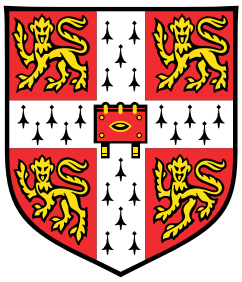
Graphical pangenomics



Erik Peter Garrison

Wellcome Sanger Institute
University of Cambridge

This dissertation is submitted for the degree of
Doctor of Philosophy

Fitzwilliam College

September 2018

for E_2 & E_3

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Erik Peter Garrison
September 2018

Graphical pangenomics

Erik Peter Garrison

Completely sequencing genomes is expensive, and to save costs we often analyze new genomic data in the context of a reference genome. This approach distorts our image of the inferred genome, an effect which we describe as reference bias. To mitigate reference bias, I repurpose graphical models previously used in genome assembly and alignment to serve as a reference system in resequencing. To do so I formalize the concept of a *variation graph* to link genomes to a graphical model of their mutual alignment that is capable of representing any kind of genomic variation, both small and large. As this model combines both sequence and variation information in one structure it serves as a natural basis for resequencing. By indexing the topology, sequence space, and haplotype space of these graphs and developing generalizations of sequence alignment suitable to them, I am able to use them as reference systems in the analysis of a wide array of genomic systems, from large vertebrate genomes to microbial pangenomes. To demonstrate the utility of this approach, I use my implementation to solve resequencing and alignment problems in the context of *Homo sapiens* and *Saccharomyces cerevisiae*. I use graph visualization techniques to explore variation graphs built from a variety of sources, including diverged human haplotypes, a gut microbiome, and a freshwater viral metagenome. I find that variation aware read alignment can eliminate reference bias at known variants, and this is of particular importance in the analysis of ancient DNA, where existing approaches result in significant bias towards the reference genome and concomitant distortion of population genetics results. I validate that the variation graph model can be applied to align RNA sequencing data to a splicing graph. Finally, I show that a classical pangenomic inference problem in microbiology can be solved using a resequencing approach based on variation graphs.

Acknowledgements

This work responds to ideas that arose in conversation with Deniz Kural. Our friendship is the first reason that I became a biologist, and his exploration of graphical models for genomes inspired my own. It is thanks to Alexander Wait Zaranek that we had the opportunity to work in George Church's lab, which pulled us both into biology from our previous fields. There I met Madeline Price Ball, who guided me during an immersive and engaging introduction to biology and genomics.

Deniz introduced me to Gabor Marth, with whom I apprenticed in the art of bioinformatics. Gabor encouraged me to contribute extensively to the 1000 Genomes Project, whose objective captured my imagination and whose participants, in particular the members of the analysis group, taught me many lessons in the way of science. I can thank Hyun Min Kang, Goncalo Abecasis, Adam Auton, Laura Clarke, Gerton Lunter, Mark DePristo, Lisa Brooks, Ryan Poplin, Zamin Iqbal, and Heng Li, for always motivating me, and for helping me to understand and correct the many mistakes I made. Meanwhile, Mengyao Zhao and Wan-Ping Lee gave me my first look inside the alignment algorithms that are such an important part of this thesis.

During those years I had the pleasure of living with Benjamin "Mako" Hill and Mika Matsuzaki, who showed me what it means to work as a scientist for the commons. Not only did I learn from them, but from the many thinkers, dreamers, and travelers who they brought into our life in Somerville. These include Hanna Wallach, who helped me to understand the theory and practice of the learning problems I first encountered in genomics, and Nicolás Della Penna, who continues to shape my understanding of many aspects of the scientific artifice, in particular the fuzzy boundary between the social and the technical. I thank my friends Nathan Trachimowicz and Barbara Eghan, with whom I passed so much time in those years, for not letting me lose touch with the beautiful natural and human world in which this work lives and from which it derives its purpose.

This thesis would cover a considerably narrower set of topics if not for the efforts of the many people with whom I have worked to build the variation graph toolkit, `vg`. These include, but are not limited to: Jouni Sirén, from whom I learned about the world of succinct data structures as I watched him build the graph sequence indexes that brought

vg to a level of quality I never could have achieved on my own; Benedict Paten, who applied his unique expertise on genome graphs to help guide the effort of the ever-growing group of project collaborators without a pause in his own stream of contributions; Eric Dawson, whose ready conversation, energy and persistence buoyed me in the early days of vg, and who laid the foundation for future work on structural variant calling on graphs; Shilpa Garg, who brought new ideas about assembly and diploid genome inference to our project while helping to establish the long read alignment algorithms in vg; Adam Novak, who arrived first and transformed the heart of vg from a weak toy model into a foundation suitable for the work of this wide-ranging team, and who continues to carry it forward; Charles Markello, whose precise work on building resequencing pipelines with vg has ensured it is and will be widely usable; Emily Kobayashi, who further honed vg's topology index and is pushing improvement in the dynamic graph indexes we use; Wolfgang Beyer, Toshiyuki Yokoyama, Orion Buske, and Ryan Wick, whose work on sequence graph visualization gave me eyes to understand my work; William Jones, whose clear-minded experiments on alignment identity and score comparison provide the basis for so many figures in this work; Hajime Suzuki, whose work on alignment acceleration lies at the core of the next phase of graph based mappers; Jordan Eizenga, with whom I explored the deep complexity of string to graph alignment algorithms; Xian Chang, who is driving the next high-performance paradigm for sequence to graph alignment; Mike Lin, whose experience and tact guided me both in the vg project and in extracurricular work for DNAnexus; Yohei Rosen, who showed us that old models of haplotype matching could thrive inside our new pangenomic, graphical world; Glenn Hickey, who, in addition to caring for the project, took vg full circle and built variant calling into the graph model; Jerven Bolleman, who found a way to link vg into the enormous world of semantic biological data; and Toshiaki Katayama, who has explored our work and done so much to bring developers of graphical pangenomic techniques together. These members of the "vgteam" have each shared so much more than I can describe here, and I am deeply grateful to have had the opportunity to work with them. The work that I present here is fundamentally based on the productive collaboration that we have shared, and I could not have completed it without their watchful critique and steadfast exploration of their own research questions.

During my time as a student, I benefited from many long conversations with my fellow Sanger PhD cohort over lunch and coffee at the Genome Campus. In particular I learned from the other students with whom I lived: Ignacio Vázquez García, Martin Fabry, Daniel Bruder, Manuela Carrasquilla, and Girish Nivarti.

Pedro Fernandes gave Tobias Marschall and me the chance to teach a course on pangenomics using `vg`, which motivated many of the applications of variation graphs that I present here. Eppie Jones, Rui Martiniano, and Daniel Wegmann have guided and supported my work on ancient DNA. Remo Sanges and Mariella Ferrante gave me a lab to be part of and a fascinating project to explore in my time in Napoli. My corrections for this thesis drew on Alex Bowe's excellent series of blog posts on succinct data structures, and I thank him for allowing me to exposit some of his examples here.

The final version of this thesis reflects a long and memorable conversation led by my examiners Aylwyn Scally and Gerton Lunter. I thank them for their clear suggestions for its improvement, and will forever be grateful for the time they devoted to this sprawling work. They focused my time on its roughest and most incomplete aspects. I am proud of the result that their critique has encouraged me to achieve.

Working with Richard Durbin has been a singular pleasure. Richard has an expansive vision for genomics, but he is always ready to dig into the details of a problem. He is a true master of his craft, able to support and guide every aspect of our work. The group he leads is motivated by his wide-ranging interests in biology. I owe its former and current members thanks for their encouragement and imagination.

Without my family, it is unlikely I would have ever begun the meandering trip that has led me to this thesis. My parents, Mark Garrison and Diane Garrison, helped me to be independent, and opened my mind to the world of ideas, which set me out on a wonderful trip. Along the way, my brother Nels and sister Astrid have kept me honest and careful of myself.

Much of this trip has been alongside my partner Enza Colonna. Non so come dire quanto mi ha aiutato, o quanti passi ho fatto in questo viaggo secondo le idee che abbiamo condiviso. I also am grateful to her parents, Donato Colonna e Concetta Tummolo, under whose almond and olive trees I wrote many pages of this work. Mi hanno sollevato dai problemi di vita quotidiana, e con il loro aiuto ho potuto scrivere la prima bozza di questa tesi in un solo mese.

Our daughter, Exa, who always convinces me to play, made sure that I was never too tired to keep going. I look forward to sharing this with her.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

All life on our planet is connected through a shared history recorded in its DNA. Over time, the genomes of organisms are copied, sometimes with error or recombination. These mutations give rise to genetic, and ultimately phenotypic diversity. Through isolation and drift, genetic diversity enables and defines the generation of new species.

Although easily surmised, this basic process is often forgotten at the level of the most common analyses in genomics. When considering the genomes of many individuals, we frequently pluck a single related genome from the tree of life to use as a reference. Using alignment, we express our sequencing data from the collection of samples in terms of positions and edits to the reference sequence. We then use variant calling and phasing algorithms to filter and structure these edits into a reconstruction of the underlying haplotypes. We can then proceed to use the inferred genotypes and haplotypes to answer biological questions of interest.

In this way, we have not fully sequenced the new genomes, but *resequenced* them against the reference genome. Pieces of the new genomes which could not be mapped to the reference will be left out of our analysis, which can distort our results.

Resequencing has arisen in response to the technical properties of the most commonly-used DNA sequencing technologies. These "second generation" sequencing-by-synthesis technologies produce abundant and inexpensive short reads of up to 250 base pairs, and in the past decade have become the largest source of data in the DNA sequencing market.

Higher sequencing costs previously motivated the application of expensive computational approaches to analyze all the sequences of interest simultaneously. The decades prior to the development of cheap sequencing saw the use of multiple sequence alignment algorithms with high computational complexity. Analyzing hundreds or thousands of sequences with such techniques is expensive but justifiable given the costs of acquiring them.

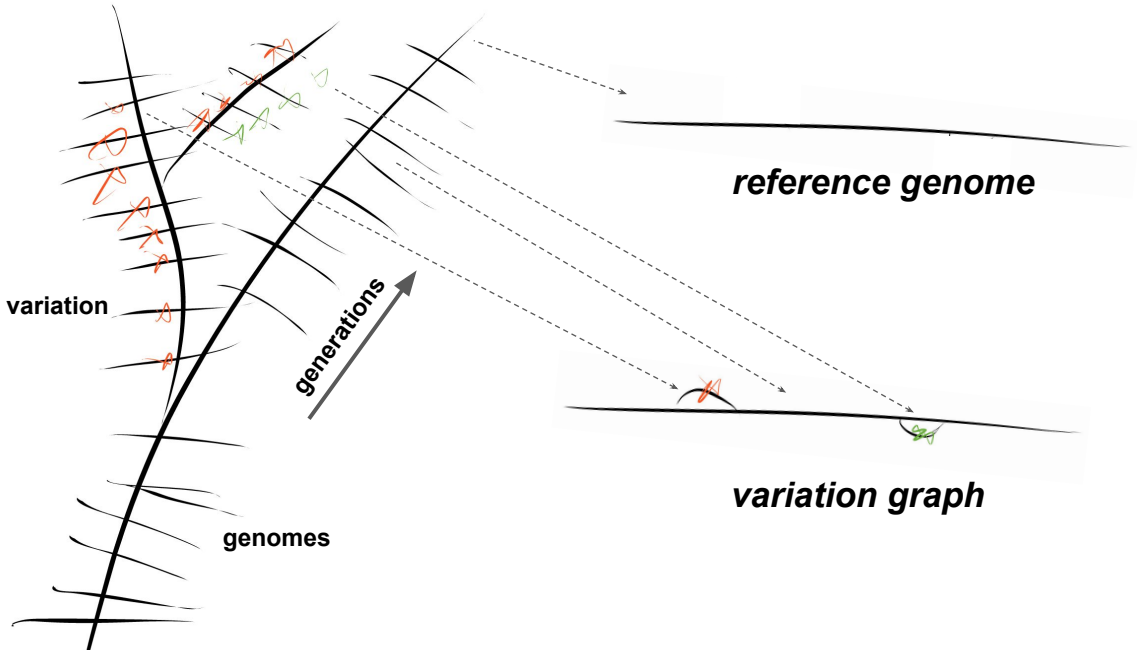


Fig. 1.1 The tree of life, reference genomes, and variation graphs.

.

However, such approaches became too expensive as new sequencing technologies allowed the generation of tens and then hundreds of gigabytes of data in a single run. The new, low-cost techniques allowed joint analyses of thousands of genomes from a single species. Resequencing provided a practical means to complete these analyses. By relating data to a common linear reference system, the alignment phase could be completed independently and in parallel, with each sample compared to the common reference genome. Only in a final phase of analysis might all the genome data be collected together for the inference of alleles at a given genetic locus.

In resequencing, the reference sequence shapes the observable space, resulting in an effect known as *reference bias*. DNA sequencing reads that contain sequence which is divergent from or not present in the reference sequence are likely to be unmapped or mismapped. This results in lower coverage for non-reference alleles, in effect forcing new samples to appear more similar to the reference than they actually are. Divergence itself frustrates the genome inference process, as alignment may produce different descriptions of diverged sequences depending on the relative position of the read. Alignment works best when the sequences we are aligning are similar to the reference. Increasing divergence requires greater computational expenditure to overcome reference bias.

We can avoid reference bias by working on pure assemblies generated only from the sequencing data in our experiment and unguided by any prior information. Doing so can be rigorous, but comes at a significant cost. Obtaining whole genome *de novo* assemblies requires greater sequencing and computational costs than resequencing, putting this approach out of reach for many study designs.

Genome assemblers frequently use a graphical transformation of their inputs that supports algorithm steps used to infer contigs implied by the reads. These data structures are typically bidirectional graphs in which nodes are labeled by sequences and edges represent observed linkages between sequences. If constructed from a set of reads that fully cover the genome, it can be shown that such a graph contains the genome which has been sequenced. In effect, the assembler works to filter the edges from the graph and un-collapse repeats in order to establish a sequence assembly.

In this work I repurpose the assembly graph data model to build a pangenomic reference system. Assembly graphs are designed to represent the full set of genomic information to which they are applied, so it is natural to use them to develop coherent reference systems for unbiased sequence analysis. By building a conceptual framework and data structures that enable resequencing against this structure, we can mirror the patterns and workflows that have already been developed for resequencing. This allows us to retain the benefits of parallel analysis even while we resolve the issue of reference bias. By recording genomic sequences as paths through this graph, I provide anchors for existing annotations and positional systems within the pangenome. I call these bidirectional sequence graphs with paths *variation graphs*.

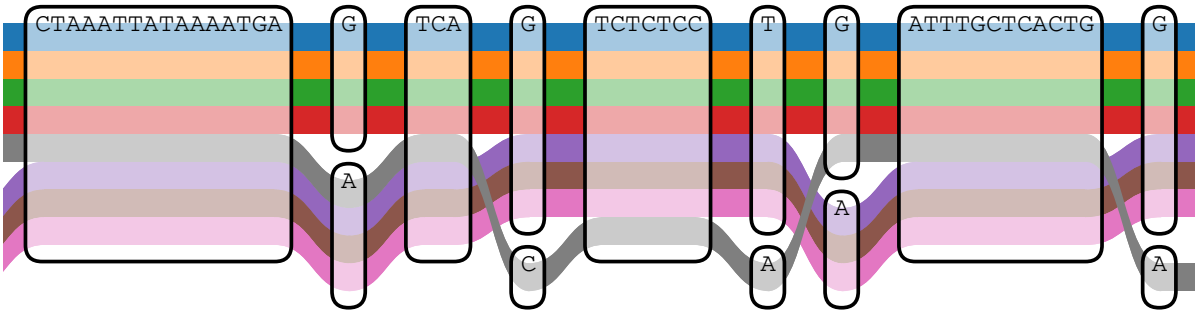


Fig. 1.2 A fragment of a variation graph built from fully-assembled *Saccharomyces cerevisiae* genomes. Colored paths represent genomes which traverse sequences (nodes). Edges are implied by the path structure of the graph. The construction and properties of the graph are described in section 3.1.2. This visualization was rendered using the SequenceTubeMap https://github.com/vgteam/sequenceTubeMap.

.

This chapter provides historical justification for the need for an integrative model for sequence analysis like the variation graph. I contextualize my work within the history of DNA sequencing (1.1), assembly algorithms (1.1.2), resequencing methods (1.2), and pangenomic models (1.3). This deep introduction is meant to justify the need for an integrative model like the variation graph to serve as a coordinating system in a research setting characterized by increasing data scale and complexity. Readers who need no further introduction to these issues should continue to section 1.4, which reviews recent work on algorithms based on data models similar to variation graphs.

The remainder of the thesis builds on this foundation. In chapter 2, I describe data structures and algorithms that allow the use of variation graphs as a reference system for unbiased genome inference. Finally, in chapter 3, I demonstrate the benefits of this approach with a series of experimental case studies.

1.1 Genome inference

Not two centuries have passed since the first experiments that demonstrated the existence of genetic material [183]. In the first part of the twentieth century, these ideas about heredity grew into the core of a modern synthesis linking biological micro- and macro-evolutionary theory to the quantitative basis of genetics [116]. It was understood that DNA encoded the information that gave rise to biological structures [11]. The discovery of the structure of DNA in the 1950s [285] made clear the nature of that information and the mechanism for its faithful transmission from generation to generation. This knowledge, coupled with the sequencing and synthesis of proteins, which demonstrated that they had distinct polymeric chemical identities [236] led to Crick's postulation of the "central dogma" of biology [50, 51]. Simply stated, the "dogma" argues that in living systems information is transcribed from DNA to RNA and ultimately translated into proteins, which guide and structure the cell and thus living organisms. The central dogma clarifies the significance of the sequence of the genome, and over the following decades a series of projects scaled up the throughput and fidelity of DNA sequencing until *genome inference* became a practical and everyday reality in biology.

1.1.1 Reading DNA

The quest to sequence genomes began with arduous and sometimes dangerously radioactive experimental techniques, in which years of researcher time could be spent in obtaining sequences of tens of bases from partly-characterized sources. It has then progressed

through three distinct phases. In the first, these early laboratory techniques gave way to automated sequencing using chain terminator chemistry, and related techniques were ultimately used to generate genome sequences for human and a number of organisms, albeit at high costs. In the second phase, *multiplex* sequencing reactions were used to miniaturize the chain terminator reaction and observe its progression using fluorescent imaging or electrical sensing, evoking a drop in cost per sequenced base of many orders of magnitude, and simplifying library preparation steps dramatically by sequencing clones of individual molecules. The third wave of development has been characterized by two techniques which allow realtime observation of single DNA molecules. These produce enormously long read lengths that are limited by the molecular weight of input DNA, but produce readouts with high per-base error rates. Supporting the second and third wave are methods that allow for haplotype-specific sequencing and the observation of long range structures in the genome.

1.1.1.1 The old school

In the 1970s a group led by Walter Fiers published the first complete gene [125], and then genome sequence [84] from the MS2 bacteriophage using laborious digestion and 2-dimensional gel electrophoresis techniques to sequence RNA based on work by Fredrick Sanger and colleagues [234, 3]. To avoid the limitations of digestion based assays, Ray Wu and colleagues developed a sequencing technique based on the partial blockage of DNA polymerization with radiolabeled nucleotides [290, 208]. Subsequently, Sanger developed a reliable DNA sequencing method based on the same DNA polymerization chain-termination concept by dividing the sequencing reaction into four, one for each base, and sorting the resulting DNA fragments in parallel on an acrylamide gel [238]. Optimized and implemented using fluorescent chemistry [264], this approach, now known as Sanger sequencing, became the foundation of the first commercial sequencing machines in the late 1980s.

Sanger sequencing was the workhorse standard of biology for nearly 30 years, from the late 1970s until the mid 2000s. Its read length is limited by the reaction efficiency required to obtain a fraction of terminations at every base in the sequence. In practice, reads of 500 to 1000 base pairs can be obtained. With clonal DNA as input the per base accuracy of the method is extremely high, as each base readout reflects the termination of large numbers of molecules [33], a feature which has ensured it remains important for validation of sequencing results [251]. However, heterogeneity in the input DNA library can produce muddled signals that rapidly become uninterpretable. Insertions and deletions (indels) will cause a loss of phase in the sequencing trace [271], a problem

which is still encouraging algorithm development [111]. In order to sequence whole genomes, which are often heterozygous, laboratory techniques were developed to allow the segregation of clonal DNA as a substrate for sequencing. These include bacterial artificial chromosomes (BACs) and their equivalent in yeast (YACs) [189]. The effective read length could be increased by using "mate pair" techniques, in which the ends of a longer molecule would be sequenced [242]. To yield fully assembled genomes, these data required the development of suitable computational techniques [198].

1.1.1.2 "Next generation" sequencing

In the late 1990s and early 2000s, several groups began exploring alternative sequencing strategies. In the ultimately dominant one, DNA that has been clonally arrayed on a surface is directly sequenced using fluorescent imaging. Sequencing progresses through the synthesis of the second strand of each of the molecules, and so these techniques are typically called "sequencing-by-synthesis." This modality allowed for a massive parallelization of the sequencing reaction, and has resulted in a dramatic reduction of cost.

In 2003 George Church and colleagues demonstrated that individual sequences could be read from polymerase colonies or "polonies" suspended in an acrylamide gel using fluorescence microscopy [188]. This fluorescent imaging model became the basis for "next generation" sequencing [246]. Contemporaneously, a sequencing-by-synthesis method which is now known as Illumina dye sequencing, was implemented using laser fluorescent imaging and reversible terminator technology developed by Shankar Balasubramanian and David Klenerman at Solexa (later acquired by Illumina) [12, 16]. Rather than polymerase colonies embedded in an emulsion or gel, Solexa's technology relied on "bridging PCR", in which the polymerized clones of a particular fragment were locally hybridized to an adapter-bearing surface of a flowcell. Controlled synthesis of the second strand, based on reversible terminator chemistry [30] and fluorescently labeled dNTPs, is then used to observe the sequence of the DNA molecule in each colony.

A diverse set of similar approaches were explored during this period, although few saw more than limited success in the sequencing market. Church's group focused on a hybridization based sequencing protocol proceeded by an emulsion based polony PCR step [247], and later attempted to commercialize an open source sequencing device (the Polonator)[1]. In ion semiconductor sequencing direct observation of pH changes were used

[1]My interest in open source projects, developed while an undergraduate studying the social sciences, led me to work on this device. The project introduced me to biology, bioinformatics, and DNA sequencing, which have attracted my interest and effort ever since.

to determine DNA sequences [233]. 454 Life Sciences' "pyrosequencing" implementation used a luciferase reporter assay to track the progression of DNA synthesis [177], and it was used to generate the first whole genome human sequence using "next generation" techniques [288]. Helicos commercialized the first single-molecule sequencing system, using a similar chemistry to Illumina's but observing single molecules rather than pools, which proved technically challenging and only saw use in its own development [108].

Illumina's sequencing protocol provides greater throughput and a superior error profile relative to these methods. Its low per base error rates and handful of context specific error types simplify analysis [6]. It is unsurprising that the vast majority of sequencing data produced in the 2010s comes from Illumina sequencers. Illumina's sequencing technology is characterized by short reads (<250bp) with per-base accuracy ($\approx 99.5\%$) comparable to that of Sanger sequencing. Although the read length has been increased by optimization of the technology, the difficulty of achieving perfect per-base reaction efficiency apparently prevents greater extension of the read length.

A number of methods extend the genome inference capacity of Illumina sequencing, allowing it to be used to infer long haplotypes and genome organization. Moleculo, and later 10X Genomics commercialized barcode-guided haplotype sequencing and assembly [296]. The later has focused on providing raw tag information that could be used downstream by an array of haplotype-resolution and assembly tools [191]. The single template aspect of Illumina paired end sequencing allows longer contiguous DNA reads to be obtained by merging partly-overlapping read pairs computationally [173]. Single-cell DNA template strand sequencing (strand-seq) can be used to obtain reads from only one half of the chromatids in a single cell [74] via bromodeoxyuridine (BrdU) treatment and cleavage of the nascent strand, which can aid in haplotype reconstruction [219]. The Hi-C method [164] uses bisulfite treatment to generate read pairs that are likely to physically co-locate *in vivo*, thus enabling the mapping of long range DNA and chromatin interactions. It may be combined with other sequencing information to obtain estimates of the syntenic ordering of contigs produced by assembly [93], which has already been used to obtain *de novo* reference quality genomes in several difficult sequencing projects including amaranth [165], *Aedes aegypti* [64], and the domestic goat [18].

1.1.1.3 Single molecules

All previously described sequencing techniques are dependent on the observation of pools of molecules. These methods benefit from amplification of DNA, which increases signal, but also adds and a potential source of error to DNA sequencing. They also suffer from de-phasing resulting from imperfect stepwise reaction efficiency, which fundamentally

limit the maximum length of an accurate read. A method to sequence single molecules accurately would theoretically allow longer read lengths, but this requires the difficult, direct observation of DNA. Efforts to develop such a method have been continuously underway throughout 2000s and 2010s. Two successful commercial sequencing platforms based on this principle are rapidly defining a new technical phase of genome inference.

By utilizing zero-mode waveguides (ZMWs) to observe DNA polymerase in real time, Pacific Biosciences (PacBio) generated the first successful commercial single-molecule sequencing system [72]. In this platform, DNA polymerase is immobilized in sub-diffraction size, picoliter detection volumes at the bottom of wells formed in aluminum on a glass slide [142]. Single stranded DNA and fluorescently-labeled dNTPs are added to the buffer above the ZMWs. As synthesis progresses, the fluorophore attached to the DNA base that is being incorporated will tend to remain inside the ZMW longer than would be expected due to random diffusion of the dNTPs, allowing a readout of the sequence of incorporated bases as a series of fluorescent pulses. The base-level error rate of sequencing is high, up to 15%. It is difficult to perfectly observe the series of fluorophores pulled into the well, and random occupancy is often indistinguishable from polymerization-mediated occupancy, which results in insertion errors. Although subtle context dependent biases do exist [206], due to their genesis in Brownian motion, the errors themselves may be considered as almost perfectly random in analysis [232, 195]. In recent years PacBio's system has become a foundational technology in genome sequencing, with many recent genome assemblies completed using it [229].

The idea that electrophoresis of DNA through nanometer scale pores might allow the direct sequencing of DNA was first postulated in the late 1980s by David Deamer and others [58]. While the sequencing model itself is among the simplest ever proposed, it would take twenty-five years of work [128, 221] before the technique was brought to market by Oxford Nanopore (ONT) [185] and used to fully sequence genomes [171, 122]. In this approach, a DNA strand is pulled through a nanometer pore by electrophoresis. The specific DNA bases in the pore effect characteristic changes in the electric current density, and the DNA molecule can be read by measuring the changes in current over time. Due to context and history-dependent effects that distort the signal, the measured patterns in the current flux must be interpreted by sophisticated models that have been trained to convert the traces to DNA sequences [54]. As with PacBio, its per-base error rate approaches 15%. In practice nanopore sequencing has the highest error rate of any commercially available method, which reflects the difficulty of mapping between the observed signal and the underlying DNA sequence. Nanopore sequencing can also obtain

the longest reads of any sequencing technology, with megabase-scale reads reported by some users.

1.1.2 Genome assembly

Due to technical limits that are unlikely to ever be fully eliminated, individual DNA sequence reads are rarely able to cover the entire genome of an organism. This means that in many cases, the best sequencing data possible is a set of random reads sampled from fragments of the genome. In whole genome "shotgun" sequencing the genome is fragmented, perhaps by sonication or enzymatic digestion, and the resulting fragments are sequenced and then reassembled using computer programs [90, 235]. This process necessitates a reconstructive step in which the information obtained from the sequenced fragments is reassembled into the whole genome from which they arose. This process is known as *assembly*, and computer algorithms implementing it have been used when inferring genome sequences since the generation of the first whole genome sequence for bacteriophage φX174 in 1977 [237, 262].

The earliest assembly algorithms have come to be known as "overlap-layout-consensus" (OLC) algorithms, due to their three-phase strategy. They first establish a set of head to tail overlaps between reads (overlap), an $\approx O(N^2)$ order problem when all pairwise relationships are considered between N sequence reads. However, given an efficient method to find read pairs that are very likely to match together, the overlap step remains tractable as the overall complexity of matching can be reduced to be approximately quadratic in read depth and linear in genome size [114]. These overlaps are then used to establish an estimate of the ordering of the reads (layout). The layout is then used to generate a consensus sequence through heuristics or dynamic programming over the layout [130]. This final phase is equivalent to the multiple sequence alignment (MSA) problem, although instead of generating an MSA as output methods would typically take the consensus sequence, as the objective is often to reconstruct a linear representation of the input genome. Early assemblers committed frequent assembly errors, which necessitated time-consuming manual "finishing" [96]. The OLC assembly approach was utilized by genome projects for the following twenty-five years, including in the public Human Genome Project (HGP), where BAC clones of 150kb fragments of the genome were sequenced, initially assembled by algorithm and finally manually finished into the "golden path" that would become the reference genome [48].

In principle, the assembly process could be fully automated, but as late as the early 1990s this frequently was not seen as feasible due to the lack of reliable algorithms [175]. The improvement of OLC algorithms eventually met the challenge, yielding methods such

as PHRAP [102] (a quality aware assembler that saw extensive use downstream of Sanger sequencers), TIGR [267] (which was used in the generation of the first assembly of a free living organism, the 1.8Mbp genome of *Haemophilus influenzae* [86]), GigAssembler [133] (which was used by the HGP), and the Celera assembler [198, 186] (which saw extensive use in the generation of early large whole genome assemblies in the late 1990s and early 2000s, including the privately funded genome project [281][2].) The process implemented in the Celera assembler (including repeat masking) has remained essential to the genome assembly problem until the present.

