's results. We also thank Max Alekseyev, Rustem Aydagulov, Vladimir Dobrynin for their helpful comments. We are grateful to Max Alekseyev, Kevin Byrne, Tao Jiang, and Qian Peng for help with generating gene orders of yeast, plant, and mammalian genomes.

Chapter 5, in part, is a reprint of the material as it appears in Son Pham and Pavel Pevzner, DRIMM-Synteny, "Decomposing Genomes into Evolutionary Conserved Segments". Bioinformatics 2010, pp. 2509-2516. The dissertation author was the primary investigator and author of this paper.

# Bibliography


[1] T. Aardenne-Ehrenfest and N.G. Bruijn. Circuits and trees in oriented linear graphs. *Simon Stevin*, pages 203–217, 1951.

[2] J. Abrham and A. Kotzig. Transformations of euler tours. *Annals of Discrete Mathematics*, 8:65–69, 1980.

[3] Max A Alekseyev and Pavel A Pevzner. Breakpoint graphs and ancestral genome reconstructions. *Genome research*, 19(5):943–957, 2009.

[4] Genome Initiative Arabidopsis. Analysis of the genome sequence of the flowering plant arabidopsis thaliana. *Nature*, 408(6814):796, 2000.

[5] Jean-Marc Aury, Olivier Jaillon, Laurent Duret, Benjamin Noel, Claire Jubin, Betina M Porcel, Béatrice Ségurens, Vincent Daubin, Véronique Anthouard, Nathalie Aiach, et al. Global trends of whole-genome duplications revealed by the ciliate paramecium tetraurelia. *Nature*, 444(7116):171–178, 2006.

[6] Jeffrey A Bailey, Amy M Yavor, Hillary F Massa, Barbara J Trask, and Evan E Eichler. Segmental duplications: organization and impact within the current human genome project assembly. *Genome Research*, 11(6):1005–1017, 2001.

[7] N. Bandeira, K.R. Clauser, and P.A. Pevzner. Shotgun Protein Sequencing. *Molecular & Cellular Proteomics*, 6(7):1123, 2007.

[8] A. Bankevich, S. Nurk, D. Antipov, A. Gurevich, M. Dvorkin, A. Kulikov, V. Lesin, S. Nikolenko, S. Pham, A. Prjibelski, A. Pyshkin, A. Sirotkin, N. Vyahhi, G. Tesler, M. Alekseyev, and P. Pevzner. SPAdes: a New Genome Assembler and its Applications to Single Cell Sequencing, 2012. Journal of Computational Biology — to appear (May 2012).

[9] A. Bankevich, S. Nurk, D. Antipov, A. Gurevich, M. Dvorkin, A. Kulikov, V. Lesin, S. Nikolenko, S. Pham, A. Prjibelski, A. Pyshkin, A. Sirotkin, N. Vyahhi, G. Tesler, M. Alekseyev, and P. Pevzner. Spades: A new genome assembly algorithm and its applications to single-cell sequencing. *Journal of Computational Biology*, 19(5):455–477, 2012.

[10] Zhirong Bao and Sean R Eddy. Automated de novo identification of repeat sequence families in sequenced genomes. *Genome Research*, 12(8):1269–1276, 2002.

[11] Serafim Batzoglou, David B. Jaffe, Ken Stanley, Jonathan Butler, Sante Gnerre, Evan Mauceli, Bonnie Berger, Jill P. Mesirov, and Eric S. Lander. ARACHNE: A Whole-Genome Shotgun Assembler. *Genome Research*, 12(1):177–189, 2002.

[12] David R. Bentley, Shankar Balasubramanian, Harold P. Swerdlow, Geoffrey P. Smith, John Milton, Clive G. Brown, Kevin P. Hall, Dirk J. Evers, Colin L. Barnes, Helen R. Bignell, et al. Accurate whole human genome sequencing using reversible terminator chemistry. *Nature*, 456(7218):53–59, 2008.

[13] Guillaume Blanc, Abdelali Barakat, Romain Guyot, Richard Cooke, and Michel Delseny. Extensive duplication and reshuffling in the arabidopsis genome. *The Plant Cell Online*, 12(7):1093–1101, 2000.

[14] Guillaume Blanc, Karsten Hokamp, and Kenneth H Wolfe. A recent polyploidy superimposed on older large-scale duplications in the arabidopsis genome. *Genome research*, 13(2):137–144, 2003.

[15] Guillaume Bourque, Pavel A Pevzner, and Glenn Tesler. Reconstructing the genomic architecture of ancestral mammals: lessons from human, mouse, and rat genomes. *Genome research*, 14(4):507–516, 2004.

[16] Guillaume Bourque, Yasmine Yacef, and Nadia El-Mabrouk. Maximizing synteny blocks to identify ancestral homologs. In *Comparative Genomics*, pages 21–34. Springer, 2005.

[17] John E Bowers, Brad A Chapman, Junkang Rong, and Andrew H Paterson. Unravelling angiosperm genome evolution by phylogenetic analysis of chromosomal duplication events. *Nature*, 422(6930):433–438, 2003.

[18] Michael Brudno, Sanket Malde, Alexander Poliakov, Chuong B Do, Olivier Couronne, Inna Dubchak, and Serafim Batzoglou. Glocal alignment: finding rearrangements during alignment. *Bioinformatics*, 19(suppl 1):i54–i62, 2003.

[19] J. Butler, I. MacCallum, M. Kleber, I.A. Shlyakhter, M.K. Belmonte, E.S. Lander, C. Nusbaum, and D.B. Jaffe. ALLPATHS: de novo assembly of whole-genome shotgun microreads. *Genome Research*, 18(5):810, 2008.

[20] Jonathan Butler, Iain MacCallum, Michael Kleber, Ilya A. Shlyakhter, Matthew K. Belmonte, Eric S. Lander, Chad Nusbaum, and David B. Jaffe. ALLPATHS: de novo assembly of whole-genome shotgun microreads. *Genome Research*, 18:810–820, 2008.

[21] Peter P Calabrese, Sugata Chakravarty, and Todd J Vision. Fast identification and statistical evaluation of segmental homologies in comparative maps. *Bioinformatics*, 19(suppl 1):i74–i80, 2003.

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

[23] Mark J. Chaisson and Pavel A. Pevzner. Short read fragment assembly of bacterial genomes. *Genome Research*, 18(2):324–330, 2008.

[24] M.J. Chaisson, D. Brinza, and P.A. Pevzner. De novo fragment assembly with short mate-paired reads: Does the read length matter? *Genome Research*, 19(2):336, 2009.

[25] M.J. Chaisson and P.A. Pevzner. Short read fragment assembly of bacterial genomes. *Genome Research*, 18(2):324, 2008.

[26] K. Chen, J.W. Wallis, M.D. McLellan, D.E. Larson, J.M. Kalicki, C.S. Pohl, S.D. McGrath, M.C.

Wendl, Q. Zhang, D.P. Locke, et al. BreakDancer: an algorithm for high-resolution mapping of genomic structural variation. *Nature Methods*, 6(9):677–681, 2009.

[27] R. Chikhi and D. Lavenier. Localized genome assembly from reads to scaffolds: practical traversal of the paired string graph. *Algorithms in Bioinformatics*, pages 39–48, 2011.

[28] H. Chitsaz, J.L. Yee-Greenbaum, G. Tesler, M.J. Lombardo, C.L. Dupont, J.H. Badger, M. Novotny, D.B. Rusch, L.J. Fraser, N.A. Gormley, O. Schulz-Trieglaff, G.P. Smith, D.J. Evers, P.A. Pevzner, and R.S. Lasken. Efficient de novo assembly of single-cell bacterial genomes from short-read data sets. *Nat Biotechnol*, 29(10):915–921, 2011.