In 2005, Myers formalized an idealized version of the assembly problem in the *string graph* data structure [194], which is a sequence graph induced from the overlaps in a set of shotgun sequencing reads. This model demonstrates that repeats greater than the length of a sequence read will collapse into single copies in the graph, while unique sequences will form loops between different repeat classes that flank them. The string graph can be shown to represent the full information available in the input sequence data, and successful assembly algorithms are built around an induction of the string graph via the construction of the FM-index [79] from Illumina read sets [253, 254, 155]. If not using compressed data structures and low-error reads, the repeats are often irresolvable and may be masked from the assembly process to improve performance on the tractable non-repetitive regions of the genome, which is a strategy still promoted and employed by Myers [195]. Canu and FALCON, which to some extent stand as contemporary implementations of the Celera assembly process, are among the best-performing assemblers for noisy single-molecule sequencing data that is the mainstay of current genome assembly projects [41, 141]. These and similar methods have shown that long reads can be used to fully assemble genomes without human finishing [171, 122].

The repeat problem has been tackled in various ways, but one of the most enduring solutions resolves the issue through the reduction of the assembly overlap graph to a de Bruijn graph (DBG) [217]. In this approach, the read set is fragmented into all subsequences of reads of a given length k, and a graph is constructed where k-mers label nodes and overlaps of $k-1$ between successive k-mers induce edges representing linkages between them. The de Bruijn graph simplifies the representation of the read set, providing a clean basis for assembly algorithms. It enabled the first [294, 255, 120], and most memory-efficient assembly methods for short read sequencing data, with techniques like bloom filters [39], succinct DBGs [23, 150], and minimizer partitioning [40] applied to generate a compressed representation of the graph. DBG based assemblies suffer from the

[2]This project apparently still relied on data produced by the HGP [284], but the significance of this reliance was disputed by researchers involved in the private project [199], who argued that the manner in which they used the public sequences avoided contamination by manual finishing done by the HGP.

loss of information induced through the k-merization of their input, causing a reduction in assembly contiguity [67], although in practice this can be mitigated by reconsideration of the input reads and read pairs [29]. They also are applicable only where the sequence error rate is low enough for overlapping reads to be expected to have exact matching k-mers of the appropriate size (typically, $k \in [20 \ldots 50]$ base pairs), and as such cannot be applied to third generation single molecule sequencing due to its inherently high error rate.

Many of the sequencing methods I have described above are still in use today. Each popular method, as it fades from use, remains relevant in a niche area where its particular properties provide it a comparative advantage. As a result, we are not presented today with a single ideal sequencing method, but a menagerie of approaches, each with its own limitations and benefits, and current assembly pipelines require thoughtful design to incorporate these myriad sources of information. It would appear that in order to use these many technologies to generate the best-possible assemblies we must bring them together in a single model [34]. A current development in assembly focuses on the design of a common interchange format for which to organize such assembly processes, which has been implemented as the GFA v1 and v2 formats.[3] This file format and the data model it implies is an essential link between the work that I present later in this thesis and the problem of genome inference.

1.2 Reference genomes

Obtaining a single genome sequence *de novo* is an arduous task, and remains a complex problem. The result is a valuable object which can be used to lower the cost of subsequent analyses and enable direct whole genome comparisons which provide a full perspective on the genetic relationship between multiple individuals or species. The need for reference genomes is clear, and they are collected in open public databases to allow their dissemination and use by researchers. NCBI's RefSeq release 89 of July 13, 2018 contains some 81,345 organisms[4], although it should be noted that only a small fraction of these genomes are eukaryotic. Recent developments in long read, single molecule sequencing have enabled great decreases in the cost and complexity of generating high-quality genome assemblies, supporting a recent project to generate reference quality genomes of ten thousand vertebrates [205, 140].

[3] https://github.com/GFA-spec/GFA-spec
[4] https://www.ncbi.nlm.nih.gov/refseq/

The reference genome serves as an anchor for annotations that describe sequences and regions of interest within the genome, such as genes, exons, chromatin structures, DNA interacting proteins, and genetic variation [248, 222, 47]. An established reference genome can serve as a conceptual foundation for the communication and interpretation of scientific results [134], and is seen as essential for collaboration and the development of a genome research community in a particular organism [260, 38].

Reference genomes tend to represent only a single version of each genomic locus. This conceptual simplicity is a core feature of their public use. Although the issue of genetic diversity has always been appreciated by those who work with genomes, expediency has encouraged the use of linear models for reference genomes. Within the HGP, members of the consortium could observe diversity within the BAC clones that they had sequenced from different human donors, and initiated a debate about the inclusion of heterozygosity in the reference itself. Ultimately, a graphical model was seen as too complicated, and practicality necessitated the publication of a linear reference based around what came to be called the "golden path" through the assembly[5]. Since early releases, the human reference genome has included alternative versions of some regions, with current releases including alternates for around 200 loci [244, 42], but these are represented as linear sequences without a unifying alignment between them, which complicates their use in resequencing and annotation [121].

1.2.1 Resequencing

Due to the high cost of obtaining error-free, full length genomes, standard practice will use the best genome assembly for a given organism as a *reference genome* when analyzing the sequences of other organisms from the same species. To do so, the genomes of the other individuals do not need to be fully assembled, and instead shotgun sequencing libraries from these new individuals may be aligned back to the reference to find small differences between the genomes. To distinguish it from whole genome sequencing and assembly, this process is known as *re*sequencing.

Resequencing has two phases. In the first phase, reads from the sample or samples under study are aligned against an appropriate, genetically similar, reference genome. In the second phase, the aligned reads (alignments) are processed together locus by locus to determine allelic variation within the samples relative to the reference genome.

[5]Personal communication with David Haussler.

1.2.2 Sequence alignment

An *alignment* expresses one sequence in terms of a set of positions, edits, and matches to another. Algorithms to determine the most plausible alignment between a pair of sequences have as long a history as sequencing itself. The first significant attempts to algorithmically assess sequence homology and divergence between protein sequences arose in the 1960s with Fitch's method for homology detection [85]. To account for insertions and deletions, this method required the comparison of many subsequences of two sequences to be compared, resulting in poor computational bounds. In 1970, Needleman and Wunsch responded with an O(NM) time algorithm for the global alignment of sequences [200]. Given strings to compare of length N and M, the algorithm builds an $M \times N$ matrix in which any possible full length alignment between both sequences can be expressed as a path through a series of cells. The matrix is designed such that a match corresponds to the shortest (diagonal) path through the matrix, and insertions and deletions correspond to horizontal or vertical movements. To determine the most-likely path, Needleman and Wunsch apply a recurrence relation dependent on the characters at each pair of positions in the strings and the values of the cells above and/or to the left. This implements a dynamic programming (DP) method [15]. For each cell, the score is given as the maximum of: the score of cell to the diagonal plus a bonus if the corresponding sequence characters are the same and minus a penalty if they are different; and the scores of the cells above and below minus a penalty corresponding to the weight given to an insertion or deletion. Finally, we determine the optimal path beginning from the opposite extreme cell of the matrix from where the scoring began, in which we walk back through the successive maximum scores until reaching the opposite extreme corner of the matrix. This "traceback" encodes the alignment, which is most-simply represented as a vector of pairs of matched bases in each sequence. It can be shown that provided full evaluation of the dynamic programming problem, the optimal alignment is obtained given a set of scores parameterizing the recurrence relation.

This alignment algorithm is known as a "global" alignment algorithm, in that the alignment covers all bases of both sequences. In practice, this type of comparison is not always needed, and it can be advantageous to obtain only the optimal sub alignments between sequences. Smith and Waterman provided a clean modification of the algorithm of Needleman and Wunsch, altering it to prevent negative scores, which allowed it to produce optimal "local" alignments [261], while ignoring regions unlikely to contain significant homology. The algorithm, further refined by Gotoh [97] to enable affine

gaps[6] and computation in $O(MN)$ time, is today one of the most important in genome analysis. The amount of work on this topic is considerable, and the subsequent decade yielded numerous modifications of the basic alignment concept, for instance reducing the memory bounds to $O(N)$ through a divide and conquer approach [196], and further explorations of affine gap scoring schemes [8, 98]. Subsequent works have offered improved implementations, using vectorized instructions to improve the runtime of the algorithm [76] and heuristics to selectively evaluate only part of the DP matrix [268]. However, such changes were not sufficient to enable alignment against large sequence databases.

$O(MN)$ algorithms for sequence alignment are impractical when either M or N becomes large. Naturally, as sequence databases grew and the size of sequenced genomes increased, heuristic strategies to efficiently reduce the alignment problem size were introduced. When aligning a short sequence against a large database we expect to obtain a sensitive alignment, but provided sufficient homology between the sequence and the database it is unlikely that we need to evaluate the full problem using an algorithm like Smith-Waterman-Gotoh (SWG). By indexing either the query or target set of sequences to efficiently obtain patterns of exact matches, candidate sub-regions of both can be isolated and submitted for more sensitive alignment.

This strategy was implemented in the mid- to late-1980s in the FASTA [215] and BLAST [9] alignment algorithms. FASTA first uses a seeding step that finds exact matches between the query and target, using chains of short k-mer seeds to establish the longest matching subsequences. A few of the best scoring candidates are enumerated and evaluated using a banded SWG algorithm. In contrast, BLAST implements a fully heuristic alignment process based solely on the k-mer seeds and ungapped alignment. This is much faster than FASTA but can perform slightly worse with highly divergent sequences. BLAST's heuristic alignment is many orders of magnitude faster than full DP based algorithms at a minor cost to accuracy. The popularity of BLAST in biology[7] is clear evidence of the importance of the alignment problem to all kinds of genomic analysis. It is also evidence that minor losses in accuracy are acceptable given the cost of sequence analysis in large data sets. Jim Kent's Blast-like alignment tool (BLAT) indexes the target set with non-overlapping k-mers and queries all k-mers in the reads, yielding a method that is less sensitive but several orders of magnitude faster again than BLAST [132].

[6] In affine gap schemes the cost of a gap per base decreases as its length increases. Such a scheme approximates the ζ-distributed excursions of a particle under Brownian motion, which structure the length of insertions and deletion mutations observed in nature.

[7] The BLAST1 paper has been cited more than 70,000 times as of August 2018.

As reliable commercial second-generation sequencing systems became available the rate of sequence data acquisition growth rapidly outstripped the rate of improvement in computing performance [149, 139]. This necessitated further improvements in the computational cost of sequence alignment. The most-widely used of these methods focused on the increasingly prevalent problem of aligning short reads to reference genome type sequence databases. Due to the high quality of the reference sequence, low error rate of the short ($\leq$100bp) reads, and low nucleotide diversity of humans (where $\theta \approx 10^{-3}$), algorithms that focused on exact string matching had great success. Much like BLAST and BLAT, the first wave of aligners capable of indexing the human reference genome and aligning short reads to it utilized exact k-mer matching via hash tables followed by local alignment [162, 148, 160]. Substantial improvements would be yielded by the development of aligners based on contemporary developments in compressed data structures.

1.2.2.1 Compressed full text indexes

The suffix tree [287] encodes all suffixes of a sequence S in the structure of a tree such that the suffixes may be enumerated by a depth first search (DFS) of the tree. This structure can be used to determine if a given sequence $q = c_1 c_2 \ldots c_{|q|}$ is present in S in $O(|q|)$ time. Search begins at the root, progressing across the topology (edge or node) of the tree which is labeled with the next character until no further matches may be found. By labeling the tree with the sequence positions corresponding to each node, the search may also yield the positions of the exact matches detected within S. Suffix trees may be built in linear time and space relative to their input [278], and support diverse algorithms for string comparison [10], such as whole genome alignment [61], but they require relatively large amounts of memory per input base. They were superseded by equivalent data structures with better memory bounds such as the suffix array, which represents the lexicographically ordered suffixes as a vector of numbers [176], and its compressible sibling the Burrows-Wheeler Transform (BWT) [28]. Compressed suffix arrays (CSA) (equivalently, the "fast, minute" FM-index) are data structures which combine a compressed representation of the BWT with auxiliary data structures that support rank and select operations on it [77, 81, 103]. To support suffix array operations in this compressed context, including pattern matching and positional queries, standard implementations include additional auxiliary information, in particular a sampled subset of the entries in the suffix array. See section 2.4.3.2 for a detailed review of the important features of these data structures as they relate to string matching.

The FM-index is used in modern short-read aligners such as BWA [158] and BOWTIE [145], which apply a backtracking search algorithm to directly align sequences to the

suffix array encoded in the FM-index. This approach is fast, but has problems detecting indels (as these require exponentially more backtracks to infer) and performs less well with increasing read length. In response, the authors merged initial exact matching with a final DP step to yield "long read" capable aligners like BWA-SW [159] and BOWTIE2 [144]. Further refinements of this concept yielded BWA MEM [153], which uses a heuristic algorithm to determine "supermaximal exact matches" (SMEMs) and reseed "sub matches" within them using a bidirectional FM-index (the FMD index). Due to its relative robustness to error and variation, the MEM concept has ultimately prevailed and as of the time of this writing BWA MEM can be seen as the industry standard method for aligning short reads to the genome.

Much of the second generation sequencing data has been generated for humans in medically-motivated genome wide association studies [49] or population survey projects like the 1000 Genomes Project (1000GP) [1, 45]. The development of these methods was accelerated by an open, competitive spirit fostered during the 1000GP, whose primary sequencing data remains the largest completely publicly available data set, with more than 100TB of sequence data available for download from public URLs without any authentication. There, project participants formalized the resequencing process by generating a series of data formats linking the various stages of analysis, including the sequence alignment/map format (SAM) and its binary equivalent (BAM) [161] that is the standard output format for contemporary aligners.

1.2.3 Variant calling

DNA sequencing reads of all types contain errors, and genomes contain diversity. To resolve these errors and infer the genome's state, we aggregate information from many reads mapping to each locus. In the context of resequencing, this process is known as *variant calling*. The simplest methods resemble the consensus step in OLC assembly, and are implemented as heuristic filters on the mutually gapped alignment matrix of a set of homologous sequence reads [138]. A Bayesian model can incorporate prior expectations about the genomic state with the available data to generate a posterior estimate of the probability of polymorphism that can propagate uncertainty to downstream analyses. It can use first principles to integrate various sources of information in addition to the sequence of the reads themselves, including the base quality (BQ), or machine-estimated probability of an erroneous base call, and mapping quality (MQ), which represents the aligner's estimate that the given alignment is a mismapping or ambiguous [151]. A Bayesian approach also supports the joint analysis of many individuals from the same population. For instance, in a panmictic population under neutral selection the pattern

of observed genotypes should be consistent with Hardy-Weinberg Equilibrium (HWE), and to have confidence in a given genotyping call, the evidence for variation should be stronger than the prior odds of there being no genetic variation at the site.

The earliest implementations of Bayesian variant calling and genotyping were applied to expressed sequence transcripts (ESTs) [178]. Competition fostered by the 1000GP encouraged the development of variant calling algorithms based on a variety of principles. The simplest methods would detect variation given pointwise SNP and indel descriptions directly from the alignments [161, 62]. However, this technique was shown to be susceptible to inconsistencies in the alignment process, and several groups developed methods that would reevaluate the alignments in a reference-independent manner in order to homogenize the representation of small variation. These techniques became known as "local assembly" variant detection algorithms, and include the windowed haplotype detection implemented in freebayes [91] as well as full local *de novo* assembly based on de Bruijn graphs as implemented in Dindel [4], Platypus [230] and the GATK's HaplotypeCaller. In parallel, several whole genome *de novo* assembly methods, including SGA and Cortex, were applied to the full data set, yielding variant calls that minimized bias towards the reference genome. The final project results were merged into a population genome assembly using statistical phasing algorithms [27, 112, 60] guided by genotyping results from sequencing and genotyping arrays [45]. Members of the 1000GP also developed a file format for describing collections of resequenced genomes, including their genotypes and inferred haplotypes, the variant call format (VCF) [53], which has become the standard interchange format for sequencing-based variant and genotyping information.

Due to the absence of a reliable truth set, early variant calling method implemented conceptually-derived inference methods rather than machine learning techniques. Subsequently, projects at Illumina (Platinum genomes) and NIST (Genome in a Bottle) have generated "truth sets" for variant calls matched to cell lines for which large amounts of sequencing data is publicly-available [69, 299]. These truth sets have then enabled the development of "universal" variant callers using machine learning techniques [218][8]. It may be expected that this trend will continue as the number of highly accurate independently sequenced genomes increases.

1.2.4 The reference bias problem

Short reads are insufficient to generate *de novo* assemblies of reference quality, and this issue is exacerbated when they are used in resequencing, as the prior information

[8] Along with Nicolás Della Penna, I developed a similar but much simpler method based on a linear learner: https://github.com/ekg/hhga

provided by the reference is relatively strong and can distort our results [266]. Most aligners operate on the principle of matching each sequence read to the linear reference, and differences between the read and the reference induced by both error and variation will tend to reduce the success of mapping. As I will demonstrate later in this work, reference bias is most severe for larger variants. However, the bias towards the reference is relevant even for SNPs, a fact which adds great complexity to experimental contexts that are sensitive to slight changes in allele observation count, such as allele specific expression (ASE) quantification from RNA sequencing [263], or in the context of short and high error reads as are common in the sequencing of ancient DNA [297].

Advances in sequencing technology can reduce reference bias in some contexts where long reads can be obtained. Long reads can overlap structural variants that would contain shorter reads, allowing their direct discovery by alignment. However, costs of second generation sequencing continue to drop, so it seems likely that there will continue to be a cost advantage to resequencing with short reads for the near future. Nonetheless, reference bias remains relevant even in a future in which all sequencing is completed with long, low-error reads. As long as the reference is used as a basis space for analysis, it will be impossible to develop unbiased representations of all sequences in a given cohort. We cannot consistently describe variation in sequences which are not in the reference unless we bring these sequences into communication with each other. It is non-trivial to establish if structural variants independently described against the reference represent the same allele [34]. We can use improvements in assembly methods, such as the linked DBG [277], to build space-efficient joint assemblies of populations of genomes. But these approaches are unlikely to improve in efficiency by the many orders of magnitude required to consider applying them directly to sequencing from hundreds of thousands or millions of genomes.

1.3 Pangenomes

Following the completion of the 1000GP, researchers have sought to use the population reference established by that project as an input to genome inference processes. Rather than establishing a single linear reference genome, these methods base their analysis on a representation that contains some or all of the known variation in the species of interest. In these approaches, the reference system becomes a *pangenome*[9], or data space representing all the genomes and their interrelationships. The term was first used to describe the sequence information obtained from DNA and RNA for a cancer

[9] "pan-" from Greek $\pi\alpha\nu-$, meaning "all" or "every"

sample [250], but later became an important concept in microbiology as results from bacterial genome sequencing indicated extensive diversity between bacterial genomes [272, 182]. Due to horizontal gene transfer (driven in large part by the permissive sex lives of bacteria), mobile DNA in the form of viruses and transposable elements, and their enormous population sizes, the genome diversity of many prokaryotes is much greater than that seen in larger, complex organisms. In microbial pangenomic theory, the main object of interest is the open reading frame (ORF) and its distribution across species in a clade [282], with particular interest to classification of ORFs or genes into a gradient between those that are essential and found in every species (the "core" pangenome) to those that are found infrequently (the "dispensable" pangenome). The term "pangenome" is by no means microbiology-specific, and has also seen use in species contexts where small, homozygous genomes support practical direct whole genome comparison, such as *Arabidopsis thaliana* [31]. With reducing sequencing costs, the levels of diversity in eukaryotic genomes can be more easily appreciated, and in the 2010s evidence has rapidly accumulated that significant levels of large-scale variation occur in the genomes of many species, humans [163, 265, 266, 34], arabidopsis [7], brewer's yeast [292], and the fruit fly [35].

Evidence that non-reference genomic variation matters even in a human or medical context motivated extensive discussion within a sub-project of the Global Alliance for Genomics and Health (GA4GH)[10]. At the beginning of my studies I participated in the GA4GH's reference variation task team (RefVar), which was led by Benedict Paten, David Haussler, and Richard Durbin. The group had regular meetings where its members entertained proposals for new variation-aware genomic data models and discussed results obtained with software implementations of them. By chance, a meeting of the GA4GH in June 2015 in Leiden overlapped a conference held at the Lorentz Centre on "Future Perspectives in Computational Pan-Genomics"[11], whose participants were discussing ways to apply the concept of pangenomics to many problems in genomics. Can Alkan, who had been invited to both meetings, brought members of the GA4GH's RefVar group to the concurrent workshop, where both groups presented on their work and ultimately joined efforts. This exchange motivated members of the RefVar group to consider many alternative resequencing and genome modeling problems. For the consortium, our software `vg` became a template for the pangenomic resequencing concept that it would present in the paper resulting from the meeting [46] (figure 1.3). And in turn, the consortium imagined the missing pieces that would be required to fully enable a

[10]The GA4GH is an international consortium of researchers and genomics professionals chartered with the development of new genomics data formats and interchange systems https://www.ga4gh.org/.

[11]https://www.lorentzcenter.nl/lc/web/2015/698/info.php3?wsid=698&venue=Oort

pangenomic reference system and support common genome inference patterns using it. Much of the work I will present in chapters 2 and 3 of this thesis follows the design presented by this group.

1.3.1 On pangenomic models

My own work builds on a particular model for encoding a pangenome. Here, I will briefly describe alternative models and justify the use of the graphical one that I present, while the remainder of the chapter will provide background on more-closely related graphical approaches more closely related to my work.

Traditional techniques from microbial pangenomics have focused on cataloging the distribution of ORFs across bacterial species [209]. In this sense the pangenome is not so much a sequence-based object, but a matrix encoding the presence or absence of genes across the species of a given clade.

If we want to use pangenomic principles to resolve issues with resequencing, then we must take the concept of pangenome more literally, and build a representation that losslessly encodes genomes together with a focus on their sequence content. In this perspective, the classical bacterial pangenome becomes a derivative product that we can produce using analyses based on a sequence-oriented pangenomic reference. I will mostly focus on sequence-based pangenomic models. The main classes are described visually in figure 1.4.

The simplest possible sequence-aware pangenome is just a set of whole genome sequences of many species or individuals, in which all sequence homologies and evolutionary relationships are implicit (figure 1.4A). The unfolded pangenome resolves reference bias, and can be extended with new data by simply including new genome sequences. This model does not benefit from compression related to shared sequences in the pangenome, as adding a new genome always adds all the sequence in that genome to our system. Without additional information about homologies, an unfolded pangenome cannot represent new sequences in terms of recombinants between known sequences.

An MSA (figure 1.4B) provides a matrix describing the relationships between the sequences as well as the sequences themselves. We must introduce a concept of a gap character to pad the matrix. The MSA is a linear object, and cannot represent structural variation compactly. An MSA has an equivalent representation as a sequence DAG, as in figure 1.4D.

Assembly graphs, in particular DBGs (figure 1.4C), provide a simple decomposition of collections of genomes. However, a strict DBG without any labeling loses the mapping back to the original genomes. Sequence graphs, as in figure 1.4E, when annotated with

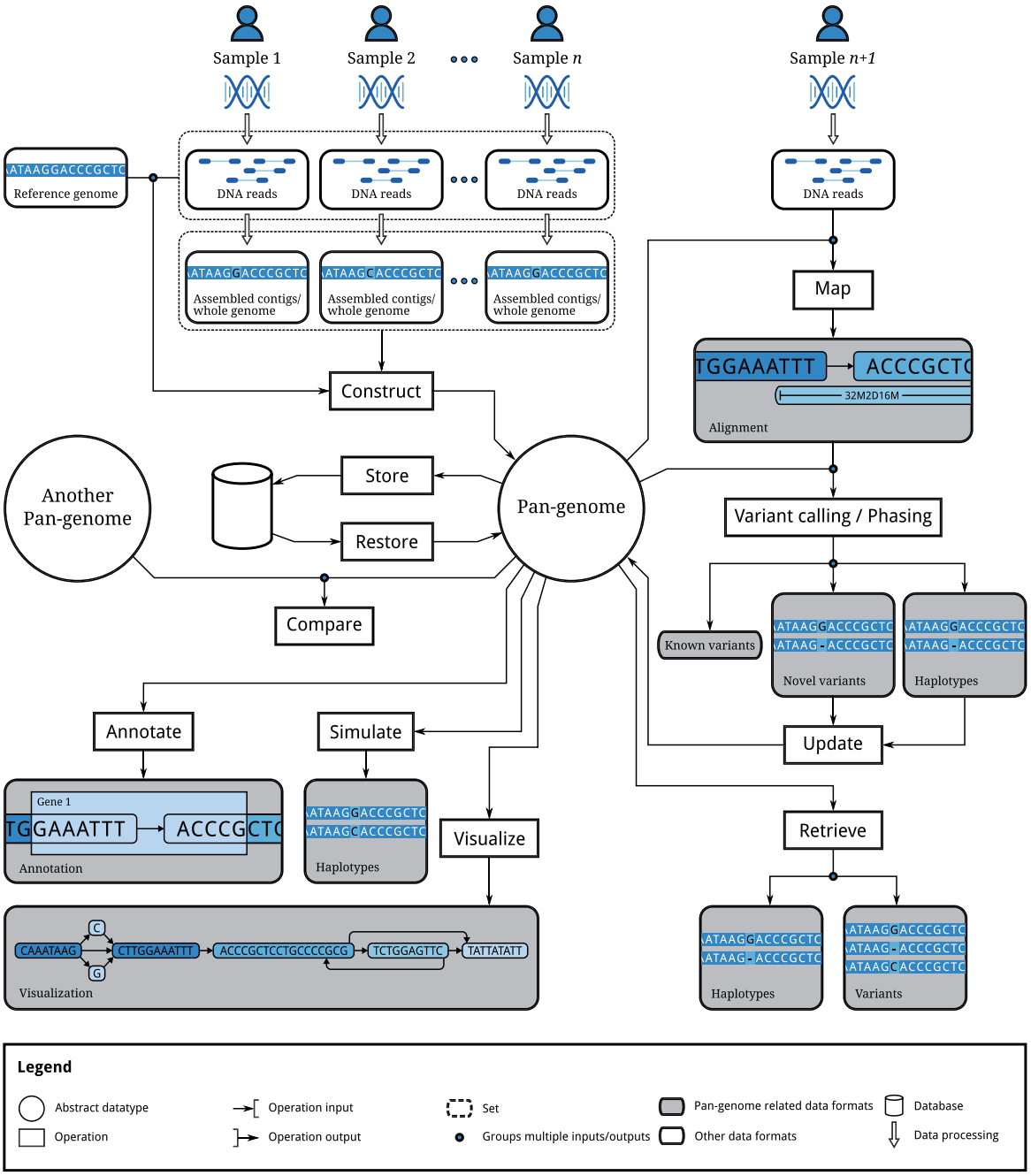


Fig. 1.3 An overview of techniques required to support pangenome-based resequencing. In vg, we have implemented virtually all the presented components and algorithms. Reprinted from [46].

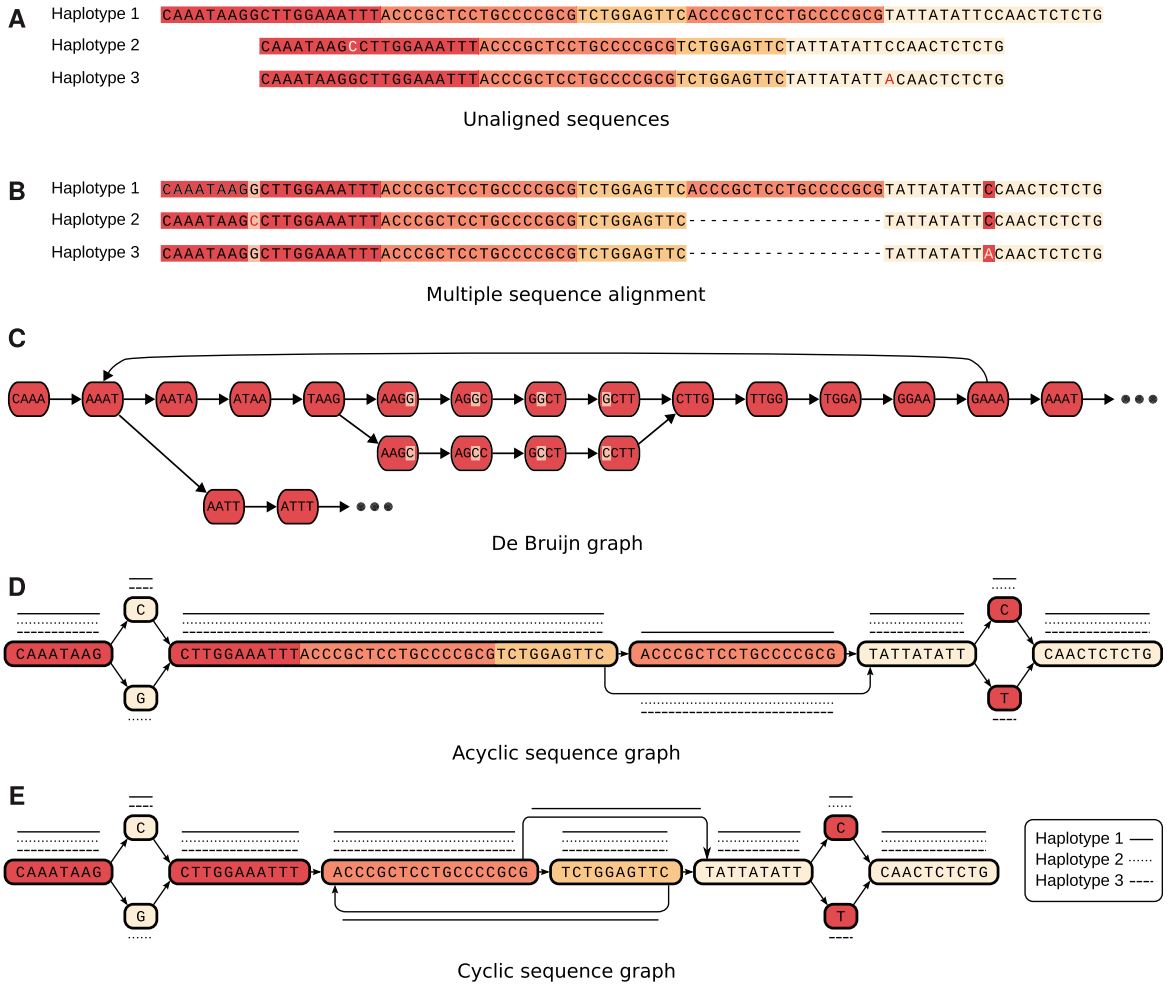


Fig. 1.4 Various pangenomic models of a small collection of sequences. (A) shows an unfolded pangenome, (B) provides an MSA encoding the same sequences, (C) shows the DBG with $k = 3$, (D) is an acyclic version of the pangenome, akin to the alignment in (B), and (E) represents a compressed alignment graph allowing cycles to represent copy number variants. Reprinted from [46].

.

the full set of input paths, provide a lossless representation of the input genomes. If they are also bidirectional like DBGs (not shown in figure 1.4E) then they can directly represent copy number variations and inversions.

1.3.2 The variation graph

In this thesis I employ a reference system that encodes genomes and the base-level relationships between them. This model can be understood as a kind of all-versus-all alignment between the sequences in the pangenome. If the data model is to allow recombination between known sequences (a key contributor to genomic diversity) and tandem repeat copy number variation (which occurs readily in genomes), then it can be represented as a regular language encoded in a graphical model like a nondeterministic finite automaton (NFA).

We can adjust the regular language model slightly so that it has properties similar to those of DNA. NFAs are represented graphically with states (in our case, pangenomic coordinates) as nodes connected by edges labeled by characters (e.g. DNA bases) in the alphabet of strings that the language recognizes. In DNA the atomic element is the DNA base, which is represented as a character. Graphical models with nodes (or edges) labeled by sequences and edges (or nodes) representing allowed transitions between them are a straightforward generalization of the linear string. Compression can be achieved by allowing the labels on the nodes to have more than a single character on them. Because DNA is double-stranded, any such a language implies a reverse complement language which recognizes the reverse complement of any sequence in the first. Formalizing this by allowing edges to transition between different strands of the graph allows the model to directly represent sequence inversions.

If we combine these adjustments, we arrive at a kind of regular language model that resembles DNA and allows the representation of collections of related sequences by allowing us to represent homologous sequences and all kinds of natural polymorphism between them. This structure is often referred to as a *bidirectional DNA sequence graph*, indicating that the graph is sequence-centric, directed, stranded, and allows transitions between strands. *Sequence graph*, then implies a simpler concept in which the graph is meant to model sequences but only one strand is considered. Assembly and multiple sequence alignment methods have employed sequence graphs of both types since the earliest computational analyses of biosequences, and it is sensible that we might employ them to represent collections of genomes.

The conceptual basis of the work I present here is the extension of the bidirectional DNA sequence graph with *paths* that may be used to describe sequences as walks through

the graph. In this way, the panel of reference genomes or sequences used to construct the graph may be related to the graph itself, and the relationships between them made evident in the structure of the graph. Existing knowledge expressed with respect to known sequences may thus be projected into the graphical pangenome model. Similarly, entities within the graph may be projected out into the space of a given path (as described in section 2.5.10). These properties ensure that the pangenome fully encompasses existing reference technologies. In addition, maintaining the sequences in the space of the graph makes the graph lossless, in that it fully represents the input sequences without additional information. Perhaps most importantly, this feature resolves the exponential decay in mutual information which limits the applicability of Markovian models like the sequence graph to modeling natural sequences [166]. I term this combination of a bidirectional sequence graph and paths a *variation graph*, as it represents sequences and the variation between them. In chapter 2 I will formalize the variation graph model and important auxiliary data structures that enable its modification and use in resequencing.

1.4 Graphical techniques in sequence analysis

Many genome analysis algorithms employ sequence graphs. I introduced the alignment algorithms described in 1.2.2 in terms of a matrix, but they may also be described as algorithms on graphs, although the nodes in these graphs correspond to matrix cells and thus alignment states rather than characters or sequences. Similarly, hidden Markov models (HMMs) have a long history of use in bioinformatics [66], and these models bear similarities to the bidirectional sequence graph model. Here, I will focus on those methods that are most closely related to variation graphs, and upon which my work draws most heavily. These include techniques for generating and encoding multiple sequence alignments, genome assembly graphs, RNA splicing graphs, and the related gene model graphs used in RNA sequence analysis, and the sequence DAG implied by the VCF format.

1.4.1 (Multiple) sequence alignment

Optimal multiple sequence alignment generalizes the problem of pairwise sequence alignment from a 2D matrix to an N-dimensional lattice, where the optimal mutual alignment of N sequences of average length L can be determined in $O(L^N)$ time [32]. In the early days of sequencing, when only a handful of sequences might be considered in one analysis, such costs were almost acceptable, even if they limited the number

of sequences in the MSA to only 3. By pruning regions of the lattice in which no optimal alignments could occur, the authors of the tool "MSA" increased the number of sequences which could be optimally aligned into an MSA to 6 [168]. In contrast to the optimal alignment approach, progressive multiple sequence aligners such as CLUSTAL build a guide tree based on alignment of all sequences to all others in $O(NL^2)$ time, and then generate the MSA progressively using the guide tree in $O(L^2 \log N)$ time, resulting in polynomial time algorithms capable of generating MSAs for hundreds of sequences using contemporary computers [110]. In this form of progressive alignment, the MSA is built recursively from the leaves to the root of the guide tree, with each step combining pair of MSAs representing different branches of the tree using the best pairwise alignment between the sequences they contain. The progressive approach is fundamentally greedy, and susceptible to errors that propagate along the guide tree, although such errors can be mitigated by structuring the alignment using biological priors [273]. Further improvements to the quality of the MSA can be gained by guiding the progressive alignment with a limited kind of global information about the relationships of all the sequences, as in T-COFFEE [201], but this popular method exhibits a worst-case computational complexity of $O(N^3L^2)$.

The progressive alignment in MSA algorithms like CLUSTAL may be represented graphically. In the late 1980s Eugene Myers and Webb Miller developed algorithms to optimally align sequences to sequence graphs, and sequence graphs (in the form of regular expressions) to each other in $O(MN)$ time (where M and N are the sequence length of the graphs) [197, 291]. Unfortunately, to my knowledge these were never implemented in publicly available software for sequence analysis[12]. A recent implementation of sequence to graph alignment [227] transforms a sequence graph into an alignable graph which is acyclic and partially ordered, on which a bit-parallel alignment algorithm is applied to achieve high performance when aligning long noisy reads to arbitrary sequence graphs. This work is related to that of Wu, Manber and Myers [291], wherein the authors provide an algorithm for the alignment of pairs of regular expressions using a transformation of the regexes to NFAs and bit-parallel resolution of the final alignment. Later in chapter 2 I will present and evaluate a similar approach to align reads to arbitrary bidirectional sequence graphs through transformation of the graph into an ordered graph against which accelerated sequence to graph alignment may be run.

Christopher Lee later provided the first implementation of an MSA algorithm based on sequence to graph alignment [147]. Apparently unaware of the work of Myers and Miller, he instead built on the concept of "consistent equivalence relations" that DIALIGN used

[12] Myers was unable to provide related source code on request.

to represent the MSA [190]. Where DIALIGN's authors appear to consider the MSA in the space of the N-dimensional lattice, Lee encoded the equivalence relations in a partial order graph, which is often referred to as a directed acyclic graph (DAG). In this DAG, characters label nodes and edges label observed linkages between them in sequences embedded in the MSA. Lee demonstrated that a straightforward generalization of the recurrence relations used in Smith-Waterman-Gotoh would allow the alignment of sequences of length N to the DAG of sequence length M in approximately $O(MN)$ time. To determine the score at a given position, partial order alignment (POA) considers matches and deletions relative to all the characters that immediately preceded the current one in the partial order. Later, POA was extended to allow the direct alignment of pairs of MSAs using partial order to partial order alignment (PO-POA) [100]. Like CLUSTAL-W and other progressive methods, POA would build its MSA using pairwise alignments across a guide tree. But, rather than aligning the last pair of sequences, PO-POA alignment would be used to align the MSAs from each branch together. This resolved problems with order dependence, yielding MSAs with nearly the same accuracy[13] as T-COFFEE across a range of problems. As PO-POA proceeds over a neighbor joining guide tree, it requires $N \log N$ alignment steps. Given low sequence divergence, the cost of each step will approximate L^2. The algorithm thus has a lower bound of approximately $O(L^2 N \log N)$, which the authors confirmed with experiments demonstrating subquadratic scaling in the number of input sequences.

Despite its use of an algorithm that scales cubically with the number of input sequences, T-COFFEE's higher accuracy has resulted in it receiving ten times the citations of POA. The low rate of use meant that the POA concept was "rediscovered" by participants in the 1000GP who needed a computationally inexpensive method to align sequences to VCF-based pangenomes.[14]

Lee's POA model provides a simple pattern for thinking about pangenomes. However, POA MSAs are linear objects which cannot capture many natural kinds of genetic variation such as repeats or rearrangements without duplication of the rearranged sequences. Several methods have extended the MSA concept to unordered graphs, including the Threaded Blockset Aligner (TBA) [21], which models the MSA graph as a set of partially ordered MSAs linked by unordered larger scale connections, and the A-Bruijn Aligner (ABA), which models the MSA using a de Bruijn graph and represents

[13]Here goodness is quantified using a sum of pairs score (SPS) metric representing the goodness of the alignment.

[14]Sequence to graph DAG alignment was one of Deniz Kural's main PhD projects. I worked with him in Gabor Marth's laboratory, and we applied his implementation of POA to generate accurate genotype likelihoods for indels and complex variation in the final phase of the 1000GP. We learned of the prior work later, when a reviewer pointed out that the algorithm was roughly equivalent to POA.

a solution to problems in MSA using techniques that later become important to short read assembly [225]. Variation graphs generalize these models without the limitations of order (as in POA and TBA) or k-mer based graph structures (as in ABA).

The POA model also inspired the development of a graphical model that supports the comparison of various editions or versions of the same text in the field of textual criticism, the *variant* graph [241, 106]. The variant graph can be understood as a POA DAG built from a collection of texts, whose nodes are labeled by the input texts that traverse them. *Variant* graphs are very similar to the *variation* graphs that I present in this thesis, but while variant graphs are specialized for linear written texts, variation graphs aim to model DNA and genomic variation. Although they have similar structure, these two models are in fact an unusual instance of interdisciplinary convergent naming.[15] Variant graphs are used mostly to reduce the effort required to collate and manually review different versions of texts. Recent work has focused on their visualization, resulting in techniques that are very similar to those developed for variation graphs which I present later in this thesis [124].

1.4.2 Assembly graphs

The problem of assembling large genomes is not dissimilar from that of multiple sequence alignment. MSA algorithms tend to be applied to the alignment of a single coherent genomic locus. Their input sequences might be expected to have approximately the same length, and maintain synteny between them. In contrast, whole genome shotgun assembly methods cannot rely on such assumptions, as reads of a large genome rarely overlap, and both strands of DNA will be sampled, yielding ambiguity about relative orientation. Thus, the graphical models used in assembly must maintain bidirectional structure, which distinguishes them from those used in multiple sequence alignment. Earlier in section 1.1.2 I gave a historical outline of the development of these methods in response to changes in available sequencing technology. Here, I provide deeper technical detail to describe these methods and illustrate their relationship to the variation graph model.

[15] I developed the term "variation graph" without knowledge of this prior work. I first become aware of it in the week before the initial submission of this thesis, when Daniel Bruder brought textual variant graphs to my attention. In practice, some researchers use both terms to refer to the model I present in this thesis.

1.4.2.1 Overlap graphs

One interesting feature of overlap-based assembly graphs is that their efficient construction tends to yield a representation in which sequences attached to nodes partially overlap with the nodes in their neighborhood in the graph[16]. This follows from the fact that the graph induction algorithms use a kind of pairwise alignment, retaining relationships between sequences in a pairwise rather than compacted N-wise form. In the case of DBGs and the FM-index based string graph assemblers, an iterative overlap-wise sequence comparison where unitigs (unbranching sequences in the graph) are inferred from the compressed sequence graph yields overlap graph. Given a node-labeled sequence graph, it is common to think of the edges as representing the overlaps between the nodes they connect. This represents an incomplete compression of the relational information in the graph, but this is typically not important to the use of these methods. They are judged by the quality of the set of contiguous sequences (contigs) they output rather than their raw assembly graph. These methods will traverse linear portions of the graph to generate contigs, after pruning or ignoring edges, which the uncompressed overlap representation does not inhibit.

Due to its duplicated representation of sequences within overlaps, the overlap graph model is more complex to use as a reference system (in which positions should be unique) than a "bluntified" representation in which the overlaps are non-ambiguously reduced into the nodes in the graph and its linkage topology. In their most general form, overlaps are themselves alignments, and have a natural encoding in graph form (section 1.4.1). However, assembly data models often encode alignments using several dissimilar data structures, a fact which is reflected in the complexity and redundancy of the GFA version 2 specification[17]. In general this overlap representation makes it difficult to work directly with such graphs using the algorithms we will introduce, and they must be reduced into a "blunt-ended" bidirectional graph.

It is also possible to directly induce a bidirectional string graph from a set of pairwise alignments, sidestepping the overlap issue. In section 2.2.6 I will present my design and implementation of an external memory algorithm that transforms sets of pairwise alignments into a variation graph using a transitive closure of the equivalencies implied by the alignments.

[16] In many formulations, sequences in the overlap graph are attached to edges rather than nodes. This model is equivalent to one in which sequences are attached to nodes, which I will use here for consistency. The same convention is used in `vg`'s data model and in the GFA interchange format that it reads and writes.

[17] https://github.com/GFA-spec/GFA-spec/blob/master/GFA2.md

1.4.2.2 De Bruijn graphs

De Bruijn graphs [56] are graphs in which a set of k-mers are taken as the nodes of the graph, and edges are added for each pair of k-mers $k_1 \rightarrow k_2$ in which the last $k-1$ bases of k_1 are the same as the first $k-1$ bases of k_2. As discussed previously in section 1.1.2, this model simplifies the overlap graph structure, allowing efficient calculation and representation of the graph. For instance, k-mer lengths may be chosen so that they fit inside a machine word, allowing bitwise operations and integer math rather than string comparison to be used to infer the graph structure directly implied by the k-mers themselves.

In second generation sequencing, where per-base error rates are low and read lengths are short, little information is lost by breaking the read set into k-mers of 1/3 or 1/5th the length of the original reads, and so the de Bruijn graph model has been readily applied to cheap short read sequence data since its introduction to genomics in the mid 1990s and early 2000s [117, 217]. Velvet [294], which used a straightforward but memory-costly hash table strategy to encode the DBG. Zamin Iqbal then extended the DBG model to support various kinds of pangenomic analysis by labeling each k-mer with "colors" representing read counts from different samples [120]. The colored DBG (cDBG) model has seen application in RNA-seq transcript quantification, where the model is used as a reference basis for the relation of known transcripts to pseudoalignments of reads to the cDBG [25]. Later versions of the Cortex assembler have extended this to fully encode long reads or contigs relative to the DBG [277]. Similarly, improvements in performance have been yielded by linking read pairs in the DBG model [13].

The simplicity of the DBG has made it possible to develop very memory-efficient data models to support its use in assembly. The DBG can be effectively encoded in the FM-index of a read set [23], and this *succinct* DBG model underpins the most-scalable assembly methods currently available [150]. Other techniques, such as bloom filter encodings and minimizer partitioning schemes are also used to provide time and space efficiency to DBG methods [39, 40].

In the compacted DBG generalization non-furcating regions of the graph are merged into a single node with label length $> k$. The compacted DBG is now often a typical output for DBG assemblers [40, 187]. The compressed nodes and their overlaps with their neighbors comprise a set of unitigs that, together with their neighbor relations, are often taken as the most-raw kind of assembly output. DBG assemblers like SPAdes and Minia3 infer longer contigs by filtering and further contracting the unitig graph [13].

As with any overlap graph, DBGs must be made into blunt-ended sequence graphs before they can be utilized by variation graph based algorithms. The basic method for

doing so is simpler than for generic overlap graphs, as overlaps in DBGs are exact string matches. In my work I have found this an important feature, as in addition to being generated by efficient methods, it ensures that DBGs are universally convertible into variation graphs.

1.4.2.3 String graphs

The string graph is a formalism that describes the full information represented by a shotgun sequencing experiment and an all-against-all alignment between its reads [193, 194]. Myers argued that the then-current paradigm of assembly, which attempted to generate the shortest common superstring (SCS) incorporating all the N input sequence reads, failed to reconstruct the genome correctly in the context of repeats in the genome that are longer than the average read length L. He then posited the "chunk graph" (later, string graph) as a graphical model of the overlap set, showing that the correct consensus sequence would by definition exist as a walk through the graph, and that constraining the collapse using coverage information would improve reconstruction of the genome read in the shotgun sequencing experiment. In this graph nodes (or edges) represent sequence reads and directed edges (or nodes) represented observed ϵ-approximate overlaps between them. Repeat units in the genome that are longer than L will collapse in this graph, provided errors in the reads can be corrected so such repeats become fully identical.

This idea was introduced at the same time as de Bruijn graphs, with Myers' work on string graphs and the first description of a genome assembly algorithm using de Bruijn graphs both presented at the same workshop in 1995 [193, 117]. Myers' 2005 formalization of the string graph [194] responds particularly to de Bruijn models, and he points out that generating k-mers from the reads as the basis for the graph means that the resulting graph is not "read coherent", or in other words does not accurately represent the information in the read set. Subsequent work has shown that the boundaries between the two models are not so well-defined. As a specialization of the overlap string graph, the de Bruijn model may be applied to certain complex subsets of a string graph, creating a kind of hybrid assembler where the k-mer model is used to resolve the most-difficult components, while the generic overlap model is used elsewhere [115]. Similarly, the high cost of error and graphical complexity suffered by the string graph encourage the use of k-mer based read correction methods, which could be seen as filtering the reads using a DBG model.

String graphs are often described as "lossless" representations of the input read set and the alignments between them [152]. Neither in practice, in Myers' formalizations, nor its implementations like the Celera assembler (CABOG) [186] is this strictly true. In the

model, overlaps are assumed to be $\epsilon-$correctable at approximately the raw sequencing error rate. In practice, this filtering can result in a loss of input sequence from the string graph. String graphs can consume very large amounts of memory when fully constructed without filtering from a read set [156, 141]. Input filtering, mostly to reduce repeat content, is used to mitigate this issue. If not aggressively corrected, repeats tend to generate ultra-dense graph regions that are known as "hairballs" which can increase the complexity of assembly by orders of magnitude.

To deal with repetitive sequences in the genome, one solution is to mask out repetitive k-mer, minimizer, or alignment seeds used in generating the overlap set, as in CABOG, FALCON, and miniasm [186, 41, 156]. Recently, the authors of Canu showed that alternative probabilistic seed filtering based on the *tf-idf* (term frequency, inverse document frequency) metric can retain information about repeats and support their separation rather than excision from the assembly string graph [141].

String graph methods related to the Celera assembler, such as FALCON and Canu, implement error correction steps before overlap inference, as this reduces memory requirements during assembly. Similarly, methods that generate assemblies via a string graph induction step from Illumina sequencing data (SGA, fermi, and fermikit) also apply an error correction processes before the generation of the string graph [254, 252, 155], which helps to reduce the graph complexity and improve contiguity of the resulting assembly.

Allelic diversity in either a string graph or de Bruijn graph will, if sufficiently separated from repeats and other allelic variants, result in a bubble, or graph component connected to the rest of the graph via single sources and sinks. Like de Bruijn graphs, string graphs have been used to support variant calling, for instance finding heterozygotes represented as bubbles in the assembly graph [152]. All methods that I am aware of will establish variants relative to a reference sequence threaded through the graph. Also, thus far no method has specifically merged the two concepts of variant calling and graph based assembly finishing together, although assembly projects may use a variant caller to establish variants in their contigs [122, 240].

With minimap and miniasm Heng Li took the approach of efficient all versus all alignment and overlap graph generation without prior read correction [156]. With no multiple alignment step, this generates an assembly in which the error rate in the contigs approaches that of the input reads. Tools like racon [280] have been developed to subsequently generate a consensus, but these work on the level of individual contigs rather than an assembly graph. Li [156] also proposed to mix and match different assembly components (for instance using DALIGNER [195] rather than minimap for the overlap step, or swapping quiver for nanopolish for consensus generation) by establishing

a set of standard data types including the pairwise alignment format (PAF) and the graphical fragment assembly (GFA). These encode the results of the overlap step and a graphical model for the assembly at any state of its progression. GFA is used in a wide number of methods, but as of August 2018 it appears that very few tools both read and write GFA[18], and much of the assembly improvement steps are implemented on contigs (encoded in FASTQ or FASTA format) rather than the string graph itself.

1.4.2.4 RNA sequencing graphs

While many approaches to transcript analysis consider each particular gene transcript separately, a more compact representation would be as a *splicing graph* in which all alternative splicing junctions are represented by edges connecting bases of the underlying reference sequence [109, 147]. Transcript assembly has attracted many of the same approaches as those used in genome inference, but their application must be adjusted to account for variation in read coverage of several orders of magnitude caused by large differences in expression of different genes [179], and in some cases they may not be suitable as the ideal of a transcript assembly is not a linear sequence assembly, but a splicing graph that models the combinatorial relationships in the transcriptome [99]. Standard assemblers, parameterized or modified to better support RNA transcript assembly have been applied to the problem since their development for genome assembly [20, 231, 245]. But methods which are specifically design to meet the needs of the problem have arguably been more popular [99, 37].

Direct use of the splicing graph concept is limited, with the most-popular workflow involving "splice-aware" alignment to a reference transcript model followed by quantification of expression versus the alignment and transcript model [274]. More recently, probabilistic approaches have come into favor, and to support these assembly models have been applied to transcript quantification via pseudoalignment to a colored DBG annotated with reference transcripts [25]. In contrast, Daewan Kim's HISAT2 aligns RNA-seq reads to a whole genome splicing graph [136, 137], including SNP and indel variation directly in the whole genome index.

1.4.2.5 Genome alignment graphs

The graphs used in genome alignment algorithms are the nearest in content and structure to variation graphs, and it can be shown than a number them of nearly identical representational capacities [131]. The *alignment graph*, first introduced in the context of

[18] https://github.com/GFA-spec/GFA-spec#gfa-1

multiple sequence alignment [129, 226], represents a collection of alignments in graphical form, supporting the induction of a sequence graph (or MSA when such a graph is partially ordered). Vertices in the alignment graph $G_a = (V, E)$ represent characters in the input sequences S, and edges represent cases where characters in the input sequences have been aligned, with an additional relation $\prec$ on V such that $v \prec w$ holds if and only if v precedes w in S. As it contains both the sequences and their alignments, this graph may then be contracted to produce a compressed graph that represents both. We can then contract G_a into the base graph G_s, adding a node for each connected component in G_a and labeling it with the corresponding character, while adding an edge for each pair of nodes X and Y which represent a pair of input characters in G_a that satisfy $\prec$. Each column of an MSA matrix or each base in a variation graph would correspond to a particular connected component in G_a. It is worth noting that the equivalence is not exact unless we generalize the alignment graph to represent alignments in both the forward or reverse complement orientations, which has been explored in the A-Bruijn model [225].

The Enredo graph model $G_e = (V, E)$, used by Benedict Paten's Enredo/Pecan multiple genome aligner [210], generalizes the alignment graph model to be bidirectional by representing each genome segment in the graph with two nodes, a head and tail. The graph maintains two kinds of edges, E_s which represent genome segments and are equivalent to nodes in G_s, and E_a which represent breakpoints of adjacencies between segments. To obtain a graph like G_s from the Enredo graph, the MSA resolver Pecan is applied to sets of homologous E_s connecting the same head and tail nodes.

Paten further refined multiple genome alignment by the development of the Cactus graph $G_c = (V, E)$ [211]. To construct the Cactus graph, we build a precursor graph G'_c by adding a node for each adjacency-connected component in the Enredo graph G_e and adding edges between nodes for each segment E_s whose head and tail lie in both adjacency components. To obtain the Cactus graph, we collapse three-edge connected components in G'_c into single nodes, yielding an Eulerian graph with a tree like structure. The nodes in the Cactus graph can be shown to correspond to a tree of *ultrabubbles* (graph components connected to the rest of the graph by one or two head and tail nodes) in the sequence graph from which it was constructed [213]. This forms the basis for the development of genotype models on top of arbitrary graphs.

Without additional labeling, these graph models are insufficient to fully reproduce their input, and although it may be implemented in corresponding software, existing models to do not clarify how this labeling is accomplished [131]. Variation graphs respond

to this issue by making the path embedding of the sequences in the graph explicit in the model.

1.4.3 Pangenomic alignment

As the review presented in this chapter shows, the idea of using pangenomic models as a reference basis is not new, and interest in the topic extends back to the earliest explorations of the multiple sequence alignment problem. But it was only in the last decade, as surveys of populations of genomes routinely produced sets of phased variant calls [169, 286, 31, 1, 45] that the idea of scaling up alignment to enable the alignment of short reads to populations of genomes gained traction. Numerous methods use some form of pangenome aware alignment. These have helped me to understand the problem and guided my work to a significant degree. Each has important limitations relative to more generic problems. For instance, some operate on sequence DAGs, and cannot easily be generalized to work on arbitrary bidirectional sequence graphs. Others implement a limited form of alignment that prevents their use for every kind of sequencing technology. Virtually all use the pangenome only as an additional source of information during alignment, and do not use the pangenome as the reference system itself.

1.4.3.1 Alignment to unfolded pangenomic references

As discussed in section 1.3.1, the simplest pangenomic model stores each genome sequence individually, without recording the implied homologies or relationships between the sequences. Such a model grows linearly with the number of genomes included, and in this way it does not benefit from compression provided by sequence relationships and shared evolutionary histories. Annotations can be provided for each genome and interlinked at a higher level of semantic relationships, as is done in major genome annotation catalogs like Ensembl genomes [135]. This expedient approach provides almost all the benefits of the kind of pangenomic models I present in this thesis, excepting that it cannot represent base-level sequence relationships within the semantic model without significant effort. Whenever a researcher uses BLAST or BLAT to search a large database of sequence they are interrogating this "unfolded" pangenome. This pattern could be thought of as the default in bioinformatics, and it is the starting place for many DNA-based analyses.

In terms of methods specifically designed to align large sequence read sets against an unfolded pangenome, few are currently under development. Of note, the CHIC aligner provides an initial implementation of such an idea [279] This method is based on an indexing strategy which uses Lempel-Ziv (LZ77) [298] parsing of the pangenome to

generate a "kernel" sequence encoding the pangenome that may be indexed and used by a standard short read aligner (the authors use BOWTIE2). By using this kernel, the seeding step is aware of alternative sequences embedded in the pangenome, but the local alignment step is ultimately run against a linear reference. As this approach aligns directly to the unfolded pangenome it is not able to correctly estimate mapping quality, as the reference does not encode any model about the relationships between the genomes that comprise the pangenome and consequently we cannot determine when multiple alignments match to homologs in different genomes or paralagous copies dispersed across one or many genomes. The linear growth in data scale can add algorithmic complexity when sequencing many genomes, and so it is understandable that experiments using the CHIC aligner only used up to 100 genomes at a time. By relying on the linear reference to report alignments, CHIC gains the ability to hook into standard resequencing workflows, but it loses the ability to describe variation in sequence that is not contained in the reference.

1.4.3.2 Alignment to tiled pangenomic references

A near-approximation of the unfolded model is one in which genomes are broken into small pieces in the construction of the model. These blocks or tiles can then be formed into sequence DAG by the addition of edges showing linkages between successive blocks [105]. These models present as a specialized kind of sequence DAGs[19], and can represent the whole set of sequences in a pangenome in a semi-compressed way using the same principles as in POA, DBG, or string graph models. Each tile represents a known haplotype in a given window of the pangenome. As new genomes are added to the structure, new tiles only need to be added where we observe a new sequence in a given window.

A tiled pangenome graph was employed by the pangenomic aligner, GenomeMapper [243], which was produced in support of the *Arabidopsis thaliana* 1001 Genomes Project (AT1001GP). As arabidopsis frequently selfs, individuals may have extremely low levels of genomic diversity, which in turn simplifies the process of genome inference [31]. At the same time, several percent of the reference genome is missing or highly divergent in various accessions (strains) [43, 293]. These conditions break standard short-read aligners developed for mapping Illumina sequencing data against low-diversity reference genomes. GenomeMapper models the pangenome using 256 basepair tiles. A k-mer index is used to seed local alignment of very short (<50bp) reads against the pangenome. The authors

[19]Known implementations appear to be unable to directly represent inversions and copy number variants in their structures, instead encoding them as if they were indels.

report an extension of the Needleman-Wunsch alignment algorithm that allows alignment against the graph wherein the graph traversal is converted into a tree of alignments by duplicating the alignment process at each furcation. This alignment algorithm is exponential in the average number of forks per sequence base, which may contribute to the observation that it runs hundreds of times slower than standard alignment methods on 100bp reads [170]. GenomeMapper does not have a graph-specific alignment format, and instead reports alignments against the genome to which each is most similar, which the authors call "reference free". So that the alignments may be used downstream in standard approaches, they may be projected against another chosen reference genome.