[29] Liying Cui, P Kerr Wall, James H Leebens-Mack, Bruce G Lindsay, Douglas E Soltis, Jeff J Doyle, Pamela S Soltis, John E Carlson, Kathiravetpilla Arumuganathan, Abdelali Barakat, et al. Widespread genome duplications throughout the history of flowering plants. *Genome Research*, 16(6):738–749, 2006.

[30] Aaron CE Darling, Bob Mau, Frederick R Blattner, and Nicole T Perna. Mauve: multiple alignment of conserved genomic sequence with rearrangements. *Genome research*, 14(7):1394–1403, 2004.

[31] Aaron E Darling, Bob Mau, Frederick R Blattner, and Nicole T Perna. Gril: genome rearrangement and inversion locator. *Bioinformatics*, 20(1):122–124, 2004.

[32] E.D. Demaine and M.L. Demaine. Jigsaw puzzles, edge matching, and polyomino packing: Connections and complexity. *Graphs and Combinatorics*, 23:195–208, 2007.

[33] Colin N Dewey, Peter M Huggins, Kevin Woods, Bernd Sturmfels, and Lior Pachter. Parametric alignment of drosophila genomes. *PLoS Computational Biology*, 2(6):e73, 2006.

[34] Fred S Dietrich, Sylvia Voegeli, Sophie Brachat, Anita Lerch, Krista Gates, Sabine Steiner, Christine Mohr, Rainer Pöhlmann, Philippe Luedi, Sangdun Choi, et al. The ashbya gossypii genome as a tool for mapping the ancient saccharomyces cerevisiae genome. *Science*, 304(5668):304–307, 2004.

[35] N. Donmez and M. Brudno. Hapsembler: An Assembler for Highly Polymorphic Genomes. In *Research in Computational Molecular Biology*, pages 38–52. Springer, 2011.

[36] Radoje Drmanac, Andrew B. Sparks, Matthew J. Callow, Aaron L. Halpern, Norman L. Burns, Bahram G. Kermani, Paolo Carnevali, Igor Nazarenko, Geoffrey B. Nilsen, George Yeung, et al. Human genome sequencing using unchained base reads on self-assembling DNA nanoarrays. *Science*, 327(5961):78, 2010.

[37] Zheng Fu, Xin Chen, Vladimir Vacic, Peng Nan, Yang Zhong, and Tao Jiang. Msoar: a high-throughput ortholog assignment system based on genome rearrangement. *Journal of Computational Biology*, 14(9):1160–1175, 2007.

[38] Wataru Fujibuchi, Hiroyuki Ogata, Hideo Matsuda, and Minoru Kanehisa. Automatic detection of conserved gene clusters in multiple genomes by graph comparison and p-quasi grouping. *Nucleic acids research*, 28(20):4029–4036, 2000.

[39] Brian J Haas, Arthur L Delcher, Jennifer R Wortman, and Steven L Salzberg. Dagchainer: a tool for mining segmental genome duplications and synteny. *Bioinformatics*, 20(18):3643–3646, 2004.

[40] Steven E Hampson, BS Gaut, and Pierre Baldi. Statistical detection of chromosomal homology using shared-gene density alone. *Bioinformatics*, 21(8):1339–1348, 2005.

[41] Timothy D. Harris, Phillip R. Buzby, Hazen Babcock, Eric Beer, Jayson Bowers, Ido Braslavsky, Marie Causey, Jennifer Colonell, Jamers DiMeo, J. William Efcavitch, et al. Single-molecule DNA sequencing of a viral genome. *Science*, 320(5872):106, 2008.

[42] D. Haussler, S.J. O'Brien, O.A. Ryder, F.K. Barker, M. Clamp, A.J. Crawford, R. Hanner, O. Hanotte, W.E. Johnson, J.A. McGuire, et al. Genome 10K: a proposal to obtain whole-genome sequence for 10000 vertebrate species. *J Hered*, 100(6):659–674, 2009.