1.4.3.3 Alignment to graphical assembly models

In order to report their results relative to the reference, assembly methods require the ability to align the reference genome into the sequence graph they have generated *de novo* [120, 254], although this can be achieved through the alignment of the assembly graph's unitigs to the reference genome [155], which results in greater sensitivity to small variation but may also increase bias towards the reference genome. Aligning external sequence to an assembly graph is equivalent to extending the assembly to include the sequence and recording the path through the assembly graph that represents it.

The de Bruijn Graph Aligner (deBGA) inverts this approach by enabling the alignment of short reads to a reference DBG (RdBG) [170]. An RdBG is a compacted DBG in which one or more reference genomes have been embedded, and the authors of deBGA enable alignment against it through a k-mer index that is used to drive a kind of MEM-based seed and extend alignment. The alignment process itself attempts to link patterns of k-mer hits in unitigs in the graph into larger alignments, finally producing a local alignment by applying Smith-Waterman-Gotoh to the read and the particular reference genome region to which it aligns. While deBGA uses a DBG to structure alignment, it can output a BAM file against the linear reference genome for use in variant calling or other analyses. The authors claim that deBGA is fast and, thanks to its use of a DBG-based index, robust to alignment problems introduced by repeats. However, it is not clear how the method develops mapping qualities, and due to its inability to represent evolutionary homology or equivalence between regions of genomes, this would appear to be is a significant limitation of the reference data structure they have chosen.

deBGA is the first published method specifically designed to align short reads against arbitrarily-structured graph genomes. While deBGA was designed to work on collections of linear references, the authors note that it would also be possible to apply the indexing strategy to any sequence graph, as k-mer enumeration may be used to convert any

sequence graph into a DBG[20] The authors do not evaluate its operation on generic DBGs, and instead compare it to linear reference based aligners on the same collections of genomes.

1.4.3.4 Genotyping using a sequence DAG

Variant calls in the VCF format, when combined with the reference genome to which they refer, form a sequence DAG that encodes all the genomes from which the calls were derived as well as novel recombinations between them. As this feature of variant calls was appreciated, it led to the development of several methods which first map reads globally to a linear reference and then realign them locally to a variation-aware reference. This can be shown to reduce reference bias, but it can only do so in a localized sense as this form of graph resequencing cannot change the global placement of reads.

By the final phase of the 1000GP [45] it was apparent that indels and complex variation were more difficult to genotype than SNPs. Their significance was appreciated by 1000GP subprojects that examined the putative functional effects of such variants [36], which motivated efforts to develop a high-quality variant set including them for the final release. The best-performing methods for generating the SNP genotype likelihoods (GLs) were only able to model biallelic SNPs [283], and additional methods would be applied to derive GLs for non-SNP, non-biallelic variant types.

While participating in the project, Deniz Kural and I extended (as glia[21]) a POA algorithm he had developed to realign poorly-mapped reads (such as those with softclips, many mismatches, gaps, or unaligned fragments anchored by their pair mates) to a sequence DAG created from the reference and candidate alleles in the region. By projecting the alignment back into the reference space, we were able to generate a BAM output from glia. I implemented improvements to freebayes that allowed it to genotype alleles represented by observations in these alignments, as well as contamination estimates that were essential to high-quality GLs in low-coverage data. This pipeline outperformed alternatives, and was ultimately used to generate GLs for all non-SNP variation in the project.

The indels were integrated into the phased scaffolds provided by the SNPs through two cycles of genotyping. In the first, GLs generated by the method were given to MVNcall [184], a multalleleic, site-independent phasing algorithm that can phase genotypes at a given site onto a fixed background haplotype scaffold. The posterior genotype and

[20] Jouni Sirén and I had implemented an aligner based on a DBG transform of an arbitrary sequence graph at the time of deBGA's publication. The authors of deBGA were apparently as unaware of our work as we were of theirs.

[21] https://github.com/ekg/glia

phasing quality estimates produced by MVNcall, along with a number of other metrics related to indel sequencing error such as sequence entropy and homopolymer context, were then used to establish a support vector machine (SVM) classification model which was trained using data from high-quality genomes and applied to the full set of alleles to remove likely errors. The final set of alleles were fed back through the glia, freebayes, and MVNcall process to ultimately produce the set of indels in the 1000GP phase 3 release.

Hannes Eggertson's GraphTyper [71] implements a similar workflow to the genotype likelihood generation method applied to the 1000GP indels. In the GraphTyper pipeline, Illumina reads are aligned against the reference genome using a standard short read aligner. Those alignments with soft clips and apparent differences from the reference are matched to a sequence DAG built from the reference and VCF in a window around the read's candidate mapping location. GraphTyper matches short k-mers between the read and the sequence DAG using a k-mer index of the reference structure and 1bp-overlapping k-mers from the read. This does not produce an alignment *per se*, but rather a list of variant traversals supported by each read. This transformation provides sufficient information to genotype the variants in the graph. The pipeline may update its reference system so that it includes both known variation from other studies and new variation discovered during analysis. GraphTyper uses the graph reference system internally to improve algorithm performance, but results are projected into the linear reference and the graph has no other representation than VCF. This prevents the representation of "nested" variation or non-SNP or indel SVs. GraphTyper's efficient and accurate performance on exceptionally large resequencing problems supports the authors' design decisions and firmly asserts the utility of the graph realignment approach.

Sibbesen and colleagues generalize the genotyping problem to arbitrary variation graphs with BayesTyper [249]. They adopt a kind of pseudoalignment model in which exact k-mer matches between a read set and the reference are used to establish support for paths across bubbles in the variation graph. These can then be used to build probabilistic models of genotypes for any kind of variation represented in the graph, both small (SNPs and indels), including nested variation.

1.4.3.5 Population reference graphs

Pangenomic references need not be used as a coordinate system in the same manner as the linear reference genome. Instead, they can be used as a prior to infer most-likely underlying haplotypes of a given individual. Further refinement can be achieved by aligning the read set back to the inferred haplotypes. This approach makes sense if

alignment against the pangenome is extremely costly, but efficient haplotype inference patterns can be applied to the genome.

Population reference graphs (PRGs) [63] are POA-like graphs in which sequences aligned by MSA are collapsed in the case of identity above a given k-mer size, which embeds some information about local phasing into the graph. First assembled from long sequences, the PRG is then augmented with local variation information, with SNPs and indels added to all paths at appropriate positions. Dilthey and colleagues develop a genome inference model based on the comparison of the PRG to a DBG built from reads from a given individual. By comparison between these two structures, weighted by k-mer frequency in the genome and PRG, they provide sufficient annotations to the PRG to run an HMM on the graph to infer the most-likely underlying pair of haplotypes. Short read alignment can be used to map the full reads back to these haplotypes in an *ad hoc* manner in order to find new small variation against them and fully resolve their sequences. The authors demonstrate that this method can be applied to the human MHC, where high sequence diversity frustrates methods optimized for typical regions of the human genome.

1.4.3.6 Succinct pangenomic sequence indexes

Generalizations of the FM-index to support indexing of sequence DAGs, or equivalently regular languages, yielded a number of short read to sequence graph aligners. Such methods enable pattern matching against pangenome graphs in much the same way as done by FM-index based short read aligners.

The Generalized Compressed Suffix Array (GCSA) [257, 258] enables pattern matching against arbitrary finite regular languages. It indexes a reverse-deterministic automaton[22] that encodes the sequence DAG implied by a VCF and reference genome. To enable path queries, it builds the BWT based on the sorted prefixes of the language encoded in the automaton, adding support structures that allow pattern search as in an FM-index. Additional support bitvectors akin to those used to encode and index labeled trees [82] allow the traversal of the sequence DAG during query matching. Sirén constructs the index using the prefix-doubling method used to construct sorted suffixes, wherein prefixes and their starts and ends may be extended from length L to $2L$ through a sort and join operation. Because it indexes a memoryless automaton and the construction requires the enumeration of all the prefixes of the automaton, GCSA suffers from exponential costs in index generation in the order of the number of possible recombinations represented in the

[22]Reverse determinization ensures that each prefix of this automaton can only have a single starting position.

sequence DAG. Careful curation of the pangenome allows the construction of the GCSA for the entire human genome in the memory of an available computer (1TB), provided the construction is broken into chromosomes and local complexity in the input sequence DAG is reduced. Experiments with the GCSA used backtracking search as in BWA [158] to directly align reads with differences from the indexed graph, but the method was not developed into a full-featured short read alignment method.

Alternative schemes make no fundamental change to the CSA/FM-index model, but rather embed information about particular kinds of pangenome graphs into the sequence that is indexed. BWBBLE [113] encodes SNPs by extending the reference alphabet to include ambiguity codes which match more than one DNA base. To encode indels, it extends the reference by adding a sequence for each indel which includes the non-reference allele and the surrounding $2l$ bases of the reference. The resulting index can support queries of up to length l. A mapping between the positions in the extended reference and the original reference sequence space is used to project alignments from the extended reference to the base reference. This approach is simple and allows for a linear-time indexing of the pangenome, but the added complexity of the extended reference and larger alphabet required to represent SNPs yield a query time that is 100-fold slower than the backtracking BWA method [113].

In gramtools [172], a particular kind of sequence DAG is indexed in a large-alphabet CSA based on wavelet trees [104].[23] Maciuca and colleagues develop an encoding for a sequence DAG in which bubbles may not have any deeper internal structure, and any traversal across a bubble is required to contain sequence. Effectively, bubbles contain a number of alternate alleles that may be represented as linear sequences, and conveniently, this kind of graph is exactly that which is used in VCF. To build its vBWT index of the graph, gramtools linearizes the alt-bubble sequence DAG into an integer vector (the "linear PRG") in which DNA bases are represented normally while the structure of each bubble is encoded as a series of alternate alleles delimited by i, flanked by delimiters $i-1$. Each bubble gets a unique even integer i, requiring the use of alphabets with millions of symbols. A CSA-based FM-index is then built from the linear PRG. The use of a unique pair of integers for each bubble ensures that the alleles in the bubble will be sorted together in the CSA of the linear PRG. To support direct matching to the sequence DAG, the standard backwards search algorithm is augmented to consider the alternate alleles

[23] Wavelet trees reduce the representation of a large alphabet of size Σ into a tree of $O(\log \Sigma)$ bitvectors that recursively partition the sequence space into each character. These bitvectors may be compressed efficiently while maintaining accessibility, and by augmenting them with rank and select supports the entire structure can be used to determine the number of each class of character before a given position, or select the ith of a given character. This allows for the implementation of LF mapping on a CSA represented in a large alphabet.

when it encounters an odd integer greater than 4 (the alphabet space allocated to DNA sequence encoding). On encountering a bubble, the search algorithm adds new suffix array intervals for each alternate allele to a heap of intervals that are extended together at each step. As this scheme results in an exponential increase in the number of suffix array intervals as a query traverses multiple variants, the input set of alleles must be structured in a way to reduce the density of bubbles. In their experiments the authors set an allele frequency threshold and merge alleles within a certain window size into larger haplotype alleles[24]. It is perhaps due to this exponential factor that gramtools' query performance is many times slower than BWBBLE and around 1000 times slower than bwa mem on a linear version of the same reference [172]. In effect, the method trades exponentially expensive construction for exponentially expensive queries. Gramtools does not implement any local alignment model, and to demonstrate the method's conceptual utility, the authors apply it to infer the most-likely linear genome for a given query set, to which the read set is ultimately mapped using a standard aligner.

HISAT2 implements variation-aware alignment against human genome scale graphs, including alignment to RNA splicing junctions [137]. Its hierarchical GFM-index structure allows seeding alignments against sequence DAGs. To find seeds globally, it uses an implementation of the generalized compressed suffix array (GCSA) [257]. It builds the global index including common short SNPs and indels. Then a set of local GCSA indexes are used to allow fast search of the splice graph and determination of candidate splicing junctions even when supported by minimal evidence in the read. HISAT2 is an important point of reference as a scalable and mature implementation of sequence to graph alignment, and is presently the only widely-used aligner based on the GCSA index. Like other methods, HISAT2 writes BAM alignments resulting from the expression of alignments to the sequence graph against the linear reference genome. Unlike the other pangenomic aligners, HISAT2 can achieve very high throughput, and is competitive with standard short read aligners even when considering splicing and small variants.

Sirén's more recent work on this topic has produced GCSA2 [256]. This model removes the automaton reverse-determinization step, allowing the index to be built on top of any kind of graph. GCSA2 indexes a DBG generated from a bidirectional sequence graph in which nodes retain both k-mer identity and their starting and ending positional context in the graph. This positional information is used to drive the prefix doubling step required to build the BWT of the DBG. The space requirements of GCSA are avoided by terminating the prefix doubling at a length appropriate to accommodate queries

[24]The algorithm used to convert the input VCF into the haplotype bubble form appears to be similar to vcfgeno2haplo in vcflib: https://github.com/vcflib/vcflib/blob/master/src/vcfgeno2haplo.cpp.

of interest, which in practice is limited to 256bp. By computing the longest common prefix array (LCP) of its sorted suffixes, GCSA2 encodes the implied suffix tree, which supports MEM-based seed generation algorithms for efficient and sensitive alignment against variation graphs.

1.4.3.7 Mapping to *k*-mer based pangenome indexes

A k-mer index is trivial to build from a graph: one just needs to be able to enumerate or sample k-long walks through the graph and link the graph position to the k-mer in an efficient hash table or other kind of index. The size of the index can be reduced by sampling k-mers in some pattern. This indexing strategy was used in an early version of `vg`, which supported experiments by the GA4GH-DWG [203] oriented at understanding the utility of various graphs constructed for a set of loci where GRCh38 encoded alternative sequences. This same indexing technique is used by a proprietary method developed by members of the GA4GH-DWG from Seven Bridges Genomics Inc. (SBG) [223]. The SBG graph aligner's (SBGA) input is constructed from higher-frequency variants found in various public variation resources. It masks out regions of high allelic complexity and builds a reference-rooted edge-labeled sequence DAG to serve as an alignment target. To enable graph mapping it builds an index of spaced k-mers by walking short segments of the graph. The k-mers and their locations are used as seeds to establish mapping candidate loci. A local alignment is obtained for each candidate, and the output is transformed into the reference space as BAM.

Curiously, to align the reads to the graph locally SBG's aligner does not use a method like POA, and instead uses a kind of local backtracking exact matching against the tree of paths through the local graph which has exponential complexity with respect to the density of variation. If the variant density and read error rates are kept low, this scheme allows SBGA to achieve extremely high read throughput, equivalent to that of bwa mem without any apparent loss in accuracy on the linear reference. In this context the authors demonstrate their method can significantly (although slightly) improve sensitivity to known indels. However, the method is not adaptable to other graph types or sequencing contexts, with reports from users indicating high increases in runtime with increased read error and variant density. Due to the fact that SBGA is closed source and proprietary it is not possible to appreciate exactly why.

1.5 Overview and objectives

Resolving the genome of a sample *de novo* requires sequencing and assembly. Standard approaches to assembly are built on graphical models that allow for ambiguity, and out of this system they attempt to derive a set of contigs which represent the true haplotypes of the genome. When we have already assembled a genome related to the one we wish to infer, we can describe our new sample in terms of a reference genome, with ambiguity and variation represented only in the alignments of the new sequences where they can be mapped to the reference genome. This reduces the cost of sequencing, as we can infer much of the genome using lower sequencing coverage and shorter reads, but it also exposes resequencing based genome inference to reference bias, which is a distortion of inferred genomes towards the sequence of the reference genome.

I posit that by extending the reference genome to be a pangenome with a graphical representation, we can enable population aware resequencing that avoids reference bias to any particular linear reference. The model I develop, the variation graph, extends the bidirectional sequence graph models used in assembly to support the labeling of paths through the graph, and allowing the representation and relation of linear reference systems within itself.

Variation graphs are related to a wide array of graphical models used in bioinformatics. They have similarity with string graphs, multiple sequence alignments, and whole genome alignment graphs. This indicates that they could serve as a unifying basis for many domains of sequence analysis which have traditionally been separated by methodological differences, such as assembly and variant calling.

Recently, several methods have been published which provide variation-aware alignment or genotyping based on sequence DAGs built from population resequencing. These methods demonstrate the difficulty of the problem of generalizing resequencing to graphical reference systems at the multi-gigabase scale of vertebrate genomes. In virtually all cases, they use a graph reference model internally, but express their results in terms of a linear reference genome, producing BAM alignments or genotyping results in VCF format. None implement a generic strategy to work with a graphical model equivalent to the variation graph. The method I develop, `vg`, is not the first pangenomic aligner, but it is the first that models its results in terms of the pangenome itself, enabling full resequencing analyses to be completed within the graphical model.

In the following chapters I will precisely define the variation graph and associated data models that allow representing all kinds of resequencing data types in the context of a variation graph. Then I will describe the algorithms implemented in `vg` that enable the use of variation graphs as a basis for genome analysis, including their construction from

many data sources, serialization and visualization. I will present indexing methods that support the queries needed to enable resequencing, and I will provide algorithms that implement efficient read alignment to large scale graphs. Finally, I will cover applications of the method I developed. I will show that `vg` is applicable to a wide range of genomic inference problems, with a particular focus on the quality of alignment of both short and long sequence reads against variation graphs constructed from all kinds of sources.

Throughout this work I use the first person singular *"I"* to refer to work that I completed alone, while the plural *"we"* where presenting work completed in collaboration with others. Wherever possible, I clarify the nature of the collaboration and identify my collaborators.

Chapter 2

Variation graphs

Variation graphs (VGs)[1], previously introduced in section 1.3.2, combine a bidirectional sequence graphs with paths that model sequences as walks through the positional space of the graph. They link graphical models and linear sequence models. This allows them to be used to model the relationships between collection of sequences, including all variation contained therein. The encapsulation of these two divergent ways of modeling about bioinformatic data systems allows them to bridge traditionally isolated analysis modalities.

In this chapter, I will articulate the variation graph model and lay out the algorithms and data structures that enable its use as a reference system in pangenomic resequencing. First I will provide formulations for the graph, its paths, edits, alignments, and genotypes define within it. Then I will present algorithms that induce the variation graph from different data models introduced in the previous chapter. I describe the serialization techniques used to exchange variation data via computer files or network connections. I develop index structures to enable queries of the graph's topology, sequence, and path spaces, and algorithms to derive optimal alignments to the graph. Understanding variation graphs requires techniques to visualize them, and I will present various approaches, each with particular advantages and drawbacks. Working with variation graph references necessitates a number of graph-modifying operations, including augmentation, sorting, pruning, and bubble simplification. Finally, I will discuss how variation graphs can provide normalized basis spaces for the analysis of pangenomes, such as through various projections of alignment sets and the graph including coverage maps, ultrabubble decomposition, and haplotype matching.

[1] I will refer to variation graph as VG, and to the software implementation of the VG model `vg`

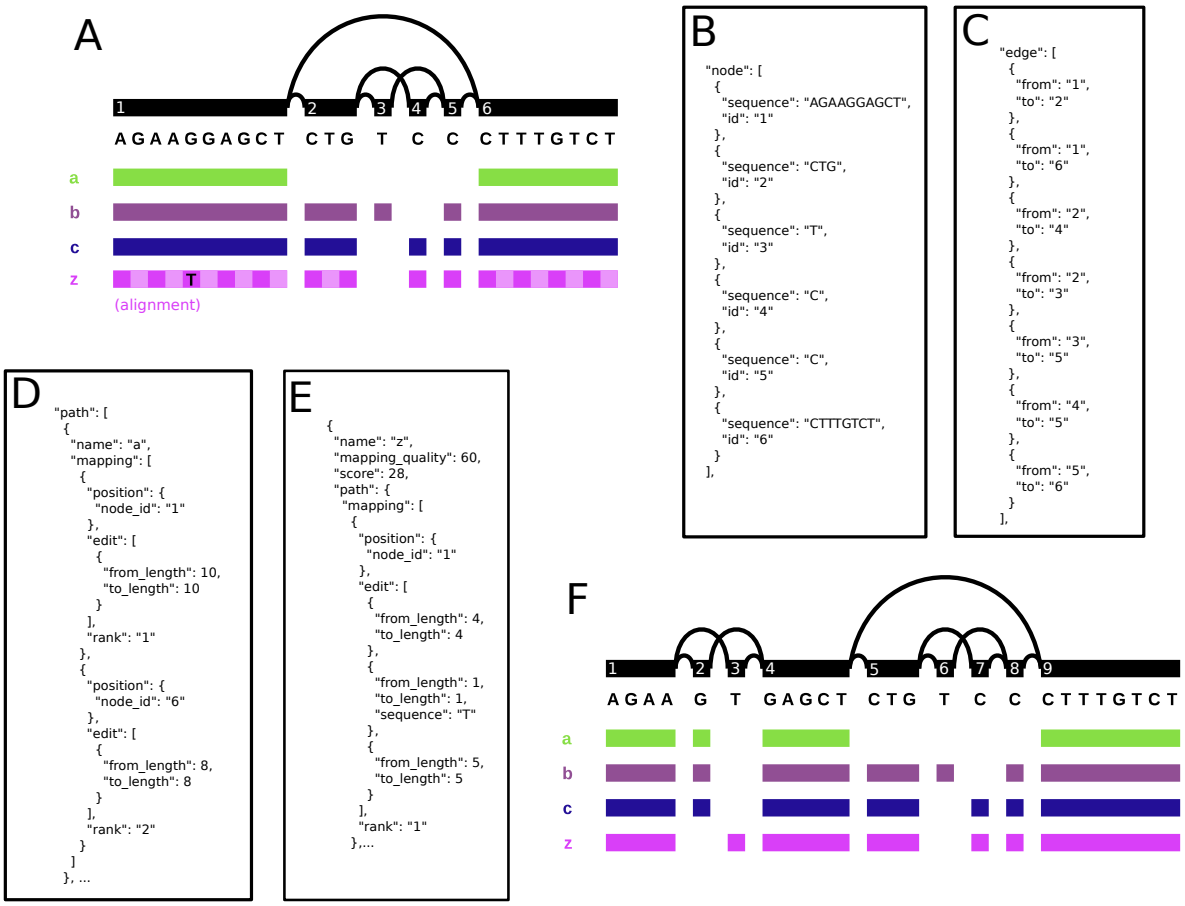


Fig. 2.1 A step in the progressive construction of a variation graph from homologous fragments of four haplotypes in the GRCh38 alternate allele set for the HLA gene DQB1-3119. (A) shows the variation graph constructed by progressive alignment of three sequences labeled *a*, *b*, and *c*. The graph topology is shown at the top of the panel, with nodes represented by black bars, labeled by their node IDs (white numbers), and with edge connections between nodes shown by arcs above. Paths are indicated in the colored bars below, with an identifying name to the left. This graph is partially ordered, without cycles, and so the sequences of paths may be enumerated by concatenating the node sequences above the filled portions of each of the path rows. A fourth sequence *z* has been aligned to the graph, and is shown in a checkerboard magenta pattern. It contains a single SNP relative to the graph, which is shown by the black `T` matching the fifth position of the first node. The JSON presented in (B) describes the full node set of the graph in (A), while (C) lists the edges. (D) shows the path *a* embedded in the graph in full. Other paths are not shown, but are recorded similarly. (E) describes the alignment of *z* against graph (A). Only the beginning of the path of *z* against (A) is given. The first mapping in the path describes a SNP between the matching subsequence in *z* and n_1. Note the second edit in the mapping, which specifically describes the SNP as a replacement of a subsequence in n_1 by `T`. Graph (F) is the result of the application of the *edit* operation to graph (A) and alignment *z*. The node IDs have been reassigned. The inclusion of the SNP increases the node count of the graph by 3. Path $[n_1]$ in graph (A) maps to $[n_1, n_2, n_4]$ in (F).

2.1 A generic graph embedding for genomics

We define a *variation graph* to be a graph with embedded paths $G = (N, E, P)$ comprising a set of *nodes* $N = n_1 \dots n_M$, a set of *edges* $E = e_1 \dots e_L$, and a set of *paths* $P = p_1 \dots p_Q$, each of which describes the embedding of a sequence into the graph. By generalizing these paths to support edits against the graph, we provide a mechanism to describe relations between the graph and other sequences. Augmenting the path with additional information important to sequence analysis allows us to construct an *alignment*. Collections of pairs of paths covering the space of two graphs describe a graph to graph alignment, or *translation* which can be generated when the graph is edited, to allow for the projection of coordinates and sequences in one graph into the space of the other. A limited form of this translation is a *genotype*, which maps the implied bubble formed across multiple copies of a homologous locus into the space of the graph. Collections of genotypes are the primary output of resequencing. Phasing algorithms extend genotypes into longer phased haplotypes, which we record as paths through the graph. These data models thus provide a sufficient informational basis for resequencing against variation graphs.

2.1.1 The bidirectional sequence graph

Each node n_i represents a sequence $seq(n_i)$ that is built from an alphabet $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}, \mathtt{N}\}$. Nodes may be traversed in either the forward or reverse direction, with the sequence being reverse-complemented in the reverse direction. We write $\overline{n_i}$ for the reverse-complement of node n_i, so that $seq(n_i) = revcomp(seq(\overline{n_i}))$. Note that $n_i = \overline{\overline{n}}$. For convenience, we refer to both n_i and $\overline{n_i}$ as "nodes".

Edges represent adjacencies between the sequences of the nodes they connect. Thus, the graph implicitly encodes longer sequences as the concatenation of node sequences along walks through the graph. Edges can be identified with the ordered pairs of oriented nodes that they link, so we can write $e_{ij} = (n_i, n_j)$. Edges also can be traversed in either the forward or the reverse direction, with the reverse traversal defined as $\overline{e_{ij}} = (\overline{n_j}, \overline{n_i})$. VGs can contain ordinary cycles (in which n_i is reachable from n_i), reversing cycles (in which n_i is reachable from $\overline{n_i}$), and non-cyclic instances of reversal (in which both n_i and $\overline{n_i}$ are reachable from n_j).

The bidirectional sequence graph underlying the variation graph model is illustrated in panels A, B, and C of figure 2.1.

2.1.2 Paths with edits

We implement paths as an edit string with respect to the concatenation of node subsequences along a directed walk through the graph. We do not require the alignment described by the edit string to start at the beginning of the sequence of the initial node, nor to terminate at the end of the sequence of the terminal node. To allow the path model to support differences from the graph, each path is composed of a series of node mappings $p_i = m_1 \dots m_{|p_i|}$ which are semantically identical to the alignment format used by standard aligners. Each mapping $m_i = (b_i, \Delta_i)$ has a starting position encoded as a node and offset in the graph $b_i = (n_j, o_i)$ and a series of edits $\Delta_i = \delta_1 \dots \delta_{|m_i|}$. Edits $\delta_i = (f_i, t_i, r_i)$ represent a length f_i in the graph node n_j (a "from length" in the reference), a length t_i in the sequence the path encodes (a "to length" in the query), and an additional sequence r_i that would replace the sequence at the given position in the reference in order to transform it into the query. In the case of exact matches, we allow the replacement sequence r_i to be empty.

Alignments are often described in terms of matches, mismatches, and indels. We encode matches when $f_i = t_i \wedge r_i = \emptyset$, single mismatches when $f_i = t_i = 1 \wedge r_i \neq \emptyset$, deletions when $f_i > 0 \wedge t_i = 0 \wedge r_i = \emptyset$, and insertions when $f_i = 0 \wedge t_i > 0 \wedge r_i \neq \emptyset$. As paths are described by a series of mappings with independent positions, they can represent all kinds of structural variation relative to the graph. When mapping positions are always at the start of a node, the edit set for the path contains only matches, and the edges traversed by the path are all present in the graph[2], we say that the path is *embedded*. The paths from which we construct the variation graph are fully embedded, and in practice paths that contain differences occur only in the alignment of new sequences into the graph.

The example variation graph in figure 2.1 panel A contains several paths, while panel D explicitly describes a portion of one of the paths using an equivalent model to the one described here.

2.1.3 Alignments

Auxiliary read information is important when analyzing collections of DNA sequencing data sets. Each read has a name, and an identity related to a particular sequencing experiment. It may be related to a particular genomic sample or individual. DNA sequence reads themselves result from a previous set of analyses run on raw observations

[2]Note that paths may contain disjoint mappings that are not connected by edges in the graph, which allows them to represent structural variations.

derived from DNA, perhaps fluorescence or current traces or images. The process of collapsing this raw information into the sequence read yields a confidence in addition to a base call. These are recorded in a quality string in FASTQ. The need to collect this information has resulted in the development of the SAM/BAM sequence alignment format, which provides a standard for linking the called bases (sequence), quality information, read name, features of the alignment against a reference genome and additional optional typed annotations.

I follow this same model in developing an alignment format for read alignments to the graph. An aligned set of sequences Q, $A = a_1 \dots a_{|Q|}$, represents a sequencing experiment. Each aligned read connects a sequence, an (optional) quality string, a path through the graph including possible edits, and an optional set of D_i annotations: $a_i = (s_i, q_i, p_i, k_1 \dots k_{D_i})$. In principle the read sequence can be reconstructed from the path, but retaining the sequence information makes the alignment object lossless with respect to the input FASTQ and provides redundancy which can help in data processing.

Panels A and E of figure 2.1 demonstrate how an alignment to a variation graph can encode putative variation, in this particular case encoding a SNP.

2.1.4 Translations

A generalization of the alignment is the translation set $\Phi = \phi_1 \dots \phi_{|\Phi|}$, which relates paths in different graphs to describe the mapping between them. A translation $\phi = (p_f, p_t)$ defines the projection between two paths which may arise in the context of two graphs G_f and G_t. In this use each p_f corresponds to a path relative to G_f (conventionally the base or reference graph), and each p_t to some path in G_t.

If each node and edge and path in both graphs is represented in some graph translation in Φ then it provides an isomorphic relationship between the graphs. Provided each Φ encodes an isomorphism, then we can layer a series of Φ_i together to provide a coherent coordinate space across any number of updates to a given graph. Consider a function pattern *translate*, which allows the projection of paths relative to G_f through translations Φ to yield paths in G_t: $translate(p_i, \Phi) \to p_j \in G_t$, and similarly allows the transformation of a base graph into a target graph: $translate(G_f, \Phi) \to G_t$. If we have a series of $(G_i, \Phi_1) \dots (G_\rho, \Phi_\rho)$, where $translate(G_i, \Phi_i) \to G_{i+1}$ and thus each Φ_i describes an isomorphism between G_i and G_{i+1}, then we can generate a graph translation Φ_Δ providing $translate(G_1, \Phi_\Delta) \to G_\rho$. We build this graph translation with the function $layer(\Phi_\alpha, \Phi_\beta) \to \Phi_{\alpha \circ \beta}$ by rewriting each path translation $\phi_i \in \Phi_\alpha$ so that its p_t refers to G_β. We do so by projecting the p_t through Φ_β, and finally adding any $\phi_j \in \Phi_\beta$ for which $p_f = \emptyset$, as these represent insertions of new sequence in G_β relative to G_α.

2.1.5 Genotypes

As path to path relationships can provide descriptions of allelic diversity, they form the basis for a graph-relative genotype encoding. To represent the exact genotype of a particular sample with ploidy ν at a given locus ι we can simply collect the multiset of alleles $\pi_\iota = (p_1 \dots p_\nu)$. We could alternatively build a probabilistic model ϖ of an unphased genotype by using a set of μ alleles $\{p_1 \dots p_\mu\}$. To do so, we associate likelihoods γ_ξ for each possible genotype π_{ι_ξ} that could be sampled from the alleles such that $\varpi = \gamma_1 \dots \gamma_{\frac{\mu!}{\nu!(\mu-\nu)!}}$. In practice, we develop our γ_ξ out of quality information from the reads and a sampling model related to ν [91, 151]. Existing genotyping models can be applied to drive genotyping using read sets aligned to the graph, and the output of the genotyper is defined fully in the space of the graph.

2.1.6 Extending the graph

Given an alignment a_i, we can edit the graph G so that it includes the alignment and the sequence it represents as an embedded path, $augment(G, a_i) \rightarrow (G', \Phi)$, such that $translate(p_i, \Phi) \in G'$. To update the path space of the graph we project all paths, including that of a_i, through the translation implied by the augmentation of the graph with p_i. Any other alignment a_j whose path p_j overlaps p_i would no longer be valid, although it could be projected through the graph translation Φ as well to express it in the space of the new graph G'. Updating the graph one alignment at a time is inefficient as we need to build and layer a new translation for each alignment. It is simpler to edit the graph in a single step, taking a collection of alignments and including them in the graph, $edit(G, A) \rightarrow (G', \Phi)$.

One way to accomplish this is to first take the set of unique mappings represented in the paths of A, $\Omega = \{m_1 \dots m_{|\Omega|}\}$, and for each m_i cut n_i at the breakpoints where any new variation would need to be added in, adding new nodes to represent the cut portions. Then, walking through each alignment we add in unique novel sequences and their linkages to the preexisting nodes or new breakpoints to the graph. This process will disrupt the identifier space of the nodes and edges of the graph, but it naturally yields a translation that can be used as described in section 2.1.4. Both alignments and genotypes are based on paths, so this mechanism can be used to extend the graph based on any sequence level differences observed through alignment or variant calling.

Figure 2.1 provides an example to provide a concrete intuition about the *edit* function. Editing the graph in panel A with the alignment of sequence z yields the graph in panel F. The corresponding translation is described in the figure legend.

2.2 Variation graph construction

We will use variation graphs as the core model for a number of essential processes in genome inference. This model can represent many graphical sequence models used in genomics. Each one necessitates conversion into the variation graph model. Here I describe the transformation of a number of graphical models into variation graphs, including MSAs, assembly graphs, and alignment graphs induced from pairwise alignments. In some cases the conversion is direct, but in others it requires the addition of new labels to our model. Variation graphs may also be built from first principles, provided a function that aligns a sequence into the graph and the editing operations described in 2.1.6.

2.2.1 Progressive alignment

If we have a series of k queries $q_1 \dots q_k$, then we can build a progressive alignment by a series of edit and alignment operations applied to the variation graph. First, take the empty graph $G_\emptyset$, to which any alignment will yield a path p_1 that has no mappings and which encodes the query sequence q_1 as a replacement sequence in the path. We then edit the graph to add the sequence using $edit(G_\emptyset, p_1) \to G_1$. For each subsequent q_j we obtain the next graph by finding the alignment $align(q_j, G_j) \to p_j$ and editing the graph with it to yield the next graph $edit(G_j, p_j) \to G_{j+1}$ until $j = k$ and we obtain our final graph. This simple approach is attractive as it allows the variation graphs to be built from whole sequences using only techniques that are native to the variation graph model itself. However, it is obviously order dependent, with potentially different results if the set of input sequences are presented in different orders. I later presents results based on this multiple sequence to graph alignment process, `vg msga`.

2.2.2 Using variants in VCF format

As discussed in section 1.4.3.4, the VCF format that is popular in resequencing implies a sequence DAG. We can consider how to build a trivial variation graph using the core operation *edit*. First, we build a variation graph from the reference genome $Q_{\mathbf{ref}}$: $G_{\mathbf{ref}}$. This graph contains one path $p_{\mathbf{ref}}$: $seq(p_{\mathbf{ref}}) = Q_{\mathbf{ref}}$. As described in section 2.1.5 each locus reported in VCF can be encoded as a set of paths $P_{\mathbf{vcf}} = p_1 \dots p_V$, each representing a different allele. We now edit the graph to embed these allele paths, $edit(G_{\mathbf{ref}}, P_{\mathbf{vcf}}) \to G_{\mathbf{vcf}}$. It is possible to regenerate the VCF file input by walking the positions of $p_{\mathbf{ref}}$ and enumerating the overlapping paths as alleles in VCF format.

For efficiency, we have not implemented VCF to variation graph conversion with specifically this algorithm, but instead build up $G_{\mathbf{vcf}}$ by walking along the reference genome $Q_{\mathbf{ref}}$ and processing each locus sequentially. This exploits the partially ordered property of the VCF to limit memory requirements. For regions before, after, and between variant records at reference offsets i and j we add a node $n_{\mathbf{curr}} : seq(n_{\mathbf{curr}}) = substr(Q_{\mathbf{ref}}, i, j)$, linking these by edges to those nodes ending at position i of the reference and adding corresponding mappings for the reference path to $p_{\mathbf{ref}}$. At simple variant sites we add each of the alleles as a new node $n_{\mathbf{var}_i}$, including an edge for each $e_{\mathbf{curr} \prec \mathbf{var}_i}$. Here we also handle the reference allele differently in that we append a mapping $m_{\mathbf{ref}} = ((n_{\mathbf{ref}}, 0), \emptyset)$ to $p_{\mathbf{ref}}$.

As long as the VCF records are ordered, this process allows for streaming conversion of the VCF format into a variation graph. However, VCFs used to represent structural variation often do not describe a fully-ordered series of loci. For instance, a large deletion may be described in one record, and followed by a number of records describing variation on the reference within the scope of the deletion. In the graph, this results in the nesting of bubbles, and requires a deviation from a simple streaming algorithm in order to be handled. Deletions must be recorded and linked into the downstream portion of the graph as it is later generated.

VCFs may also encode phased haplotypes, which, like the reference, have a natural representation in the graph as paths. Parsing these may require multiple passes over the VCF due to the memory requirements for storing large numbers of haplotypes uncompressed and cross-indexed to allow traversal in RAM. To prevent $O(H|G_{\mathbf{vcf}}|)$ growth of the required memory to store these, we implement compression strategies on the haplotype set that exploit their repetitiveness. The VCF format does not impose a semantic requirement that the encoded haplotypes are valid, which introduces some complexity in the implementation of this method. We must break haplotypes where they are found to be invalid in order to record them in VG format. For instance, a phased VCF may report more than ν (expected ploidy) alleles for a given individual, such as when deletion and SNP variants overlap. We expect these haplotype paths to be embedded in the graph. Although haplotype sets are equivalent to large collections of paths, we term their components *threads* to indicate that they have a simpler representation than arbitrary paths.

2.2.3 From gene models

A reference-based RNA splicing graph is usually expressed as a set of named intervals in BED or the General Feature Format (GFF). As with the generic VCF generation

algorithm, we can convert the transcripts to alignments A relative to the graph. Then we can then embed the transcript paths in the graph $edit(G_{\mathbf{ref}}, A) \rightarrow G_{\mathbf{splice}}$. Any transcript in our set is thus encoded by the graph, and can be matched to it directly with alignment. The resulting structure will also support novel isoforms built with splices from the known set.

2.2.4 From multiple sequence alignments

A multiple sequence alignment in matrix form has a simple translation into a sequence DAG and thus a VG [147]. Given a set of sequences $Q = q_1 \dots q_\kappa$ their υ-long mutual alignment may be described in a $\kappa\upsilon$ matrix X designed to maximize $\sum_{i=1}^{\upsilon}\sum_{j=1}^{\kappa}\sum_{k=1}^{\kappa}\delta_{X_{ij}X_{ik}}$, where δ is the Kronecker delta, or some generalization of this. The alphabet used to encode the matrix is the same as the input sequences with the addition of a special gap character $\square$ which does not match itself, and gaps thus do not contribute positively to the matrix score. To build a variation graph $G_{\mathbf{msa}}$ from the MSA we proceed from $i = 1 \rightarrow \upsilon$ through X. For each unique character $\mathcal{B}$ in the query alphabet $\Sigma \setminus \square$ found in each row i, we create a node $n_{\mathcal{B}}$ in $G_{\mathbf{msa}}$ and append a mapping to each path p_i for which $n_{\mathcal{B}} \in q_i$. We construct the edges of $G_{\mathbf{msa}}$ by taking the distinct pairs of consecutive node traversals found in the path set $P_{\mathbf{msa}}$ produced after the generation of the nodes in MSA traversal. Adding add an edge e_{ij} for each pair of nodes (n_i, n_j) consecutively traversed in $P_{\mathbf{msa}}$ ensures that our sequences are present as walks through the graph. We can optionally compact series of nodes (which represent single characters) where no furcations occur to obtain a simpler graph.

Instead of a matrix, we can formulate the multiple alignment as an alignment graph (described in section 1.4.2.5). By making this graph bidirectional, and thus equivalent in information content with the Enredo graph, it becomes equivalent to a variation graph. Aligners that produce data formats of this type, such as Cactus [212], can thus be used to produce VGs, so long as the relationship between the input sequences and the graph is recorded and can be converted into a path description. In section 3.1.2 I discuss the use of this method to build a pangenomic reference system for a diverse set of yeast strains.

2.2.5 From overlap assembly and de Bruijn graphs

Overlap-based sequence graphs used in assembly, described in sections 1.4.2.1, 1.4.2.2, and 1.4.2.3, are nearly identical to variation graphs. The critical difference between these models and the variation graph is that they attach a label to each edge (or node) describing the alignment between the pair of nodes (or edges) which they connect. Variation graphs

do not support such a feature in their basic definition, as it is unimportant for any use besides temporarily representing overlap graphs. We call the process of transferring sequence information from the edges to the nodes *bluntification*.

We start by assigning the sequence of each node to be the sequence of the corresponding read. If we shorten the sequence of a node to reduce an overlap between a pair of nodes, it will render other overlaps on the same nodes incorrect. Thus, it is essential that the bluntification algorithm work by the reduction of sets of overlaps on edges which are transitively closed by connection to the same ends of each node, $net(e_{ij}) \rightarrow \forall e_{i*} \in G \cup \forall e_{*j} \in G$, considering both strands of the graph when doing so. For each net we apply a function $pinch(net(e_{ij})) \rightarrow G_{\mathbf{pinch}_{ij}}$, which reduces the overlaps between the nodes in the net into a blunt-edged variation graph. We then link $G_{\mathbf{pinch}_{ij}}$ back into the rest of the graph by connecting with the inbound links to each node involved in the net.

In a de Bruijn graph or string graph as generated by SGA or Fermi2, overlaps are exact matches and so are defined given only a length. This simplifies the implementation of *pinch*, as no further computation is required to correctly determine the mutual alignment of overlapping sequences. In contrast, overlaps in a generic string or overlap graph are correctly defined as alignments. Resolving a single pairwise alignment into structures in the graph is trivial, but it becomes considerably more complex when many sequences map into a transitively closed set of overlaps. These nets can then be resolved into an alignment graph by an algorithm similar to that given in section 2.2.6, but in practice, `vg` implements bluntification using a pinch graph library developed for whole genome alignment [212].

2.2.6 From pairwise alignments

A set of pairwise alignments imply a variation graph, however I know of no contained method that will generate the variation graph or lossless string graph from these alignments. To explore this, I developed an algorithm to do so that operates in external memory, which I here present in detail. It operates by conversion of the alignment set into an alignment graph and the subsequent use of this graph in the elaboration of the variation graph including paths representing the input sequences. The resulting graph is a lossless representation of the input and the alignments between them. To distinguish the approach from string graphs, which imply error correction, I call this variation graph induction model the *squish graph*.

`seqwish`[3] implements a lossless conversion from pairwise alignments between sequences to a variation graph encoding the sequences and their alignments. As input, we typically take all-versus-all alignments, but the exact structure of the alignment set may be defined in an application specific way. `seqwish` uses a series of disk-backed sorts and passes over the alignment and sequence inputs to allow the graph to be constructed in low memory relative to the size of the input sequence set. Memory usage during construction and traversal is limited by the use of sorted disk-backed arrays and succinct rank/select dictionaries to record a queryable version of the graph.

As input, we have Q, which is a concatenation of the sequences from which we will build the graph. We build a compressed suffix array (CSA) mapping sequence names to offsets in Q, and also the inverse using a rank/select dictionary on a bitvector marking the starts of sequences in Q. This allows us to map between positions in the sequences of Q, which is the format in which alignment algorithms typically express alignments, and positions in Q itself, which is the coordinate space we will use as a basis for the generation of our graph. We encode the set of input pairwise alignments between sequences in Q as object A. Although these alignments tend to be represented using oriented interval pairs in Q, for simplicity and robustness to graph complexity, we describe A as a set of pairs of bidirectional positions (sequence offsets and strands) $[1 \dots |Q_1 \dots Q_{|Q|}|]$, such that $A = \{(b_q, b_r), \dots\}$. We sort A by the first member (b_q) of each pair, ensuring that the entries in A are ordered according to their order in Q.

To query the induced graph we build a rank/select dictionary allowing efficient traversal of A, based on a bit vector A_{bv} of the same length as A such that we record a 1 at those positions which correspond to the first instance of a given b_q and record a 0 in A_{bv} otherwise. We record which b_q we have processed in the bitvector Q_{seen} which is of the same length as Q. This allows us to avoid a quadratic penalty in the order of the size of the transitive closures in Q generated by pairs in A.

Now we inductively derive the graph implied by the alignments. For each base b_q in Q not already marked in Q_{seen}, we find its transitive closure $c_q := \{b_q, b_{r_1}, \dots\}$ by traversing aligned base pairs recorded in A. We write the character of the base b_q to an entry s_i in a vector S, then for each b_c in c_q we record a pair (s_i, b_c) into N and its reverse, (b_c, s_i) into P. We mark Q_{seen} for each base in each emitted cluster, so that we will not consider these bases in subsequent transitive closures. By sorting N and P by their first entries, we can build rank/select dictionaries on them akin to that we built on A that allow random access by graph base (as given in S) or input base (as given in Q).

[3] https://github.com/ekg/seqwish

To fully induce the variation graph we need to establish the links between bases in S that would be required for us to find any sequence in the input as a walk through the graph. We do so by rewriting Q (in both the forward and reverse orientation) in terms of pairs of bases in S, then sorting the resulting pairs by their first element, which yields $L = [(b_a, b_b), \ldots]$. These pairs record the links and their frequencies, which we can emit or filter (such as by frequency) as needed in particular applications. In typical use we take the graph to be given by the unique elements of L.

Our data model encodes the graph using single-base nodes, but often downstream use requires identifying nodes and thus we benefit from compressing the unitigs of the graph into single nodes, which reduces memory used by identifiers in analysis. We can compress the node space of the graph by traversing S, and for each base querying the inbound links. Maintaining a bitvector S_{id} of length equal to S we mark each base at which we see any link other than one from or to the previous base on the forward or reverse strand, or at bases where we have no incoming links. By building a rank/select dictionary on S_{id} we can assign a smaller set of node ids to the sequence space of the graph.

Given the id space encoded by S_{id} we can materialize the graph in a variety of interchange formats, or provide id-based interfaces to the indexed squish graph. To generate graphs in `vg` or GFA format, we want to decompose the graph into its nodes (S), edges (L) and paths (P). The nodes are given by S and S_{id}, and similarly we project L and P through S_{id} to obtain a compressed variation graph.

2.3 Data interchange

In `vg`, a schema language, Google Protocol Buffers (Protobuf), is used to define a compact description of data structures sufficient for the representation of all the required components. One cause of this pattern was my involvement in the GA4GH-DWG at the beginning of my thesis, which was then seeking a coherent way of describing graph genomes[4] to support pangenomic resequencing and related information exchange across the internet. I implemented the schema for `vg` in the popular Protobuf schema language. This provided a core API on which to build `vg`. It also implied a set of streaming data formats, which I implemented as a template library capable of serializing any stream of Protobuf objects. Due to the reliance on Protobuf, the only code needed to implement reading and writing of these formats is `vg` schema and the stream library[5]. This greatly

[4] https://github.com/ga4gh/ga4gh-schemas

[5] At the time of writing the schema, https://github.com/vgteam/vg/blob/master/src/vg.proto and stream parsing library https://github.com/vgteam/vg/blob/master/src/stream.hpp total around 1000 lines of code, and are sufficient to link any C++ program into the `vg` ecosystem.

simplified the process of developing libraries for working with the variation graph data models. Although in practice the Protobuf data structures are slower to parse than handmade C-struct serializations like BAM, the amount of effort required to begin writing efficient and structured binary data formats was considerably less with the schema based approach. Most importantly, the schema based definition of the core data types in `vg` helped new developers and researchers using the system quickly appreciate the basic concepts.

Several data formats are important to `vg`. In `.vg` format, the graph itself is serialized in non-overlapping chunks, where each edge e_{ij} is stored once in the chunk $G_{\mathbf{chunk}} : n_i \in G_{\mathbf{chunk}}$. Path mappings must have a rank that identifies their position in the path in order to be subdivided in this way. This allows them to be read in and rebuilt even if they have been serialized out of order. A series of alignment objects is a sensible output of the mapping algorithm `vg map`. The file format produced by writing out a series of Protobuf alignment object serializations using the `stream.hpp` library is called GAM, for Graphical Alignment/Map, in analogy to SAM (Sequence Alignment/Map format).

These data models have various other equivalent serializations. The GFA format can be used to directly encode VGs. However, GFA lacks a representation of an alignment with the semantics required by `vg`. VGs that are partially ordered can be deconstructed into VCF files. As paths in variation graphs can be used to represent any kind of existing annotations, data providers who represent annotations across many genomes (such as ENSEMBL Genomes) can build their annotation sets which were previously spread across many genomes into a single one embedded in a VG. To enable this several collaborators[6] have developed a Resource Description Framework (RDF) compatible version of the core VG model.

2.4 Index structures

As described in section 1.2.2, the large collections of read data produced by current sequencing methods require efficient read alignment to support downstream analysis. Typically, these methods develop indexes of their reference genome, using k-mer hash tables or FM-index/CSA based data structures that support efficient arbitrary-length exact matching. These indexes remain static during the resequencing analysis, and can thus be designed to be very compact and to support efficient queries.

When the genome is just a linear string, distances between locations may be computed trivially, and subsets of the sequence are simply substring operations on the vector

[6] Jerven Bolleman and Toshiaki Katayama among others.

representing the genome, so no additional structure beyond a full text index is required to seed the alignment of reads to the genome. However, this situation changes in graphs, where the computation of distances is more complex and particular topologies of the graph must be recorded and reproduced. Graph distances may be estimated using an approximate sort and the paths embedded in the graph. However, to do so requires efficient indexes of the path structure in the graph. Furthermore, loading the entire graph into memory in a naïve manner can be very expensive, and effort is required to minimize the runtime costs to enable resequencing even on lower-memory commodity compute servers.

2.4.1 Dynamic in-memory graph model

Serialized in `.vg` or compressed GFA format, the graph of the 1000GP is not much larger than the uncompressed human reference genome. However, the performance-oriented implementation of the dynamic variation graph which I developed at the beginning of my studies can use a hundred times this much memory when the entire graph is loaded into RAM. In this scheme implemented in `vg`, indexes on the node identifier space of the graph allow for fast traversal and query of nodes by identity and neighborhood, as well as insertion or deletion of nodes and edges and associated editing of paths. Various inefficiencies are accepted, such as on the hash table occupancies used to build these indexes, in the pursuit of higher performance during dynamic modification of the graph. I now believe that it should be easy to provide a dynamic VG in low memory by using a succinct encoding, but I have not yet completed any work on this issue. Operating on graphs of hundreds of millions of nodes with annotations like paths remains a difficult problem. In most cases the graph can be subdivided (as with map/reduce processing patterns [59] which underpin most industrial operation on large graphs [44]). This is particularly easy to implement for sequence DAG VGs, allowing for us to work on graphs of arbitrary sizes if they are approximately linear.

2.4.2 Graph topology index

The graph is unlikely to be changed during many kinds of analysis, and so we have the opportunity to compress it into static data structures that provide efficient access to important aspects of the graph with low memory overhead. Specifically, we care about the node and edge structure of the graph and queries that allow us to extract and seek to positions in embedded paths. We would like to be able to query a part of the graph corresponding to a particular region of a chromosome in a reference path embedded in

the graph. Similarly, if we find an exact match on the graph using GCSA2, we would like to load that region of the graph into memory for efficient local alignment.

We implement a succinct representation of variation graphs in the XG[7] library, using data structures from the C++ toolkit SDSL-lite [95]. Node labels and node ids are stored in a collection of succinct vectors, augmented by rank/select dictionaries that allow the lookup of node sequences and node ids. An internal node rank is given for each node, and we map from and to this internal coordinate system using a compressed integer vector of the same order as the node id range of the graph we have indexed. To allow efficient exploration of the graph, we store each node's edge context in a structured manner in an integer vector, into which we can jump via a rank/select dictionary keyed by node rank in the graph. Efficient traversal of the graph's topology via this structure is enabled by storing edges as relative offsets to the nodes to which they connect, which obviates the need for secondary lookups and reduces the cost of traversal. Paths provided to XG are used to induce alternative coordinate systems over the graph. We store them using a collection of integer vectors and rank/select dictionaries that allow for efficient queries of the paths at or near a given graph position, as well as queries that give us the graph context near a given path position.

An XG index of $G = (N, E, P)$ is composed primarily of the backing graph vector $G_{\mathbf{iv}} = g_1 \dots g_{|N|}$, with each g_i recording the edge context for node n_i in the graph: $g_i = (\eta_i, \Xi_i)$, a sequence vector $S_{\mathbf{iv}}$ recording the sequences labels of the nodes in a bitcompressed form, and a path membership mapping $N_{\mathbf{path}}$. Each $p_i \in P$ is encoded with a set of structures that allow random access to the graph by path position, which is important for the use of paths as reference coordinate systems in the graph. A visual sketch of this model is provided in figure 2.2.

To enable better compression, the node sequence space is recorded as a concatenation of node labels $S_{\mathbf{iv}} = seq(n_i) \dots seq(n_{|N|})$, in which each node has an offset in this sequence space defined by $seq_{\mathbf{offset}}(n_i)$. In bitvector $S_{\mathbf{bv}} : |S_{\mathbf{bv}}| = |S_{\mathbf{iv}}|$ we set 1 at each first character in a node label, and 0 otherwise: $S_{\mathbf{bv}}[i] = 1 \iff \exists j : seq_{\mathbf{offset}}(n_j) = i \vee 0$. Random access by node rank i is provided by function $S_{\mathbf{bv}}^{select_1}$, allowing us to find the sequence given a node rank in $G_{\mathbf{iv}}$.

Each η_i records node contextual information, including an external id, its offset in the sequence vector, and the degree of n_i in terms of inbound and outbound nodes: $\eta_i = [id(i), seq_{\mathbf{offset}}(n_i), |seq(n_i)|, in(n_i), out(n_i)]$. The use of an external identifier allows the index to work on subsets of larger graphs, and the information about node degree allows us to parse the edge records and efficiently traverse $G_{\mathbf{iv}}$.

[7] "X" implies compression and "G" refers to the graph that is compressed.

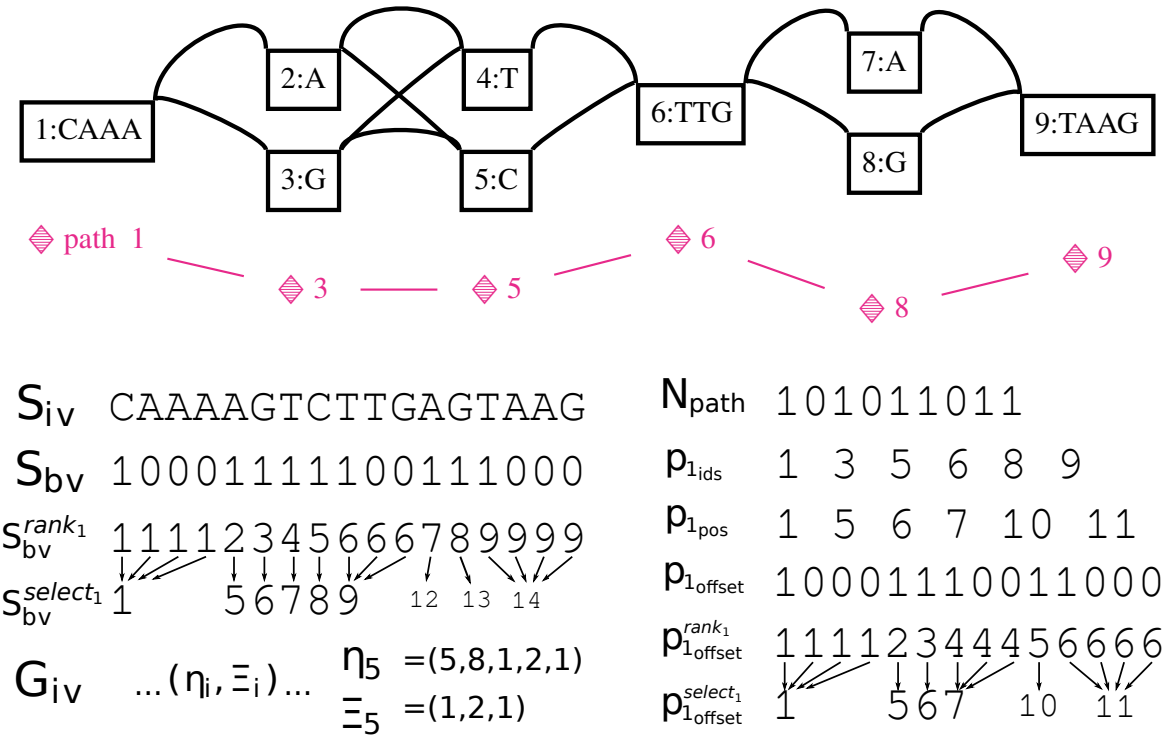


Fig. 2.2 A visual presentation of the key elements of the XG index for the given graph.

The edge context Ξ_i enumerates the set of edges that connect to this node in a structured way that allows for oriented traversals across the two strands of the graph. To enable fast traversal we rewrite the edges in terms of relative positions in the encoded $G_{\mathbf{iv}}$ vector, which is stored as a bitcompressed integer vector using SDSL-lite's template primitives. Each node record is stored contiguously. We delimit the records by a secondary bitvector $G_{\mathbf{bv}}$, for which we build supports for functions $G_{\mathbf{bv}}^{rank_1}$ and $G_{\mathbf{bv}}^{select_1}$, which allow random access of $G_{\mathbf{iv}}$ by node *id*. Previous designs decomposed the graph structure into a set of parallel vectors, but this required multiple select queries during traversal and provided poor cache locality and performance.

Node to path membership is recorded in an integer vector $N_{\mathbf{path}}$, which contains a contiguous record of path ids that cross each node. Random access to $N_{\mathbf{path}}$ is provided by a rank/select dictionary built on a bitvector delimiting the various node path membership lists. Nodes with no path membership are marked in $N_{\mathbf{path}}$ with a 0.

Each path is represented in a set of succinct data structures that let us walk the path by starting at a particular node, query the path position of a given node, or find the node at a particular path position. We store the path $p_i = m_1 \dots m_{|p_i|}$ by decomposing its mappings into the nodes (id and orientation) it traverses $p_{i_{\mathbf{ids}}} = id(n_j) \forall n_j \in p_i$ and

$p_{i_{\mathbf{dir}}}$ such that $p_{i_{\mathbf{dir}}}[j] = 0 \iff m_j = (n_j, \ldots) \vee 1 \iff m_j = (\overline{n_j}, \ldots)$. To allow rank and select queries on the node ids, we can store $p_{i_{\mathbf{ids}}}$ in a wavelet tree, although in practice performance is greatly improved by also recording a minimum node id, which decreases the alphabet size and thus memory and runtime costs of the wavelet tree. So that we may transform nodes to path positions, we use an integer vector to store a path position for each node traversal in the path, $p_{i_{\mathbf{pos}}}$. To go from path offset to node, we build a bitvector that marks the beginning of each node traversal in the path in a manner similar to that used to mark the sequence beginning of each node in $S_{\mathbf{iv}}$, such that $p_{i_{\mathbf{offset}}}$ is a bitvector of length $\sum_{j=1}^{|p_i|} |seq(p_i[j])|$ where we have marked 1 for each node start in the path, $p_{i_{\mathbf{offset}}}[j] = \left(1 \iff \exists n_k \in p : j = \sum_{m=1}^{k} |seq(p_i[m])|\right) \vee 0$. By implementing $p_{i_{\mathbf{offset}}}^{rank_1}$ we can find the node at a given path position $\mathcal{Q}$ by $p_{i_{\mathbf{offset}}}^{rank_1}(\mathcal{Q}) \rightarrow j : \left(\sum_{k=1}^{j} |seq(p_i[k])| \leq \mathcal{Q} \wedge \sum_{k=1}^{j+1} |seq(p_i[k])| > \mathcal{Q}\right)$. We can also find the position of the jth node in a path as $p_{i_{\mathbf{offset}}}^{select_1}(j)$. A compressed suffix array (CSA) and rank dictionary $P_{\mathbf{csa}}$ and $P_{\mathbf{name}}^{rank_1}$ map from path name to internal rank of the path in P.