[43] T Hubbard, Daniel Barker, Ewan Birney, Graham Cameron, Yuan Chen, L Clark, Tony Cox, J Cuff, Val Curwen, Thomas Down, et al. The ensembl genome database project. *Nucleic acids research*, 30(1):38–41, 2002.

[44] Ramana M. Idury and Michael S. Waterman. A new algorithm for DNA sequence assembly. *Journal of Computational Biology*, 2:291–306, 1995.

[45] Olivier Jaillon, Jean-Marc Aury, Frédéric Brunet, Jean-Louis Petit, Nicole Stange-Thomann, Evan Mauceli, Laurence Bouneau, Cécile Fischer, Catherine Ozouf-Costaz, Alain Bernot, et al. Genome duplication in the teleost fish tetraodon nigroviridis reveals the early vertebrate proto-karyotype. *Nature*, 431(7011):946–957, 2004.

[46] Z. Jiang, H. Tang, M. Ventura, M.F. Cardone, T. Marques-Bonet, X. She, P.A. Pevzner, and E.E. Eichler. Ancestral reconstruction of segmental duplications reveals punctuated cores of human genome evolution. *Nature genetics*, 39(11):1361–1368, 2007.

[47] M. Kampel and R. Sablatnig. 3d puzzling of archeological fragments. In *Proc. of 9th Computer Vision Winter Workshop*, volume 2. Slovenian Pattern Recognition Society, 2004.

[48] John D. Kececioglu. *Exact and approximation algorithms for DNA sequence reconstruction*. PhD thesis, University of Arizona, Tucson, AZ, USA, 1992.

[49] D.R. Kelley, M.C. Schatz, and S.L. Salzberg. Quake: quality-aware detection and correction of sequencing errors. *Genome Biology*, 11(11):R116, 2010.

[50] M. Kellis, B.W. Birren, and E.S. Lander. Proof and evolutionary analysis of ancient genome duplication in the yeast Saccharomyces cerevisiae. *Nature*, 428(6983):617–624, 2004.

[51] W James Kent, Robert Baertsch, Angie Hinrichs, Webb Miller, and David Haussler. Evolution's cauldron: duplication, deletion, and rearrangement in the mouse and human genomes. *Proceedings of the National Academy of Sciences*, 100(20):11484–11489, 2003.

[52] Romain Koszul, Sandrine Caburet, Bernard Dujon, and Gilles Fischer. Eucaryotic genome evolution through the spontaneous duplication of large chromosomal segments. *The EMBO journal*, 23(1):234–243, 2003.

[53] R. Li, H. Zhu, J. Ruan, W. Qian, X. Fang, Z. Shi, Y. Li, S. Li, G. Shan, K. Kristiansen, et al. De novo assembly of human genomes with massively parallel short read sequencing. *Genome Research*, 20(2):265, 2010.

[54] Jian Ma, Louxin Zhang, Bernard B Suh, Brian J Raney, Richard C Burhans, W James Kent, Mathieu Blanchette, David Haussler, and Webb Miller. Reconstructing contiguous regions of an ancestral genome. *Genome Research*, 16(12):1557–1565, 2006.

[55] Masayuki Machida, Kiyoshi Asai, Motoaki Sano, Toshihiro Tanaka, Toshitaka Kumagai, Goro Terai, Ken-Ichi Kusumoto, Toshihide Arima, Osamu Akita, Yutaka Kashiwagi, et al. Genome sequencing and analysis of aspergillus oryzae. *Nature*, 438(7071):1157–1161, 2005.

[56] Nicolas Martin, Elizabeth A Ruedi, Richard LeDuc, Feng-Jie Sun, and Gustavo Caetano-Anollés. Gene-interleaving patterns of synteny in the saccharomyces cerevisiae genome: are they proof of an ancient genome duplication event? *Biology direct*, 2(1):23, 2007.