A number of compression techniques can be applied to the data models in XG to reduce the size of the overall index without any loss in functionality. However, many of these compression methods producing compressed bitvectors, integer vectors, and wavelet trees, will result in slower access. In the context of read alignment, such losses may be undesirable as long as there is sufficient memory to load the entire index into system memory, and so I have tuned the index by choosing compression strategies appropriate for its use on current datasets.

2.4.3 Graph sequence indexes

As I presented in section 1.2.2, sequence alignment is driven by sequence indexes that allow efficient queries of subsequences in a given corpus (for instance a reference genome, or a set of reads). Two main varieties have proved useful. In k-mer indexes, short subsequences of length k are recorded in a hash table or equivalent data structure that allows efficient lookup. The keys in this table are the k-mers, while the values are typically a list of strand-oriented positions in the target set of sequences. BWT-based indexes emulate the suffix tree of their input text in small space, and allow $O(l)$ time queries for sequences of length l, irrespective of the size of the target corpus. Implementations of the popular FM-index [78, 80] and the functionally equivalent compressed suffix array (CSA) [103] compress the BWT using encodings that allow fast decompression or direct operation on the compressed data (compressed for example with run-length compression, or that of Raman, Raman, and Rao [224]), and augment the BWT with positional information that allows the index to be used for substring matching.

Indexing the sequence space of variation graphs requires a generalization of the principles behind these indexing techniques. Junctions allow for the representation of alternative sequences, with an exponential size relative to the number of furcations in the graph. This introduces problems of representation and scale which have required substantial work to resolve. In `vg`, I was ultimately to encourage the development of and use a practical implementation of a succinct data structure akin to the CSA, the GCSA2. This model was developed within the context of `vg`, and is the first sequence index specifically designed to work on variation graphs. It generalizes concepts and algorithms from the CSA to any kind of bidirectional sequence graph. Here, I will provide a description of related data structures from which this model draws inspiration, as well as a careful summary of the GCSA2 and the features which make it ideal as the core sequence index to drive a short read to variation graph mapper. Due to space, I will not fully elaborate this model, as it is not among my contributions. Readers are encouraged to examine the cited works if they seek a complete description of this and related data structures.

2.4.3.1 Graph k-mer indexes

It is straightforward to construct a k-mer index of a variation graph. We simply enumerate the k-length walks through the sequence space of the graph. Problematically, the k-mer space of a graph can grow exponentially where variant bubbles cluster within the given length k. To mitigate this issue, we can short-circuit the k-mers enumeration when a given number of edges are crossed within k characters. Another way to mitigate the limitations posed by the exponential number of k-mers in the graph is to build the index on disk, rather than using main memory. In the early stages of the `vg` project, I implemented a k-mer based index using a disk-backed system (see section 2.4.5). By sorting the k-mers, the index could be efficiently compressed, but still the graph for the human 1000 Genomes Project required over 200GB of memory, and was too slow to use unless cached in main memory. This index formed the basis for the first prototypes of the `vg` mapper, but is impractical due to its time and memory requirements.

2.4.3.2 The FM-index and Compressed Suffix Array (CSA)

As introduced in section 1.2.2.1, compressed full text self indexes like the CSA and equivalent FM-index provide functionality similar to that of the suffix tree (as in Figure 2.3) but in compressed space. The functionality of these indexes is the basis for the indexing techniques applied in `vg` to the alignment of reads to variation graphs. Here, I will first illustrate the construction of a full text index based on the Burrows Wheeler

transform (BWT) and suffix array (SA), and show how it can be used to locate occurrences of a pattern in the source text. This discussion will form the basis for an elaboration of the generalizations that lead to a similar kind of index built over a variation graph.

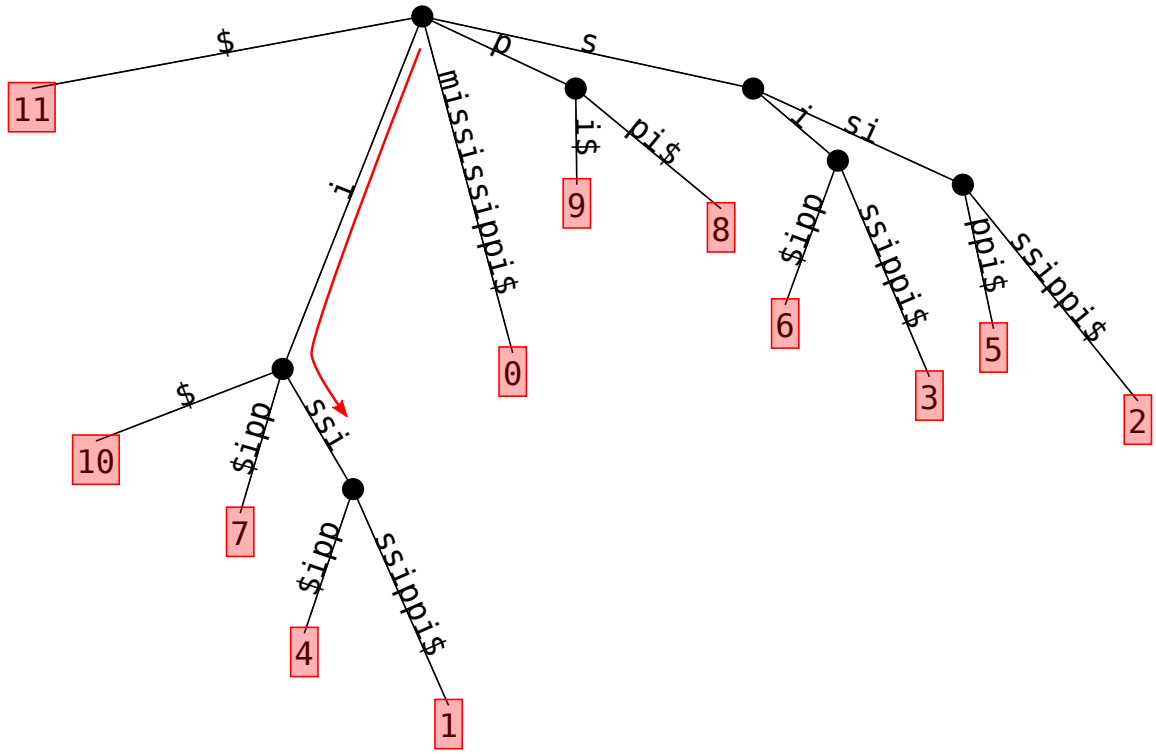


Fig. 2.3 A suffix tree built from the suffixes of the string `mississippi`. Suffixes begin at the root, shown at the top. Leaves are labeled by their suffix, which is written in terms of the 0-based starting position of the suffix in the source text. A red arrow shows a search for the string "iss" on the tree. The leaves of the tree below the ultimate point reached by this search give the positions of the matched pattern in the source text, in this case 1 and 4. Rendered with https://github.com/mpetri/draw-suffix-tree.

To construct the BWT from a given string T, we first append a marker character that is outside the alphabet (e.g. `$`) to the text. We take all rotations of this string, and lexicographically order them to form a matrix, which is sometimes referred to as the Burrows Wheeler matrix (BWM). The first column of this matrix is F, and lists the characters in the source text in lexicographic order. We can compactly represent it as a vector of counts C of characters that are lexicographically smaller than a given one, indexed by the characters as encoded in a compact set of integers. The last column of the BWM is the Burrows Wheeler transform (BWT) of the source text. The order in which the rotations (or equivalently, suffixes) appear in the sort is the suffix array (SA)

of the text. Assume a function $rank_{BWT}(c, i)$ which returns the number of characters equal to c in the prefix of $BWT[0 \ldots i]$. The function $LF(i) = C[c] + rank_{BWT}(c, i) - 1$ reconstructs the mapping between the same character in the BWT and F, thus provides a way to unwind the permutation generated by the sort and reconstruct the source text. These construction steps are illustrated in Figure 2.4.

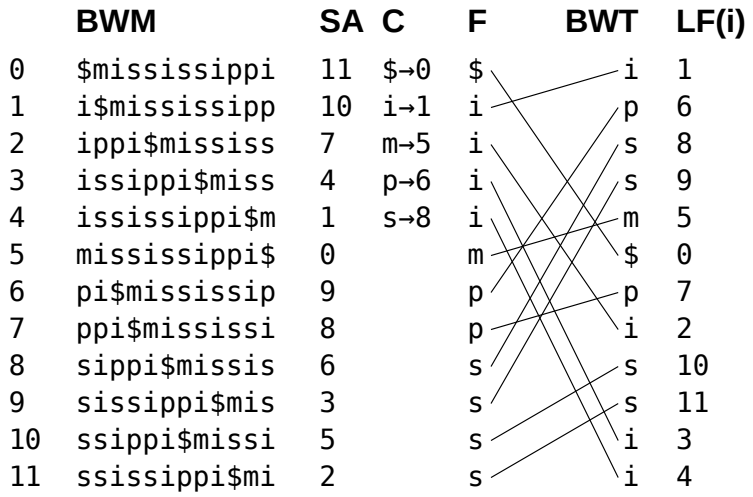


Fig. 2.4 A BWT and suffix array built from the string `mississippi`. The Burrows Wheeler Matrix (BWM) shows all rotations of the string (appended with terminal marker `$`) in lexicographical order. SA provides the suffix array, which lists the suffixes relative to the original string in their sorted order in the BWM. The first column of the BWM corresponds to vector F, which contains a series of runs of single characters in their lexicographic order, and is compactly represented in C. We find the BWT in the last column of the BWM. The function $LF(i)$ is shown for each position in the BWT, and lines drawn between F and BWT show the characters in the system that have the same identity in the source text. Produced using https://github.com/ekg/drawbwt.

We can iteratively apply LF mapping to execute search on the BWT. Algorithm 1 defines the function $find(Q)$, which returns the suffix array interval (or lexicographic range) in which suffixes are prefixed by Q, should it exist, and an empty interval if Q is not a substring of T. We begin our search by looking at the SA interval prefixed by the last character in our pattern, which is trivially obtained from array C. The body of the loop constitutes one step of backward searching. Searching works by maintaining the invariant that the prefix of the suffixes in the SA interval defined by $[sp, ep]$ is that given by the set of characters that we have considered in reverse order from the query string Q.

The addition of the longest common prefix (LCP) array on the sorted suffixes allows the CSA to emulate all algorithms on suffix trees [2]. This augmented data structure provides a compact encoding of the full suffix tree topology. In `vg`, an equivalent

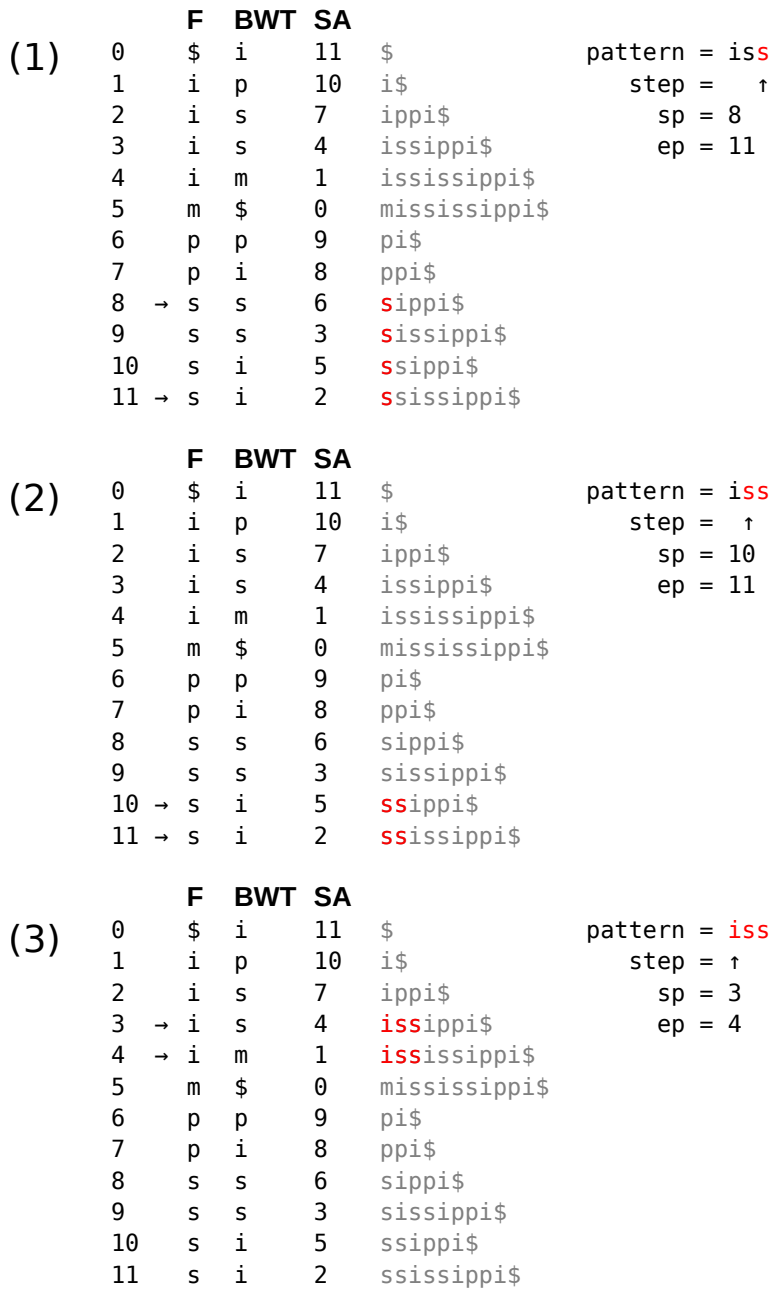


Fig. 2.5 Using backward search to find all occurrences of `iss` in `mississippi`, using the BWT and suffix array constructed in Figure 2.4. The index structure is shown in each of the three required steps (1→3), which progress from top to bottom. At each step, we consider a character, beginning from the end of the string we are searching. We apply LF mapping to find the new SA interval corresponding to the backwards extension of our query pattern. The subsequence matched up to the current step is shown in red in the sorted suffixes (in gray) represented by F, BWT, and SA. We do not store these sorted suffixes, as they are given by the BWT and SA. Nor do we store F, as the C array is sufficient to reconstruct it. Rendered with https://github.com/ekg/drawbwt.

Algorithm 1 Backward searching on the BWT

```
function FIND(Q)
    c ← Q[|Q| − 1]                          ▷ Current character in our search
    sp ← C[c]                               ▷ Beginning of our SA interval
    ep ← C[c + 1] − 1                       ▷ End of our SA interval
    for i ∈ [|Q| − 2 . . . 0] do            ▷ Step backwards through the string
        c ← Q[i]                            ▷ Update our character
        sp ← C[c] + rank_BWT(c, sp − 1)     ▷ Update sp dependent on c
        ep ← C[c] + rank_BWT(c, ep) − 1     ▷ Do the same for ep
        if sp > ep then return ∅            ▷ We do not find Q in our text
        end if
    end for
return [sp, ep]                             ▷ Return the SA interval prefixed by Q
end function
```

generalization is used to enable the determination of maximum exact matches between query P and the sequences of a graph.

2.4.3.3 BWT-based tree and graph sequence indexes

In contrast to k-mer based indexes, generalizations of the CSA to graphs have required substantial conceptual development to produce. But, they have yielded compact, efficient data structures that require similar space to their linear counterparts while supporting a wide array of query patterns.

Most generalizations of succinct self indexes from strings to graphs draw on the XBW transform [82], which provides a compressed self index of trees based on a generalization of the FM-index [83]. In the XBW transform (figure 2.6), rather than considering all suffixes of a linear sequence in the construction of the BWT, we sort all concatenated paths from the root to leaves of each node in the tree and use this as the basis for the BWT in a manner analogous to the use of sorted suffixes for a linear sequence. To record the topology of the tree and support its navigation relative to the resulting BWT, auxiliary bitvectors record which edges connect to leaf nodes e (or equivalently, an expanded alphabet as originally presented in [82]), and which are the last edge among their siblings: $\mathcal{F}$. Rank and select queries on these bitvectors allow the generalization of the *LF-mapping* (last-first) permutation and its inverse Ψ to work on the tree. The lexicographic ranks of the standard versions of these functions are mapped into the appropriate BWT ranges using rank and select queries on $\mathcal{F}$.

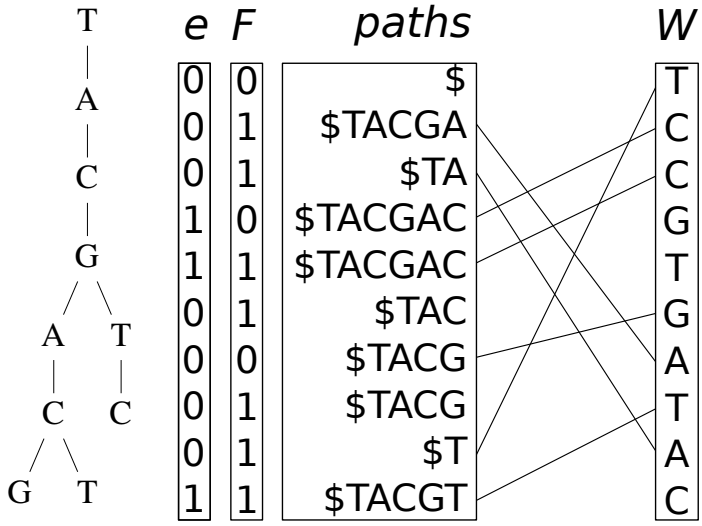


Fig. 2.6 The XBW transform and the LF function computed for the given graph. Nodes are stored in W, while bitvector e marks the leaf nodes (whose LF-mapping is $\emptyset$), and bitvector $\mathcal{F}$ marks the last nodes in each set of siblings. By construction, siblings are sorted together in the set of sorted paths. Paths are not stored, but are shown here for clarity. Example adapted from a public talk by Alex Bowe, posted at https://pdfs.semanticscholar.org/e42a/daab2806d5c0d715cb812c3a10aa26a2c75a.pdf.

Succinct de Bruijn graphs (SDBG) extend the XBW model to support DBGs by recording both in and out degree for each node [23, 192].[8] In `vg`, we do not use the SDBG model directly, but it is important to describe here as the GCSA2 model draws heavily on its design.

To construct the SDBG (illustrated in figure 2.7), we build two sorted lists of tuples representing the edge and k-mer space of the DBG $G = (N, E)$. First, we pad the graph with additional nodes labeled by a character lexicographically lower than the alphabet of the graph, so that each k-mer in G is now reachable from a path at least $k - 1$ long. This ensures that we can reconstruct the graph without loss. In F, we record the list of G's edges sorted co-lexicographically by the reverse sequence of their *ending* node and their label. In L, we record the list of G's edges sorted co-lexicographically by the reverse sequence of their *starting* node and their label. The edge-BWT (EBWT) is the sequence of edge labels as given in L.

Navigation of the graph is provided by the mapping between F and L that is based on the relative stability of sorting of edges with the same label in the two lists. If an

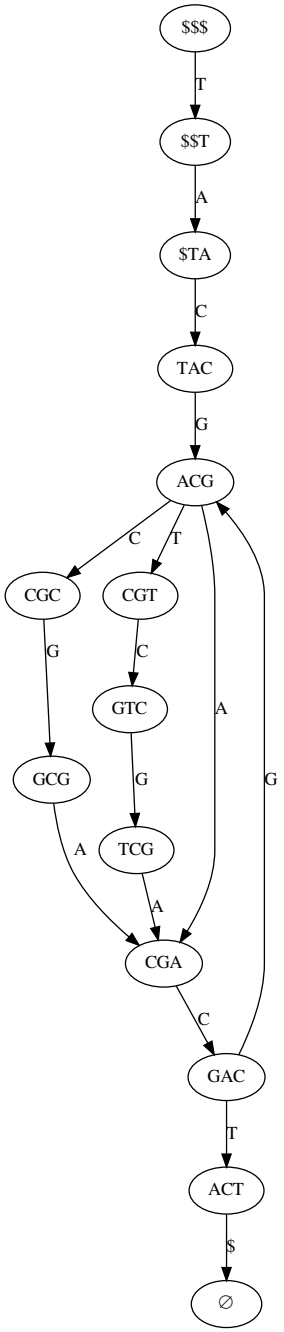
[8] See https://alexbowe.com/succinct-debruijn-graphs/ for a high-level overview of Bowe's initial model for succinct DBGs, which explains in detail how the various graph traversal and query operations are implemented.

edge e is in position p in L, and $l(e)$ is a function that gives the character label of edge e, then its position in F is given by $|d : d \in E \wedge l(d) \prec l(e)| + rank_{EBWT}(l(e), p) - 1$. Note that this function corresponds to LF mapping on the BWT. A similar extension of this idea affords backward search of patterns in node labels in the SDBG. For full navigation, we must edge topology bitvectors B_F, which marks the last edge for each node in F, and B_L, which does the same for L. We can traverse the graph using these structures. Given a character c and the co-lexicographic rank of a node, we can use B_L to find the interval in L containing its node's outgoing edges, then find the edge e labeled c in the $EBWT$, and finally use B_F to find the co-lexicographic rank of e's ending node, should it exist.

2.4.3.4 The Generalized Compressed Suffix Array

In section 1.4.3.6, I discuss a variety of indexing models that support indexing reference genomes and variants. Many schemes are possible, but the first to provide a conceptually complete solution to the problem of establishing a sequence index for a structure similar to a variation graph is Sirén's Generalized Compressed Suffix Array (GCSA) [258]. This structure allows for queries of arbitrary length within a directed, acyclic variation graph (equivalently, a finite language model or MSA) in much the same way as the FM-index or CSA do for linear sequences.

To build the GCSA, we first construct a sequence labeled DAG (which Sirén describes as a MSA) from the reference and a set of genetic variants. As in the succinct DBG, we attach marker nodes to the graph for its head and tail. The MSA is converted into a reverse deterministic automaton via a standard procedure for determinizing finite state automata. This results in a structure where there is no more than one of each character in the alphabet among the predecessors of each node. Recall (section 2.4.3.2) that backward searching using the BWT requires that positions in the text containing a given character c are sorted in the same order as text positions preceded by character c. Thus, to provide equivalent functionality for an automaton, we must ensure that each node in the automaton will be stably sorted when we enumerate and sort its suffixes. This implies a transformation of the automaton that ensures that each prefix of a path through the graph is the only way to generate a suffix with the particular prefix. Sirén implements this prefix-sorting transform in an algorithm that k-prefix-sorts the graph for a given k. The process is iterated for $k' = 2k$ until k' is greater than the length of the longest path in the automaton. At this point, it is possible to enumerate and sort the suffixes of the given automaton, build the corresponding BWT, and augment the data structure with bitvector $\mathcal{F}$, which allows a single node to have multiple predecessors, and bitvector $\mathcal{M}$, which records the number of successors or outgoing edges from each node



F	B_F	L	EBWT	B_L
$$$		$$$	T	1
CGA	A	CGA	C	1
CGA	A	$TA	C	1
CGA	A	GAC	G	0
$TA	A	GAC	T	1
GAC	C	TAC	G	1
TAC	C	CGC	G	1
CGC	C	GTC	G	1
GTC	C	ACG	A	0
ACG	G	ACG	C	0
ACG	G	ACG	T	1
GCG	G	GCG	A	1
TCG	G	TCG	A	1
ACT	T	$$T	A	1
CGT	T	ACT	$	1
$$T	T	CGT	C	1

Fig. 2.7 On the left, a de Bruijn graph with $k = 3$, with edges labeled by the character transition that they imply. Right, the construction of the succinct de Bruijn graph for the given graph. F indicates the co-lexicographically sorted set of edges keyed by the node that they enter. L indicates the co-lexicographically sorted set of edges keyed by the node that they leave. B_F marks if the edge in F is co-lexicographically the last for its corresponding node, and B_L marks the last edge for a given node in L. The "edge BWT" or $EBWT$ is given as the set of edges ordered as in L. The SDBG is given by B_F, B_L, and $EBWT$. Example adapted from [192].

in the automaton. Backward searching in this structure (LF) uses a similar principle to the XBW and succinct DBG models. We first use bitvector $\mathcal{F}$ to convert lexicographic ranks to a BWT range, and then we use $\mathcal{M}$ to convert the edge range to lexicographic ranks.

2.4.3.5 GCSA2

Sirén's update to GCSA, GCSA2, is specifically designed for application to arbitrary variation graphs [256]. It is the first index to provide an emulation of the full range of suffix tree operations in the context of a sequence graph. The crucial observation that drove the development of GCSA2 is that it is not necessary to enable full length queries of the graph for a graph path index to be useful. Shorter graph path queries are fully sufficient as the basis for sequence mapping to the graph, which is the primary application of a graph sequence index.

GCSA2 brings together ideas from succinct DBGs and the GCSA model. At a high level, we can consider it to be a compact de Bruijn graph k-mer index of a variation graph. However, in implementation several transformations are required to maintain acceptable bounds on the space required by this model. The de Bruijn graph is *pruned* by using strings smaller than k characters as nodes so long as the shorter strings still uniquely identify the corresponding paths in the graph. GCSA2 transforms the pruned DBG into a BWT, using additional bitvectors IN and OUT analogous to F and L in SDBG to record the graph topology in a generalization of the FM-index. The full data structure includes extensions to allow suffix tree operations that are important for finding maximal exact matches during sequence search.

The GCSA2 model is more flexible with respect to the complexity of input graphs than GCSA. GCSA2 uses disk backed construction methods to allow the indexing process to scale to very large graphs. The k-mer length limitation of the index allows it to be applied to any variation graph, including those that have cycles or other regions of topological complexity, with the caveat that we cannot search for sequences of length greater than k without the risk of finding false positives. Denser graphs may be indexed by limiting the maximum query length. The DBG transformation provides flexibility by decoupling the graph we are indexing from the index structure itself. For instance, edge pruning can be applied to the input graph to remove regions of local complexity, yet it is still possible to generate an index from such fragmented subgraphs because we can always generate a DBG by enumerating k-paths through a variation graph. Unlike competing graph indexing methods, such as BWBBLE and the vBWT (section 1.4.3.6), GCSA2

avoids exponential costs during backwards search. However, it does incur exponential costs in construction of the index, as it must enumerate all k-paths in the graph.

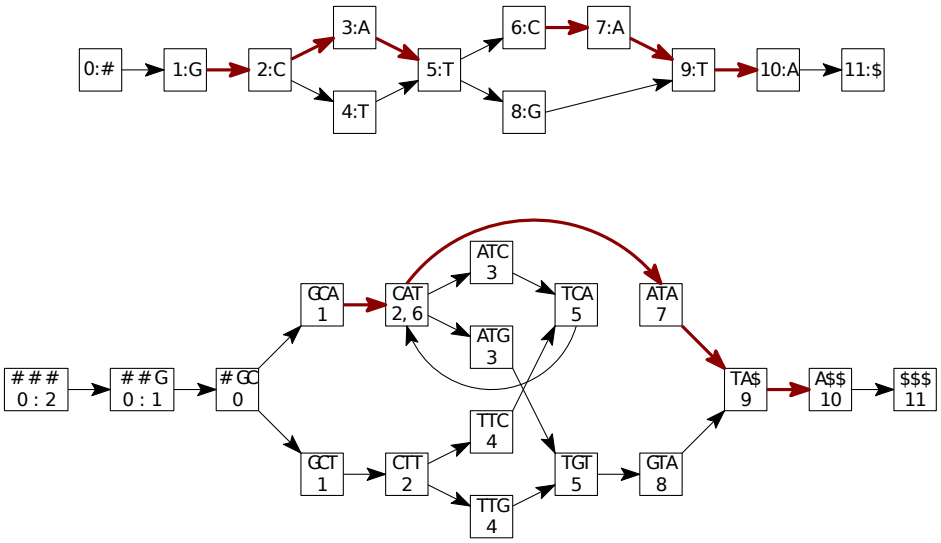


Fig. 2.8 A sequence graph (top) and its de Bruijn transformation (bottom) for $k = 3$. The highlighted path in the DBG is a false positive, as it consists of two disjoint paths in the input graph, and is shown to demonstrate the fact that the DBG is not a lossless representation of the input for any length greater than k. Reprinted from [256].

To construct the GCSA2 index from variation graph G, we transform it into a de Bruijn graph whose nodes are the full set of k-length walks through the G (as in figure 2.8) on both strands. Before doing so, to ensure that all elements of the graph are included in k-length paths, we add head and tail nodes labeled # and $ to the variation graph prior to this transformation. To maintain the lexicographic ordering invariance required for searching in the GCSA2, we add a special edge connecting these nodes prior to generating our k-mer set.[9] To allow larger query lengths without direct in-memory enumeration of the full length paths in the source variation graph, GCSA2 uses a series of disk backed steps to double the DBG k until it reaches the desired length. In practice, it uses four rounds of doubling, beginning with $k = 16$, and progressing through $2k$, $4k$, $8k$, and $16k = 256$. At each step, the DBG is pruned to remove redundancy and reduce memory usage (compare the graph in figure 2.9 to the DBG in 2.8). The final FM-index-like model is encoded using succinct data structures from SDSL-lite [95], with succinct bitvectors used for rare characters like `N`, $, and #.

[9] This can be seen in the LF mapping drawn between BWT and OUT in figure 2.9, where a line connects $ with #.

The final GCSA2 data structure model (illustrated in figure 2.9) consists of: the BWT, which (as in SDBG) represents the edge space of the pruned DBG; IN, which marks which edge is the last among the inbound edges to a given node; OUT, which marks which edge is the last among the outgoing edges from a given node; and positional samples connecting the DBG nodes with positions in the input graph G. As in the SDBG and GCSA, the nodes of the pruned DBG are not stored, but given implicitly in the number of 1-marked bits in IN and OUT. This can be seen in the relationship between the "key" vectors in figure 2.9 and the BWT vector. A vector (not shown in figure 2.9) C records the lexicographic ranges of the starting characters in the node labels, serving the same purpose as the equivalent vector in the FM-index. For the index to be useful for graph path queries, we should be able to link lexicographic ranges within the BWT to positions in the original graph. To do so, we store a set of positional samples as: a vector B_S, which indicates those nodes for which we have positional samples; a unary encoding of the number of values stored for each node B_V; and the vector of positional samples themselves V_S. Positional samples are stored at the case of branches in the pruned DBG, or at some configurable frequency otherwise. In linear regions of the pruned DBG, it is possible to use graph DBG graph traversal operations to compute un-sampled positions, thus trading off time and space as is typically done in FM-index implementations on linear strings.

The GCSA2 encoding is highly efficient on real data sets. When including the suffix tree extensions, GCSA2 uses around 1 bit per k-mer in indexes of the 1000GP pangenome graph for $k = 128$, which is favorable with comparison to SDBG indexes [256]. To appreciate the costs of indexing a human genome sized graph[10], the order-256 GCSA2 index of the 1000GP variation graph may be constructed using less than 500GB of scratch space, to and from which are written approximately 3TB during construction, all while requiring less than 50GB of RAM. This puts GCSA2's indexing resource requirements well within the specifications of standard commodity compute servers. The resulting index occupies between five and ten times the input graph's serialized size, no more than 50GB for a human genome. Although it could appear to be a critical limitation, the limit on query length is not a problem in practice. Few contemporary reads are likely to generate 256bp-long sequences with no mismatch from the reference[11], this approach effectively allows us to find all the exact matches for a typical sequencing read.

[10] Precise sizes are given later in the discussion of results.

[11] Illumina's reads rarely reach 256bp, and when they do they tend to have higher error rates in the later cycles. The 10-15% error rate of PacBio and ONT sequencing mean that a 256bp exact match is extremely unlikely, although PacBio circular consensus reads (CCR) may approach this level of accuracy.

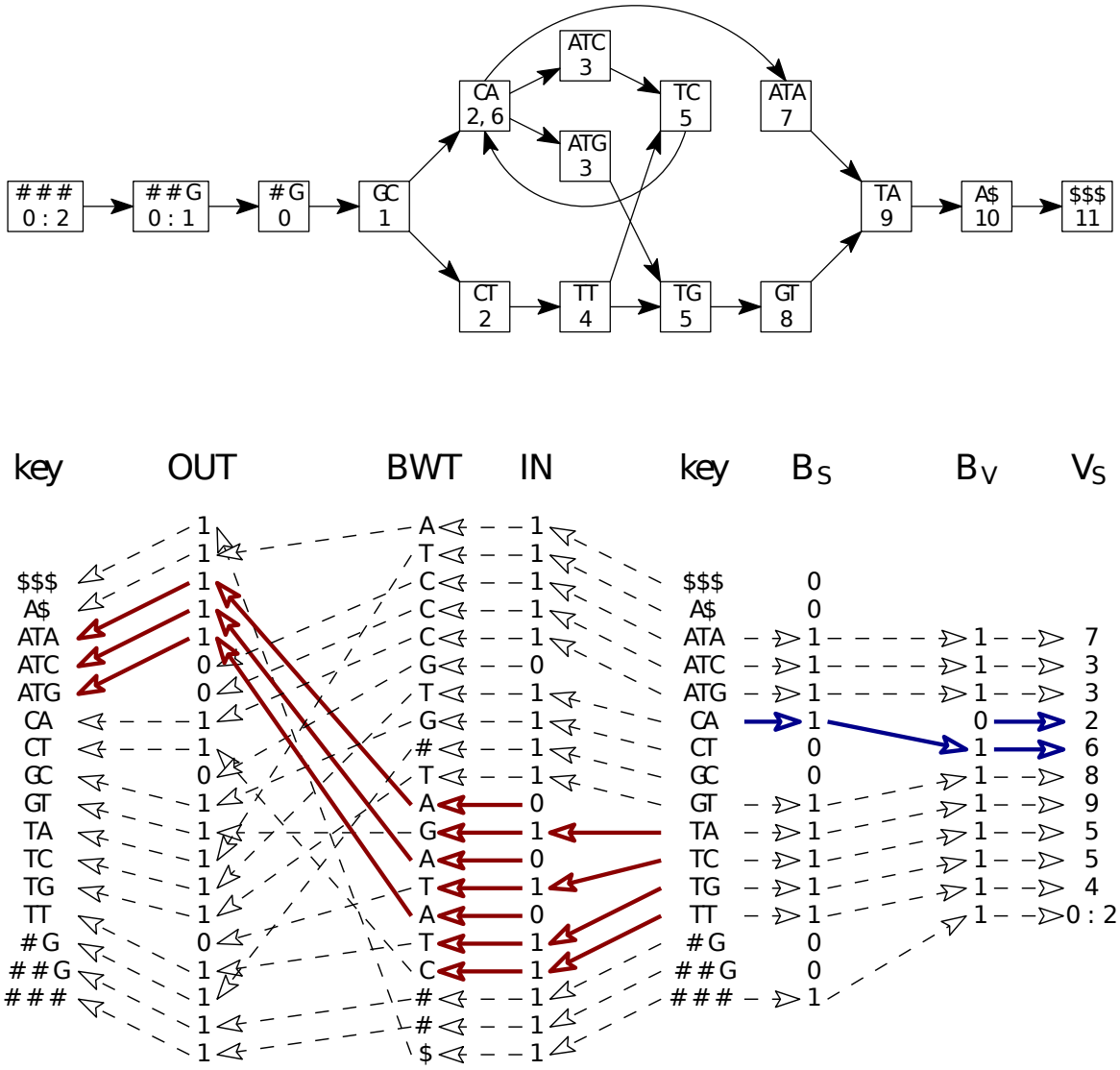


Fig. 2.9 Top: An order-3 pruned de Bruijn graph 3-equivalent to the de Bruijn graph in figure 2.8. Bottom: the GCSA2 for the given graph, including both stored elements (IN, OUT, BWT, B_S, B_V, and V_S) and the implicit node sequences shown as the vectors labeled "key". Not shown, a vector C encodes the number of nodes lexicographically smaller than a given character, supporting backward search and LF-mapping as in the normal FM-index. Leftward red arrows illustrate backward searching, showing the steps taken from `T` to `AT`. As in algorithm 1, the range of nodes beginning with `T` is found from two queries on C: $[sp, ep] = [C[T], C[T+1]-1] = [9, 12]$. The lexicographic range in BWT corresponding to this interval is found by $[select_{IN}(1, sp-1)+1, select_{IN}(1, ep)] = [10, 16]$. We then apply LF-mapping (as in algorithm 1) to find the range in OUT corresponding to the pattern `AT`. Finally, rank queries on OUT allow us to convert to a lexicographic range of nodes prefixed by the pattern `AT`, although in this case the mapping is trivial: $[sp, ep] = [rank_{OUT}(1, 2), rank_{OUT}(1, 4)] = [2, 4]$. Rightward blue arrows mark the samples belonging to each node, with the blue ones showing them for node `CAT`, which does not correspond to the given backward searching steps, but which would be found by another round of backward searching on `C`. Reprinted from [256].

2.4.4 Haplotype indexes

Recording the path set of a graph in XG (described in section 2.4.2) requires $O(N\overline{P}+\mathcal{L}|P|)$ space where $\mathcal{L}$ is the average haplotype length in terms of nodes it crosses, N is the number of nodes in the graph, $|P|$ is the number of paths in the graph, and $\overline{P}$ the average number of paths crossing each node. Although, such a representation can be compressed, the size of this representation will grow linearly with the addition of new paths, making it impractical as a means to record very large numbers of genomes.

Recording collections of paths is an important requirement for the use of VG in resequencing, as the Markovian property of the bare sequence graph $G = (N, E, P = \emptyset)$ means it can encode exponentially many paths relative to the true input path set used to build the graph. This introduces significant issues during read mapping and genome inference. With increasing variant density the number of possible sequence paths of a given length grows exponentially, and this can lead to spurious mismapping (section 3.2.2). The exponential growth of the path space of the graph has relevance for sequence indexing with GCSA2, and as described in 2.4.3, simplification of the graph in complex regions prior to GCSA2 indexing is required to build indexes in practice. A haplotype index allows the pruning operation to preserve known haplotypes, rather than defaulting to the reference genome in such cases. Efficient path indexes could be used for many operations in variant calling and phasing, and may have utility in assembly problems, for instance to losslessly record a read set embedded in a variation graph representing their mutual alignment (section 2.2.6).

Haplotype sequences from the genomes of the same species often share extensive regions of homology, which suggests that they may be very efficiently compressed. This property was used to store large haplotype sets in the *positional BWT* (PBWT) [65]. As input, the PBWT assumes a set of haplotype strings $S_1 \dots S_m$ of the same length which describe a set of haplotypes relative to a set of variable loci. $S_j[i]$ records the allele in haplotype j found at locus i. We set $S_j[i] = 0$ when haplotype j has the reference allele at locus i, and $S_j[i] > 0$ if it encodes one of possibly several alternate alleles. The PBWT can be understood as an FM-index of texts $T_1 \dots T_m : T_j[i] = (i, S_j[i])$ [87]. To search for a haplotype of h in the range $[i, j]$ we look for pattern $h' = (i, h[1]) \dots (j, h[|h|])$. The alphabet size of this FM-index is large, but the matrix like structure of the haplotype set means that we can implicitly encode the array indexes by building a separate sub index for each position. Applying run length encoding to the BWT allows extremely good compression of real haplotype sets.

With the *graph positional Burrows–Wheeler transform* (gPBWT) [202], we extended this model to work on variation graphs. The basic model is the same as the generic

PBWT except that instead of variant matrix positions we consider haplotype traversals of oriented nodes[12] n_i or $\overline{n_i}$, and rather than a local alphabet of variant alleles we encode a local alphabet $\Sigma_{n_i} = \{j \in N | e_{ij} \in E\}$ which describes the set of nodes $n_{\{j \in N | e_{ij} \in E\}}$ to which haplotypes continue immediately after the current node n_i. As in the generic PBWT we build an FM-index of $T_1 \dots T_m$, encoded in what we call the B_s arrays, which provide the local description of prefix sorted haplotypes (equivalently, threads) traversing each node n_s. To deal with the bidirectionality of paths in variation graphs, each haplotype must be encoded in its forward and reverse orientation. In [202] we demonstrated the expected sublinear scaling of the gPBWT by building an index for chr22 with increasing numbers of samples. Constructing the gPBWT for haplotype sets representing more than a few hundred samples proved difficult when using our particular implementation. Progressive construction of the gPBWT in generic graphs was enabled by encoding the gPBWT into dynamic succinct data structures and adding a single haplotype thread and its inverse one at a time. While functional, we found this to be untenable for large graphs and haplotype sets. Overheads associated with the dynamic data structures it uses were significant, but the most-difficult issue was the serial nature of the progressive construction algorithm, which gives the algorithm $O(m)$ runtime. Consequently, all our large-scale experiments were carried out using a partially ordered construction algorithm that worked using a VCF file as input.

The *graph Burrows-Wheeler transform* (GBWT) [259] simplifies the data model used by gPBWT so that it is independent of `vg`. Conceptually, the GBWT can be understood as the FM-index of a transformation of the graph's paths $p_1 \dots p_m \cup \overline{p_1} \dots \overline{p_m}$ into the text $T = \$(p1 = n_i \dots n_j) \dots \$(\overline{p_m} = \overline{n_j} \dots \overline{n_i})$ wherein the paths are rewritten as a series of characters representing node traversals in a large alphabet and delimited by a marker \$. This approach is challenging due to the large size of T for moderately-sized haplotype sets embedded in variation graphs, e.g. $|T| \approx 10^{12}$ for the 1000GP [259]. Literally implementing this model would require a large alphabet CSA with suboptimal performance bounds. Serializing the path set during construction is not feasible, which suggests a dynamic version of the model is required during large scale construction. Natural variation graphs have a number of properties, such as a manifold partial order, which ca be exploited to improve the memory usage of the GBWT.

In the GBWT we break the full FM-index into per-node records, each of which encodes a header defining an alphabet Σ_{n_i} of all the nodes that follow n_i in any path, and a body BWT_{n_i} which is the subset of the full BWT specifying which node follows n_i in each path that passes through n_i, with the paths sorted in reverse sequence order up

[12] These are described as "sides" in [202].

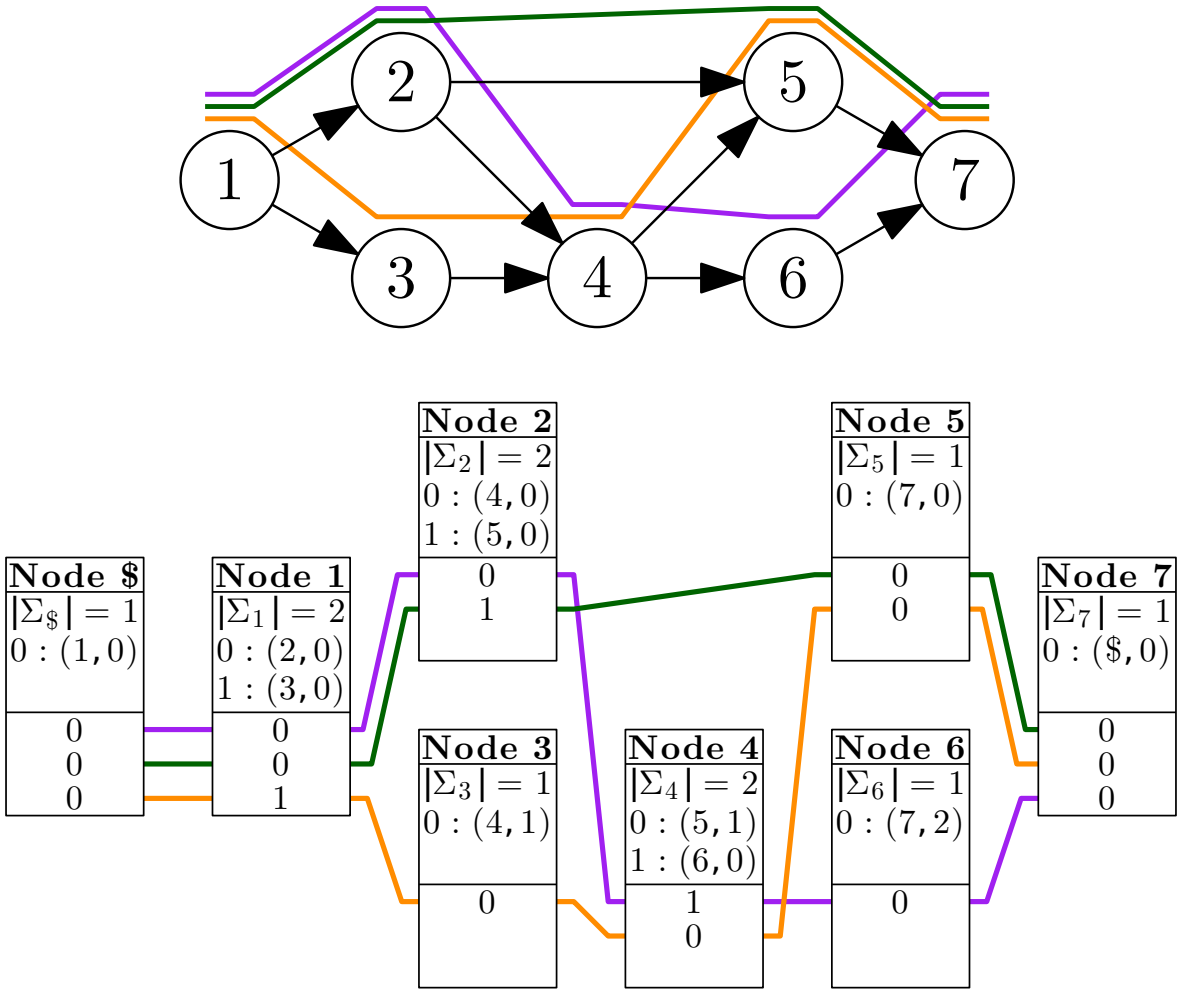


Fig. 2.10 Top: A graph with three paths, each represented as a colored line walking above a series of nodes, where $N = n_1 \dots n_7$. Bottom: GBWT of the paths. To build the GBWT we first append a marker node, $n_\$$, to the head of the graph. Each node in the GBWT is represented by a record, shown as a white box. At the top of each record we find the node identifier, which is implicitly stored in the actual GBWT. Each record consists of a local alphabet described in a header and a body consisting of the subset of the full GBWT applying to the given node. The alphabet Σ_i maps nodes which paths crossing this node reach in their next step into a more compact alphabet. It is encoded as a mapping between next node identifier i, the local character used to represent that node $\in [0 \dots |\Sigma_i|)$, and the number of instances of i in the BWTs of all nodes which are lexicographically smaller than the current one. For instance, at n_6 we find $\Sigma_6 = \{0 : (7, 2)\}$, because n_7 is referred to by the orange and green path crossing the edge $n_5 \to n_7$. This arrangement encodes the full BWT in per-node local records. We see each path represented as it passes through the BWT record for each node. At each node, the sort order of the paths implicitly represented in the BWT reflects the lexicographic sort of their prefixes. For instance, at n_4, we see $n_{BWT} = [1, 0]$, because the prefix of the purple path before n_4 is encoded as $[0, 0, 0]$, which is lexicographically lower than the prefix of the orange path, $[0, 1, 0]$. Reprinted from [259].

until n_i. Since paths that are similar before n_i tend to be similar after it, this sequence of next node values run length compresses well.

The GBWT supports essential FM-index operations including: $find(X) \rightarrow [sp, ep]$ yielding the lexicographic range of suffixes starting with pattern X; $locate(sp, ep) \rightarrow$ paths occurring in $SA[sp, ep]$; and $extract(j) \rightarrow p_j$ which returns the jth path in the graph. By encoding all paths in both orientations, the GBWT can be treated as a kind of FMD-index for haplotypes, allowing bidirectional search. This means that the GBWT in turn supports MEM-based haplotype matching, which has potential uses in genotype imputation, phasing, association mapping, and other population genetic and evolutionary assays. Although supported, this particular modality has not yet been explored.

The GBWT representation reflects a number of assumptions that tend to hold for most DNA sequence graphs. Nodes tend to have low degree, which means the local alphabet size $|\Sigma_{n_i}|$ is small, and we can afford to decompress a small local alphabet encoding efficiently. Most nodes are not traversed more than once by each path, so the BWT_{n_i} remains small and can be accessed and modified in bounded time. Due to relatedness among individuals in many species, it is sensible to assume that haplotypes will be highly repetitive, which allows for efficient RLE encoding of BWT_{n_i}. The graph is sorted, and its identifier space has been compacted, which allows us to store the same information for the entire range of node identifiers in bounded memory with respect to $|N|$. The graph tends to be locally ordered in most places, which decreases the complexity of construction.

A *dynamic GBWT* implementation presents the node records through an index over the range of $[min(i : n_i \in N), max(i : n_i \in N)]$, for each linking to its header, body, incoming edges, and haplotype identifiers. Construction employs this model in a manner similar to RopeBWT2 [154], where batches of paths are insert into the index in a single step following the BCR construction algorithm [14]. This process includes the new paths in the dynamic GBWT by rebuilding each node record affected by the extension. By breaking the construction process apart for each chromosome and finally merging the compressed GBWTs, it is possible to build the GBWT for the entire 1000GP haplotype set in around 30 hours. When constructed, the GBWT may be encoded in a compacted but immutable form that uses less memory by representing the per-node model of the dynamic GBWT in a columnar model, for instance concatenating the node BWT vectors and header information for each node each into a single compressed integer vector. The resulting GBWT requires $\approx$ 15 GB, with around half allocated to the GBWT structure itself and half to haplotype identifiers. The index consumes less than 0.1 bit per node in

the stored paths, and we should expect this to improve when we build the GBWT for larger haplotype panels.

2.4.5 Generic disk backed indexes

I began the development of `vg` alone, starting with schemas for the data models, then building an index of the graph using the disk-backed key/value store RocksDB[13]. I transcribed the data model into namespaces and sorted arrays written into the key/value store. As compressors of various types could be applied to the sorted arrays backing RocksDB, the memory required for this approach was ultimately similar to that for the final indexing models that I present here. However, performance was far worse, and the initial version of the aligner based on these systems could not achieve correct results using reasonable amounts of time for large graphs. Ultimately, this flexible database model has remained important for some pipelines, in particular as a technique to organize alignments against the graph. Other workloads such as sequence queries were untenable for large genomes, with reliable performance only possible if the entire index of spaced 27-mers was cached in RAM, requiring nearly 200G in the case of the 1000GP variation graph. The sorted disk-backed array does have the useful property of allowing prefix queries of the k-mer set, but this can easily be attained with GCSA2. On the networked storage available in my institutional setting, the construction costs for disk-backed index models were usually much worse than those of the XG and GCSA2 models.

2.4.6 Coverage index

A coverage map, of the alignments to a VG is similar to the labeling required to implement "colors" on a DBG [118]. The coverage map loses information about the edge traversals and the paths taken through the graph, which could reduce the visibility of some kinds of variation within it. But in benefit, this simple model is efficient to use. The complexity of computing the coverage map is linear in the number of input alignments, and it requires $O(\sum_{\forall n_i \in N} |n_i|)$ space to store once built.

I developed an compact coverage index by mapping the sequence space of the graph into a vector and recording coverage across it for a GAM read set. During construction, a succinct format is employed to store each base's coverage in a single byte as long as it is below 255, and in a secondary hash table if it reaches or exceeds 255. Finally, a compressed integer vector is generated, which can be queried by graph position computed from the XG index of the graph. I extended this concept with a succinct "pileup" format

[13] https://github.com/vgteam/vg/blob/master/src/index.hpp

[161] generated from the edits against the graph. In this model edits in mappings which don't match the reference were serialized into a byte alphabet using Protobuf, such that each non-reference edit e_j at position b_i was recorded as a string $\$b_i e_j$, with the idea that by building a CSA/FM-index from these I could obtain the set of edits at each graph position through pattern matching. However, I found it impossible to construct this for a large high-coverage sequencing sample, and have not continued this line of investigation.

2.5 Sequence alignment to the graph

To align sequences to a VG, we use the graph and sequences indexes described in the previous section (2.4) to derive MEMs between a query and the graph. A weighted DAG collinear chaining model is built from the MEMs which respects their relative positions in the graph and the read, favoring collinear mappings of MEMs, and a max-sum DP algorithm is applied to this alignment DAG to extract likely mappings based on the MEMs. We then align the sequence locally to the graph at each of the high scoring chains using various sensitive alignment algorithms that use various kinds of dynamic programming.

To detect structural variation and align long reads without incurring quadratically-scaling computational penalties, we apply a kind of banding and a second layer of chaining. In *chunked alignment*, large sequences are broken up into overlapping segments, each of which is aligned individually in any order or orientation. This subdivision provides a kind of banding to the alignment algorithm, preventing the evaluation of the full DP matrix, but more importantly it also allows alignments that are generated to represent any kind of variation. Each chunk is aligned independently. The same collinear chaining model, with different parameters, is used to establish the optimal global chain through the alignment chunks, thus yielding a full alignment for an input sequence of any size. Where our reads are shorter than the standard chunk size (256bp), the alignment behaves exactly as in `bwa mem`. Figure 2.11 illustrates the alignment of a long read against a complex graph.

Unfolding and *DAGification* transform a cyclic bidirectional sequence graph with inversions to an acyclic simple sequence graph one in which all k-paths in the first graph are represented. Any alignment algorithm that may be implemented on a sequence DAG can thus be used. Optionally, other alternative DP alignment algorithms implementing a banded global alignment can be applied, and I describe one of these that I implemented to *surject* alignments into a particular reference path.

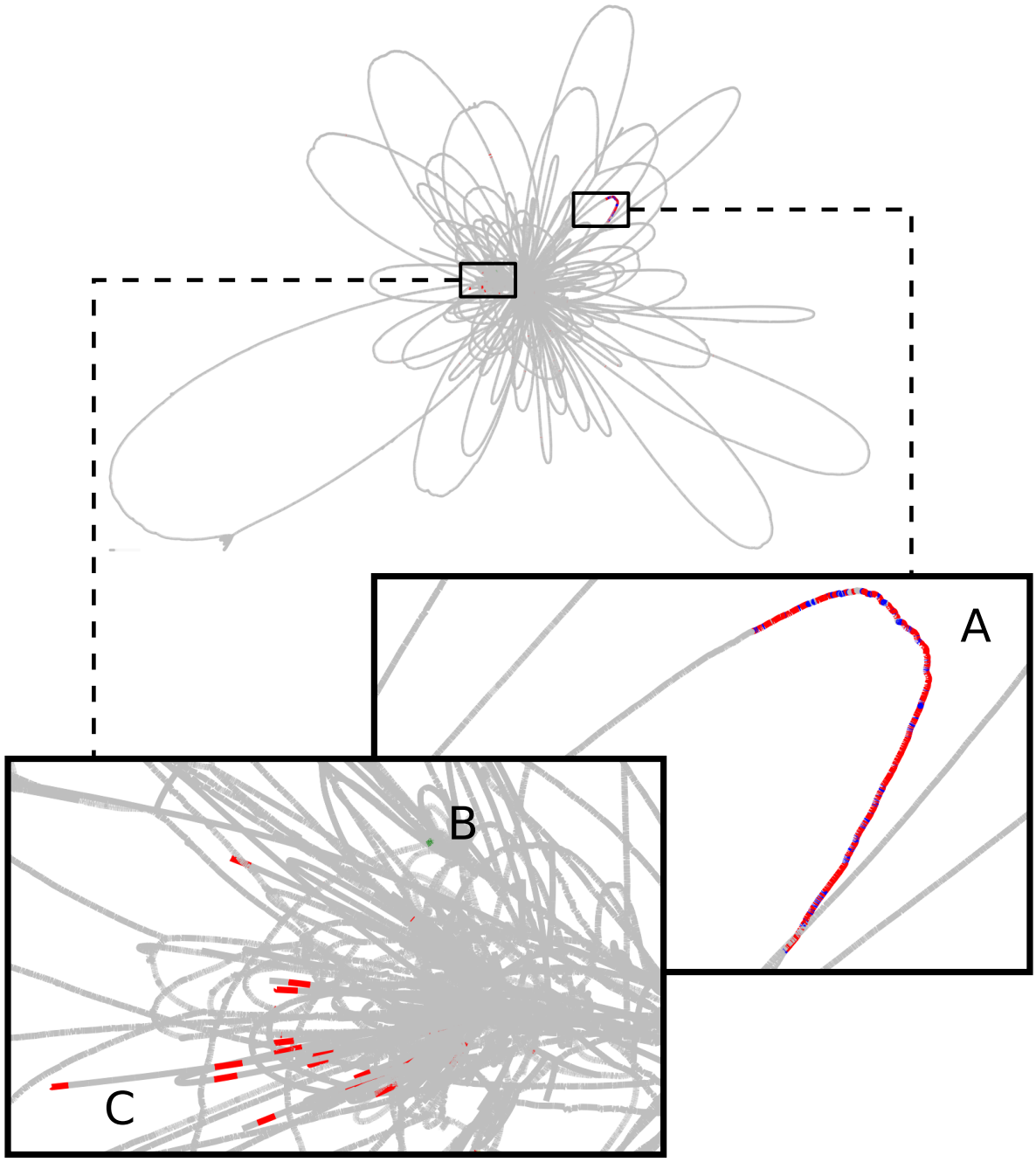


Fig. 2.11 Aligning a 32,737bp PacBio read from the SK1 strain to a yeast pangenome graph (section 3.1.2). Red nodes contain initial MEM hits, while other nodes are colored if they were matched during local alignment. (A) The nodes found in the best alignment are labeled in blue. (B) A much smaller secondary alignment is shown in green. (C) MEM hits for this read cluster in chromosome ends, which are seen as tips in the graph visualization.

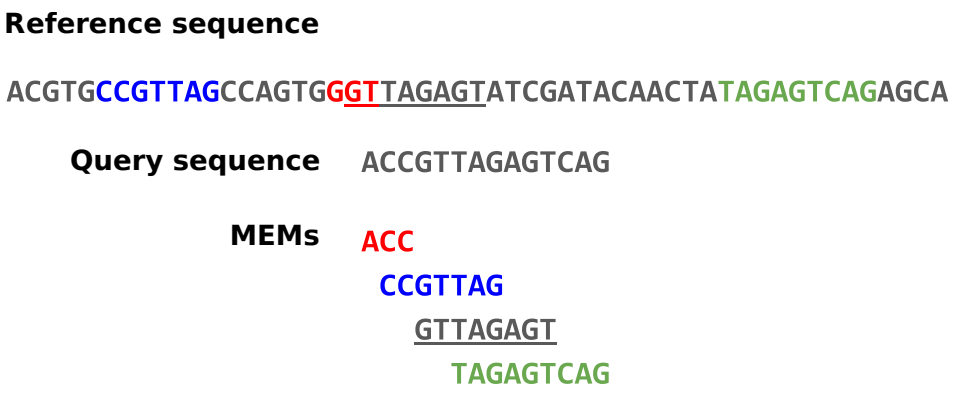


Fig. 2.12 Maximal exact matches (MEMs) found by the application of algorithm 2 to the given query sequence and a GCSA2 index built from the shown reference sequence. MEMs are listed below the query sequence with colors that match their matching location in the reference sequence. Note that one MEM is found on the reverse strand of the reference (`ACC` matches `GTT`).