[57] P. Medvedev, S. Pham, M. Chaisson, G. Tesler, and P. Pevzner. Paired de Bruijn Graphs: A Novel Approach for Incorporating Mate Pair Information into Genome Assemblers. In *Research in Computational Molecular Biology*, pages 238–251. Springer, 2011.

[58] Paul Medvedev and Michael Brudno. Ab initio whole genome shotgun assembly with mated short reads. In *Proceedings of the 12th Annual International Conference on Research in Computational Molecular Biology*, pages 50–64, 2008.

[59] Paul Medvedev, Konstantinos Georgiou, Gene Myers, and Michael Brudno. Computability of models for sequence assembly. In *WABI*, pages 289–301, 2007.

[60] A. Moitra and G. Valiant. Settling the polynomial learnability of mixtures of gaussians. In *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, pages 93–102. IEEE, 2010.

[61] Eugene W. Myers. The fragment assembly string graph. *Bioinformatics*, 21(suppl 2):ii79–ii85, 2005.

[62] E.W. Myers. Toward simplifying and accurately formulating fragment assembly. *Journal of Computational Biology*, 2(2):275–290, 1995.

[63] Benedict Paten, Javier Herrero, Kathryn Beal, Stephen Fitzgerald, and Ewan Birney. Enredo and pecan: genome-wide mammalian consistency-based multiple alignment with paralogs. *Genome research*, 18(11):1814–1828, 2008.

[64] Qian Peng, Max A Alekseyev, Glenn Tesler, and Pavel A Pevzner. Decoding synteny blocks and large-scale duplications in mammalian and plant genomes. In *Algorithms in Bioinformatics*, pages 220–232. Springer, 2009.

[65] Y. Peng, H.C.M. Leung, S.-M. Yiu, and F.Y.L. Chin. IDBA — A Practical Iterative de Bruijn Graph De Novo Assembler. In Bonnie Berger, editor, *RECOMB, Lisbon, Portugal, April 25-28, 2010. Proceedings*, volume 6044 of *Lecture Notes in Computer Science*, pages 426–440. Springer, 2010.

[66] P. A. Pevzner, H. Tang, and G. Tesler. De novo repeat classification and fragment assembly. *Genome Res.*, 14(9):1786–1796, 2004.

[67] P.A. Pevzner and H. Tang. Fragment assembly with double-barreled data. *Bioinformatics*, 17(suppl 1):S225, 2001.

[68] P.A. Pevzner, H. Tang, and M.S. Waterman. An Eulerian path approach to DNA fragment assembly. *PNAS*, 98(17):9748, 2001.

[69] Pavel Pevzner and Glenn Tesler. Genome rearrangements in mammalian evolution: lessons from human and mouse genomes. *Genome Research*, 13(1):37–45, 2003.

[70] Pavel A. Pevzner. *L*-Tuple DNA sequencing: computer analysis. *J Biomol Struct Dyn*, 7(1):63–73, 1989.

[71] Pavel A. Pevzner and Haixu Tang. Fragment assembly with double-barreled data. *Bioinformatics*, 17(suppl 1):S225–S223, 2001.

[72] Pavel A. Pevzner, Haixu Tang, and Glenn Tesler. De novo repeat classification and fragment assembly. *Genome Research*, 14(9):1786–1796, 2004.

[73] Pavel A. Pevzner, Haixu Tang, and Michael S. Waterman. An Eulerian path approach to DNA fragment assembly. *Proceedings of the National Academy of Sciences of the United States of America*, 98(17):9748–9753, 2001.

[74] S.K. Pham, D. Antipov, A. Sirotkin, G. Tesler, P.A. Pevzner, and M.A. Alekseyev. Pathset graphs: A novel approach for comprehensive utilization of paired reads in genome assembly. *Journal of Computational Biology*, 19, 2012.

[75] B. Raphael, D. Zhi, H. Tang, and P. Pevzner. A novel method for multiple alignment of sequences with repeated and shuffled elements. *Genome Research*, 14(11):2336, 2004.

[76] David Sankoff. Gene and genome duplication. *Current opinion in genetics & development*, 11(6):681–684, 2001.

[77] David Sankoff and Mathieu Blanchette. The median problem for breakpoints in comparative genomics. In *Computing and combinatorics*, pages 251–263. Springer, 1997.

[78] Devin R Scannell, A Carolin Frank, Gavin C Conant, Kevin P Byrne, Megan Woolfit, and Kenneth H Wolfe. Independent sorting-out of thousands of duplicated gene pairs in two yeast species descended from a whole-genome duplication. *Proceedings of the National Academy of Sciences*, 104(20):8397–8402, 2007.

[79] Michael C. Schatz, Arthur L. Delcher, and Steven L. Salzberg. Assembly of large genomes using second-generation sequencing. *Genome Research*, 20(9):1165–1173, 2010.

[80] Cathal Seoighe and Kenneth H Wolfe. Yeast genome evolution in the post-genome era. *Current opinion in microbiology*, 2(5):548–554, 1999.

[81] Cedric Simillion, Koen Janssens, Lieven Sterck, and Yves Van de Peer. i-adhore 2.0: an improved tool to detect degenerated genomic homology using genomic profiles. *Bioinformatics*, 24(1):127–128, 2008.

[82] Jared T. Simpson, Kim Wong, Shaun D. Jackman, Jacqueline E. Schein, Steven J.M. Jones, and Inanc Birol. ABySS: A parallel assembler for short read sequence data. *Genome Research*, 6:1117, 2009.

[83] J.T. Simpson, K. Wong, S.D. Jackman, J.E. Schein, S.J.M. Jones, and İ. Birol. ABySS: a parallel assembler for short read sequence data. *Genome Research*, 19(6):1117, 2009.

[84] Steven Skiena. *Implementing discrete mathematics: combinatorics and graph theory with Mathematica*. Addison-Wesley Longman Publishing Co., Inc., 1991.

[85] Carol Soderlund, William Nelson, Austin Shoemaker, and Andrew Paterson. Symap: A system for discovering and viewing syntenic regions of fpc maps. *Genome research*, 16(9):1159–1168, 2006.

[86] Firas Swidan, Eduardo PC Rocha, Michael Shmoish, and Ron Y Pinter. An integrative method for accurate comparative genome mapping. *PLoS computational biology*, 2(8):e75, 2006.

[87] Yves Van de Peer et al. Tetraodon genome confirms takifugu findings: most fish are ancient polyploids. *Genome Biol*, 5(12):250, 2004.

[88] Klaas Vandepoele, Yvan Saeys, Cedric Simillion, Jeroen Raes, and Yves Van de Peer. The automatic detection of homologous regions (adhore) and its application to microcolinearity between arabidopsis and rice. *Genome Research*, 12(11):1792–1801, 2002.

[89] Todd J Vision, Daniel G Brown, and Steven D Tanksley. The origins of genomic duplications in arabidopsis. *Science*, 290(5499):2114–2117, 2000.

[90] James L. Weber and Eugene W. Myers. Human whole-genome shotgun sequencing. *Genome Research*, 7:401–409, 1997.

[91] Kenneth H Wolfe. Yesterday's polyploids and the mystery of diploidization. *Nature Reviews Genetics*, 2(5):333–341, 2001.

[92] Kenneth H Wolfe, Denis C Shields, et al. Molecular evidence for an ancient duplication of the entire yeast genome. *Nature*, 387(6634):708–712, 1997.

[93] S. Young, R. Barthelson, A. McFarlin, and S. Rounsley. PLANTAGORA toolset, 2011. http://www.plantagora.org.

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

[95] D.R. Zerbino and E. Birney. Velvet: algorithms for de novo short read assembly using de Bruijn graphs. *Genome Research.*, 18(5):821, 2008.