2.5.1 MEM finding and alignment seeding

A set of super-maximal exact matches (SMEMs) of a query sequence are generated by backward search in the GCSA2 index. When backward search breaks, we step back to the last matching interval and use the suffix tree extension of the index to obtain the parent node of this interval. The GCSA2's `LCPArray` extension allows interrogation of the suffix tree structure, which in turn this allows us to remove the invariant sequence from the end of our previous matched range and continue search for the next maximal match. Algorithm 2 provides a sketch of this process as it relates to the GCSA2 FM-index encoding and LCPArray.

Not shown in this algorithm sketch are a series of "reseeding" passes whereby long MEMs are used as the basis for further exact match finding, but with a maximum limit to the detected match size. When backward search yields an exact match of this size, we use the same suffix tree operation to reset our match range with the next possible match. This reseeding operation is required to obtain high sensitivity via MEM based alignment seed lookup. At sufficient frequency to frustrate our mapping sensitivity in real genomes, a long MEM can mask out shorter sub-matches that might be contained within it, and which correspond to the correct mapping location. The resulting MEMs are not super-maximal, thus we tend to call these heuristically derived exact matches "MEMs" for simplicity.

We use MEMs to seed alignments in `vg`. As indicated in algorithm 2, the GCSA2 index supports lookup of matches by position in the original graph. In conjunction with

a distance estimator, these positions are used to build a collinear chaining model that allows us to estimate likely subgraphs matching our query (described subsequently in section 2.5.3). Chains of MEMs that are consistent with a mapping between the query sequence and the graph are found using a weighted graphical model in which the optimal alignment is likely to form a max-sum path. For each candidate chain, we then locally align the read against the graph. Scoring results from the local alignment are used to rank the candidate alignments. We then return the best alignment, or multiple candidates if multiple mappings are required, with calculation of mapping quality used to provide an estimate in our confidence in the best alignment (section 2.5.12).

Algorithm 2 Finding maximal exact matches using the GCSA2 suffix tree extension

function FINDMEMS(Q) ▷ Find the MEMs for the given query sequence Q
  $mems \leftarrow \emptyset$ ▷ The list of MEMs that we'll return
  $c \leftarrow Q[|Q| - 1]$ ▷ Current character in our search
  $sp \leftarrow C[c]$ ▷ Beginning of our SA interval
  $ep \leftarrow C[c + 1] - 1$ ▷ End of our SA interval
  $mem.begin \leftarrow |Q| - 2$ ▷ Build up a MEM structure to store our current match
  $mem.end \leftarrow |Q| - 1$ ▷ Record the current matching interval
  for $i \in [|Q| - 2 \ldots 0]$ **do** ▷ Step backwards through the string
    $last = [sp, ep]$ ▷ Store the last range, in case search breaks
    $c \leftarrow Q[i]$ ▷ Update our character
    $sp \leftarrow C[c] + rank_{BWT}(c, sp - 1)$ ▷ Update sp dependent on c
    $ep \leftarrow C[c] + rank_{BWT}(c, ep) - 1$ ▷ Do the same for ep
    if $sp > ep$ **then** ▷ The current extension has failed
      $mem.begin \leftarrow i + 1$ ▷ Remove the last (unmatched) character
      $mem.range \leftarrow last$ ▷ Set the range to the last matching range
      $mem.positions \leftarrow GCSA2.locate(mem.range)$ ▷ Get MEM positions
      $mems.append(mem)$ ▷ Store the last MEM for return
      $p \leftarrow LCP.parent(last)$ ▷ Get the suffix tree node parent of the last range
      $mem.end \leftarrow mem.begin + p.lcp()$ ▷ Remove the common prefix
      $[sp, ep] \leftarrow p.range()$ ▷ Use the SA range of the suffix tree parent node
    end if
  end for
  $mems.append(mem)$ ▷ Add the last MEM to the MEMs to return
return $mems$ ▷ Return the MEMs we've found
end function

2.5.2 Distance estimation

To cluster our MEMs we require a distance function that returns the minimum distance between any two positions $dist(b_i, b_j)$. Distance measurement between nodes in a vari-

ation graph is non-trivial, with exact solutions to the problem theoretically requiring $O(E \log\log L)$ where $L = \max_{\forall n_i \in N} |n_i|$ is the maximum node length [127]. Precomputation of the full set of distances would thus require $O(N^2 E \log\log L)$ time and $O(N^2)$ space, which is infeasible for any large graph. Many variation graphs are mostly linear, which we can exploit to build an approximate distance metric. Provided we have applied a partial sort to the graph, in the partially ordered regions we can use the offset of each node in the XG sequence vector $S_{\mathbf{iv}}$ as an approximate 1D coordinate, which we query using the corresponding rank/select dictionary $S_{\mathbf{bv}}$. We expect that in much of the graph $S_{\mathbf{bv}}^{select_1}(id(b_j)) + \text{offset}(b_j) - S_{\mathbf{bv}}^{select_1}(id(b_i)) + \text{offset}(b_i) \propto dist(b_i, b_j)$, where $id(b)$ is the function that returns the rank of position b's node in the XG index, and $\text{offset}(b)$ returns the position's offset inside the sequence label of the node.

Nonlinearities in the graph will frustrate this metric, and to manage these we rely on the positional index provided by the *positional paths* given in the XG index. In these, we can query the relative positions of nodes in the path in $O(1)$ time. Where both positions are not on the same path, we use a bounded local exploration of the graph near our positions b_i and b_j to attempt to find anchoring nodes on the same path. In our clustering step (section 2.5.3) we consider the multiple coordinate systems to develop a global pseudoalignment.

2.5.3 Collinear chaining

In most cases, single MEMs do not cover the full read length, so we need to combine information from multiple MEMs to determine a candidate mapping location. It is impractical to attempt a full DP based alignment at each MEM location, and instead, we apply a heuristic approach wherein a model is built with the available MEM seed information, their graph positions, their positions in the query sequence, and the scoring parameters used for local alignment. The *MEM Chain Model* is a DAG, $G_{\text{MemChain}} = (N, E)$, in which each node $n_i \in N$ derives from a MEM, and each edge $e_{ij} \in E$ to represent a possible transition between MEMs that we may find in an alignment.

Each n_i has a starting position in the query $mem\_pos(n_i)$ and a length $mem\_length(n_i)$. To estimate the alignment score that we would achieve by using this MEM in an alignment, we add a weight $\mathcal{W}_{n_i}$ to each node that scales this length by the match score used in local alignment $\omega_{\mathbf{match}}$:

$$\mathcal{W}_{n_i} = mem\_length(n_i)\omega_{\mathbf{match}} \tag{2.1}$$

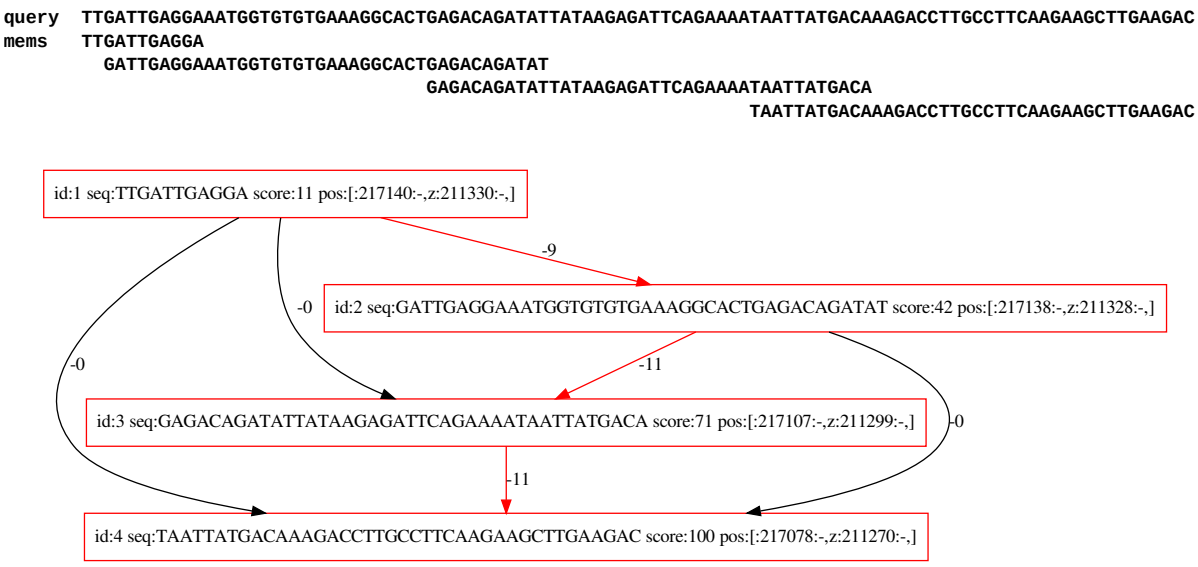


Fig. 2.13 Establishing collinear chains of MEMs to drive sequence read mapping. Above: a set of MEMs derived from a perfect read simulated from a 1Mbp test region from the 1000GP graph of chr20, using a maximum MEM length of 40bp to generate multiple MEMs for exposition. Below: the first evaluation of the MEM Chain Model for this set of MEMs. The model is shown in full, with the scores established as described in the text. Each node refers to a single MEM, its positions in the graph, and the score derived from the first pass of the max-sum algorithm. A weight is applied to each node equal to the length of its sequence times the match score used in local alignment. A score on each edge is computed as the gap open and extension score implied by the difference in distance between the MEMs in the read and in the graph positions, minus the overlap length of the MEMs in the read. The maximum scoring chain is shown in red.

Each n_i also has an associated set of graph positions $graph\_pos(n_i) = \{b_1 \dots b_m\}$. Edges e_{ij} in G_{MEMCHAIN} represent possible transitions between MEMs, and are weighted ($\mathcal{W}_{e_{ij}}$) by the minimum estimated distance between the given graph positions for each MEM, times the gap open and extension costs, less the overlap between the MEMs in the read times the match score:

$$dist_{e_{ij}} = \min_{dist(b,d)} \forall b \in graph\_pos(n_i), \forall d \in graph\_pos(n_j) \tag{2.2}$$

$$cost_{e_{ij}} = \omega_{\mathbf{extend}} dist_{e_{ij}} + \omega_{\mathbf{open}}[dist_{e_{ij}} \neq 0] \tag{2.3}$$

$$overlap_{e_{ij}} = (mem\_pos(n_i) + mem\_length(n_i)) - mem\_pos(n_j) \tag{2.4}$$

$$\mathcal{W}_{e_{ij}} = cost_{e_{ij}} - overlap_{e_{ij}} \omega_{\mathbf{match}} \tag{2.5}$$

If we are establishing G_{MEMCHAIN} based on MEMs derived from a read pair, and n_i and n_j are in different fragments in the read pair, then we derive $\mathcal{W}_{e_{ij}}$ based on a weight related to the probability of $dist_{e_{ij}}$ under an observed fragment length distribution:

$$\mathcal{W}^{\text{PAIRED}}_{e_{ij}} = P(dist_{e_{ij}} | obs\_frag\_len) \tag{2.6}$$

These node and edge weights relate to the alignment score that we would obtain by passing through a series of MEMs. We expect a positive score due to an exact match, so we apply a positive weight to each node as it represents a MEM. Transitions between MEMs may encode gaps or mismatches. We cannot estimate mismatch counts for the read from the set of MEMs obtained for our query, but we can use our position index and the function $dist(b, d)$ to estimate gap lengths. We take the score of a pseudoalignment $\mathcal{P} = n_i \dots n_j$ in the model to be:

$$\mathcal{S}_{\mathcal{P}} = \sum_{i=0}^{|\mathcal{P}|-2} \mathcal{W}_{\mathcal{P}[i]} + \mathcal{W}_{e_{\mathcal{P}[i]\mathcal{P}[i+1]}} \tag{2.7}$$

Given this definition, we expect the maximum sum walk $\mathcal{P}_{\text{MAX}} = n_i \dots n_j$ through the graph to be likely to yield the series of MEMs and graph positions involved in the maximum scoring alignment. This pseudoalignment is approximate due to the incompleteness of our score estimate and the fact that our MEM set is not guaranteed to capture the optimal alignment. To obtain a precise score we must then locally align the query against the graph.

To use G_{MEMCHAIN} to drive alignment, we need to be able to use it to derive a series of candidate alignment locations. We do so by applying a standard max-sum dynamic programming approach to G_{MEMCHAIN}. In this process, we derive a score for each node, $\mathcal{S}_{n_i}$, as the sum of own weight, the maximum score of any previous node, and the weight of the edge connecting the maximum scoring inbound node and the current node.

$$\mathcal{S}_{n_i} = \mathcal{W}_{n_i} + \max_{\forall e_{ji} \in E} \left(\mathcal{W}_{e_{ji}} + \mathcal{S}_{n_j} \right) \tag{2.8}$$

To allow traceback of the maximum scoring path, we record the maximum inbound node for each node.

$$\mathcal{T}_{n_i} = \operatorname*{argmax}_{n_j} \left(\mathcal{W}_{e_{ji}} + \mathcal{S}_{n_j} \right) \tag{2.9}$$

At the end of the scoring phase, we find the highest scoring node.

$$n_{max} = \max_{\forall n_i \in N} \mathcal{S}_{n_i} \tag{2.10}$$

Walking back through the series of recorded traceback pointers yields the maximum scoring path under the model. We define the series of nodes in the max-sum path by $n_{max-i-1} = \mathcal{T}_{n_{max-i}}$. The resulting path is expressed in reverse order relative to our traceback.

$$\mathcal{P}_{\text{MAX}} = n_{max-|\mathcal{P}_{\text{MAX}}|} \ldots n_{max-1}, n_{max} \tag{2.11}$$

To obtain a series of candidate alignments, after each pass of max-sum, we mask out the set of nodes and edges traversed by our last optimal path, run the scoring phase without these MEMs, and finally derive the next-best traceback.

Although the exact algorithm is different, in spirit our implementation is similar to that developed in [143], which extends collinear chaining to DAGs by running a similar model over a minimal set of paths covering the graph.

2.5.4 Unfolding

Every node has an implicit default orientation so that it is possible to determine edges that cause an inversion, i.e. those which connect between a forward and a reverse complement node orientation. When *unfolding* the graph, we use a breadth first search starting at every inverting edge in the graph to explore the reverse complemented portions of the graph that we can reach within length k from the inverting edge. We then copy this

subgraph, take its reverse complement, and replace the inverting edges connecting it to the forward strand of the graph with non-inverting ones. If k is as long as the longest walk in the graph, then unfolding will render the forward and reverse complement of the original graph on the forward strand of the unfolded graph.

2.5.5 DAGification

Variation graphs may have cycles. These are useful as compact representations of copy number variable regions, and arise naturally in the process of genome assembly. However, partial order alignment algorithms do not handle these structures, and so we convert cyclic graphs into k-path equivalent acyclic form in order to apply DAG-based alignment algorithms to them. To do so, we unroll cyclic structures by copying their internal nodes an appropriate number of times to allow a given query length to align through the unrolled version of the component. If our query is shorter than this limit, $k \geq |Q|$, then we are guaranteed to find the optimal alignment in the original graph by aligning against the DAGified one.

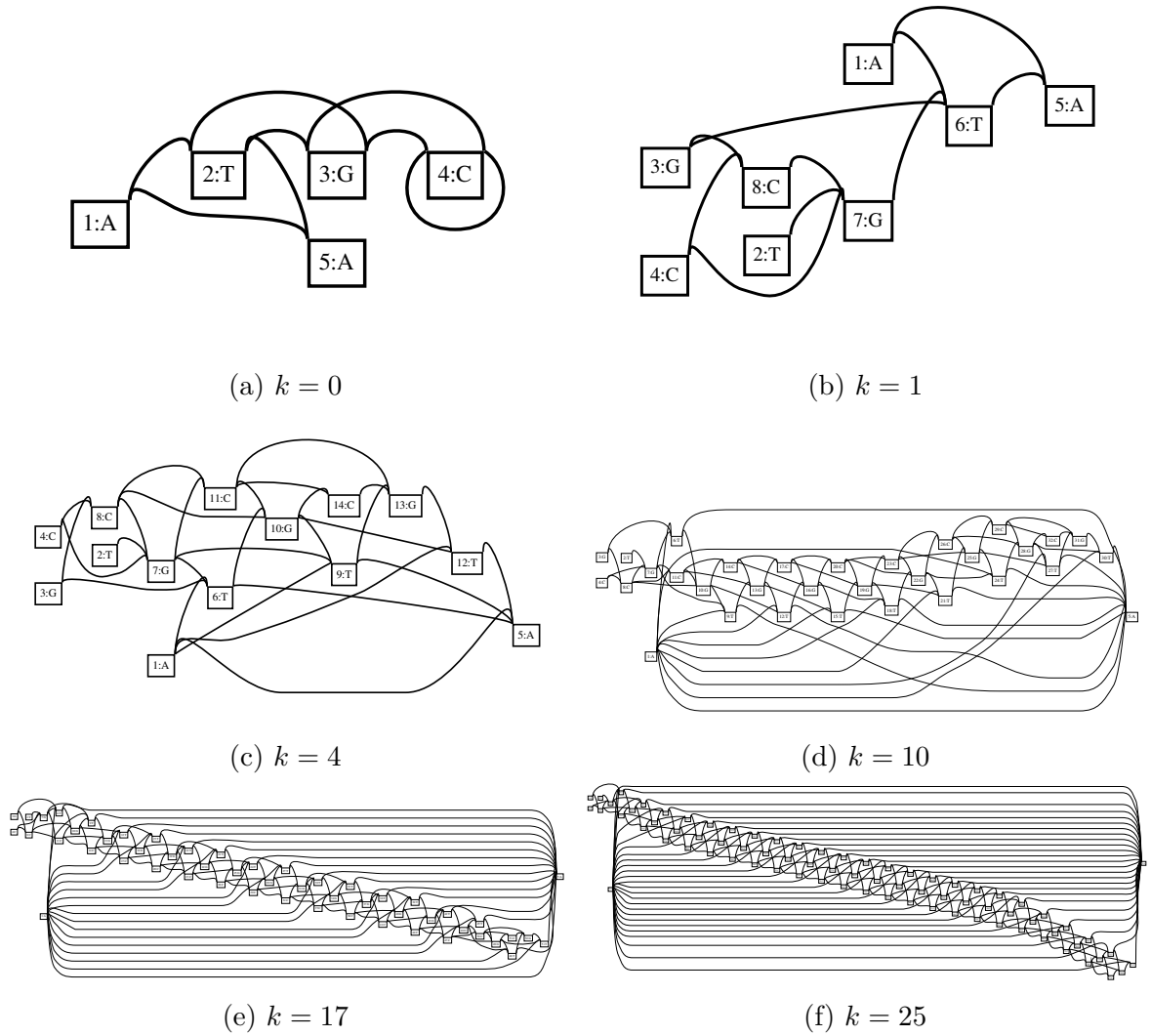
We first detect all strongly connected components by using a recursion-free implementation of Tarjan's strongly connected components algorithm [270]. Then, we copy each strongly connected component and its internal edges into a new graph. We greedily break edges in this graph that introduce cycles. Next we k-DAGify the component progressively copying the base component and, for each edge between nodes in the component, connecting from the source node in the previous copy to the target node in the current copy.

We use dynamic programming to track the minimum distance back through the graph to a root node outside the component at each step. When this reaches our target k, we stop unrolling, and add the expanded component back into the graph by reconnecting it with its original neighborhood. For each copy of a node in the DAGified component we copy all its inbound and outbound edges where the other end of the edge lies outside the strongly connected component. The resulting graph is acyclic and supports queries up to length k on the original graph using a translation that we maintain between the new graph and the source one.

2.5.6 POA and GSSW

Graph striped Smith-Waterman (GSSW)[14] generalizes an implementation [295] of Farrar's SIMD-accelerated striped Smith Waterman (SSW) algorithm [76] to enable string to

[14] https://github.com/vgteam/gssw



(a) $k = 0$ (b) $k = 1$

(c) $k = 4$ (d) $k = 10$

(e) $k = 17$ (f) $k = 25$

Fig. 2.14 DAGification of a small graph, as seen in 2.14a, with the k unrolling parameter given below each graph. In 2.14a we see a strongly connected component (SCC) of nodes 2, 3, and 4, which is copied in subsequent steps. Node ids in subsequent steps are not directly mapped to these original node ids. The DAGification algorithm proceeds by greedily breaking the cycles in this component, then copying the component and adding edges from the subsequent copy to the previous for each directed edge within the component until the minimum distance through the unrolled series of SCC copies is at least k. This minimum distance is tracked using a min-sum DP algorithm that is updated at each copy step by assigning a new minimum length for each node equal to the minimum of the previous minimum length among nodes in the previous SCC copy that it is connected to, plus their sequence label length. In panels 2.14d, 2.14e, and 2.14f, we see the unrolled SCC forming a braid in the middle of the rendered graphs, whose length increases with the increase in k. This algorithm is not optimal, as can be seen by the duplication of paths connecting in and out of the component. However, it is linear, and requires only $O(kc)$ time and space, with c representing a constant factor related to the size of the SCCs in the original graph.

graph alignment. Single-Input Multiple-Data (SIMD) instructions allow vectorized mathematical operations in a single machine instruction, and can be used to greatly speed up algorithms which can be implemented in terms of operations on vectors.

GSSW generalizes all aspects of SSW to operate over sequence directed acyclic graphs, including affine gap penalties, and retains its matrices for traceback[15]. This is simple to accomplish if the reference is a graph, as the striping of SIMD calculations in SSW across the reference is done by a single character at a time, and thus boundaries between nodes do not split the SIMD embedded variables. We can generalize SSW to GSSW by extending the recurrence relation that defines the scores in the DP matrices to consider all previous positions on all nodes that connect to the current one.

Given a query Q and a sequence graph $G = (N, E)$ with sequence length $L = \sum_i^{|N|} |seq(n_i)|$. We record the maximum scores of partial alignments between Q and G in the set of matrices $\mathcal{H} = \mathcal{H}_1 \dots \mathcal{H}_{|N|}$: each $\mathcal{H}_i$ is a $|seq(n_i)| \times |Q|$ matrix. $\mathcal{H}$ thus contains $|Q| \times L$ cells. When we have completed the scoring phase of alignment each $\mathcal{H}_i[x, y]$ will record the maximum score of an alignment between Q and G ending at $(n_i[x], Q[y])$[16]. To develop our scores, we use a scoring function $score(a, b)$, which in the case of DNA returns the value of a match (typically a positive integer) when $a = b \vee a = N \vee b = N$ and the value of mismatch when $a \neq b$ (typically a negative integer). We score a gap beginning with $\omega_{\textbf{open}}$ and a gap extension as $\omega_{\textbf{extend}}$. We record the score of a gap along G in matrices $\mathcal{E} = \mathcal{E}_1 \dots \mathcal{E}_{|N|}$ and a gap along Q in matrices $\mathcal{F} = \mathcal{F}_1 \dots \mathcal{F}_{|N|}$.

Gaps in $\hat{\mathcal{E}}$ extend across the graph, and so we need to consider all the inbound edges when we are at the beginning of a node:

$$\mathcal{E}_i[x, y] = \max \begin{cases} \mathcal{E}_i[x, y-1] - \omega_{\textbf{extend}} & \\ \mathcal{H}_i[x, y-1] - \omega_{\textbf{open}} & \\ \max_{\forall j : \exists e_{ji} \in E} \mathcal{E}_j[|n_j|, y-1] - \omega_{\textbf{extend}} & \text{if } x = 1 \\ \max_{\forall j : \exists e_{ji} \in E} \mathcal{H}_j[|n_j|, y-1] - \omega_{\textbf{open}} & \text{if } x = 1 \end{cases} \tag{2.12}$$

However, this is not the case for $\hat{\mathcal{F}}$, whose data dependencies flow vertically over the query Q:

$$\mathcal{F}_i[x, y] = \max \begin{cases} \mathcal{F}_i[x-1, y] - \omega_{\textbf{extend}} \\ \mathcal{H}_i[x-1, y] - \omega_{\textbf{open}} \end{cases} \tag{2.13}$$

[15]SSW discards these matrices for performance reasons, instead establishing the traceback later with local banded DP.

[16]Here I will use brackets [...] to identify the cells in 2-dimensional arrays.

The score in $\hat{\mathcal{H}}$ combines the affine gap calculations in $\hat{\mathcal{E}}$ and $\hat{\mathcal{F}}$. As with $\hat{\mathcal{E}}$, we here we also must consider the inbound nodes:

$$\mathcal{H}_i[x,y] = \max \begin{cases} 0 \\ \mathcal{E}_i[x,y] \\ \mathcal{F}_i[x,y] \\ \mathcal{H}_i[x-1,y-1] - score(Q[x], n_i[y]) \\ \max_{\forall j:e_{ji}\in E} \mathcal{H}_j[|n_j|,y-1] - score(Q[x], n_j[y]) \quad \text{if } x = 1 \end{cases} \tag{2.14}$$

The values of $\mathcal{H}_i$, $\mathcal{E}_i$, and $\mathcal{F}_i$ are 0 when $x = 0$ or $y = 0$ and node n_i has no inbound edges. Note that this is the initial condition provided by Gotoh to improve the algorithm of Smith and Waterman.

We fill the matrices using Farrar's SSW algorithm [76], based on Zhao's implementation [295]. By storing the full score matrices we can then trace back from the maximum score in $\hat{\mathcal{H}}$ to obtain the optimal alignments under our scoring parameters. The traceback can be represented as moves in the matrix, or equivalently as the alignment object model described in section 2.1.3.

2.5.7 Banded global alignment and multipath mapping

By modifying equation 2.14 so that it is no longer lower-bounded at 0 and changing the traceback so that it goes from beginning to end of query Q and graph G, we obtain a "global" alignment algorithm with the same properties as Needleman-Wunsch. To reduce computational costs, we can *band* the algorithm to limit the region of the DP tables which needs to be explored. This approach, as implemented in `vg` by Jordan Eizenga, forms the basis for multipath mapping, in which alignments are represented probabilistically as DAGs rather than linear series of node traversals and edits. In multipath mapping, regions between MEMs in a particular cluster are aligned using global alignment. The use of global alignment ensures that the alignment fully covers the gap between the MEMs. Multiple traceback allows for alternatives to be included, and each of these may be scored on the basis of both alignment score and haplotype matching score. His implementation is key to the development of haplotype aware mapping, which is the subject of a paper currently in preparation by myself and collaborators on the `vg` project. In the case of low-error reads, this limited exploration of the DP problem allows for fast derivation of the optimal alignments, and so the multipath mapper in `vg mpmap` achieves runtime

comparable to or exceeding `vg map`. Multipath mapping concepts also form the basis for alignment surjection, in which an alignment to the graph is projected into the linear reference.

2.5.8 X-drop DP

As our query length $|Q|$ increases, so does the practical complexity of deriving the alignment using POA/GSSW. We align longer queries against larger graphs, and so we effectively face a quadratic penalty with increasing alignment length, $|Q| \propto |L| \implies$ GSSW is $O(|Q|^2)$. The most direct solution to this is to use a banded alignment method like banded global alignment, as described in section 2.5.7. However, this method cannot exploit data parallel operations that allow dramatic speedups on modern processors.

In the course of our work on `vg`, Eizenga and I explored the application of Hajime Suzuki's adaptive banded global alignment (libgaba)[17] [268], which has been used in `minimap2` to greatly improve alignment speed with long single-molecule reads [157]. In this approach, an antidiagonal band of cells is computed at each step, of a predetermined width designed to fit into the word sizes of SIMD instructions. The band can move either "right" or "down" at each step, depending on where the highest score is found. A termination criterion is given, so that alignment stops when the maximum score falls a given amount. This is similar to the X-drop parameter used in BLAST to stop alignment extension. Although it improves performance, it can hurt sensitivity to indels.

Suzki had already implemented a version of alignment over graphs by transforming the graph into a tree through a dynamic unrolling process akin to that described in 2.5.5 and aligning to the tree using libgaba[18]. His implementation supports graph to graph alignment as described in section 2.1.4, but the exponential expansion of the alignment problem on trees is fundamentally limiting. Eizenga, Suzuki and I discussed methods to merge the bands together after traversal of unifications in the graph, but we could not establish a safe generic method to merge them. Furcated bands may only be merged directly if they map to the same query coordinate. This is unlikely to happen if the different paths in the graph that they have traversed have different lengths or if there are indels in the alignment.

During a biohackathon meeting in Kyoto, Suzuki presented an alternative banding model based on the "X-drop DP" algorithm from BLAST. In this model, the alignment is matrix broken into vertical non-striped windows that tile across the DP matrices over fixed subsequences in the query. To efficiently resolve the data dependencies between

[17] https://github.com/ocxtal/libgaba
[18] https://github.com/ocxtal/comb

successive steps, a SIMD shuffle operation is applied to the cell values stored in each window. Forward progression of each window stops when the highest score in the forefront cells drops X below the previously-observed maximum. This approach thus allows the band to spread as wide as needed to accommodate larger insertions, while being bounded by the X-drop parameter. The result is an approach that is more sensitive than the antidiagonal banded alignment in libgaba, but runs a factor of 2 slower for equivalent band sizes.

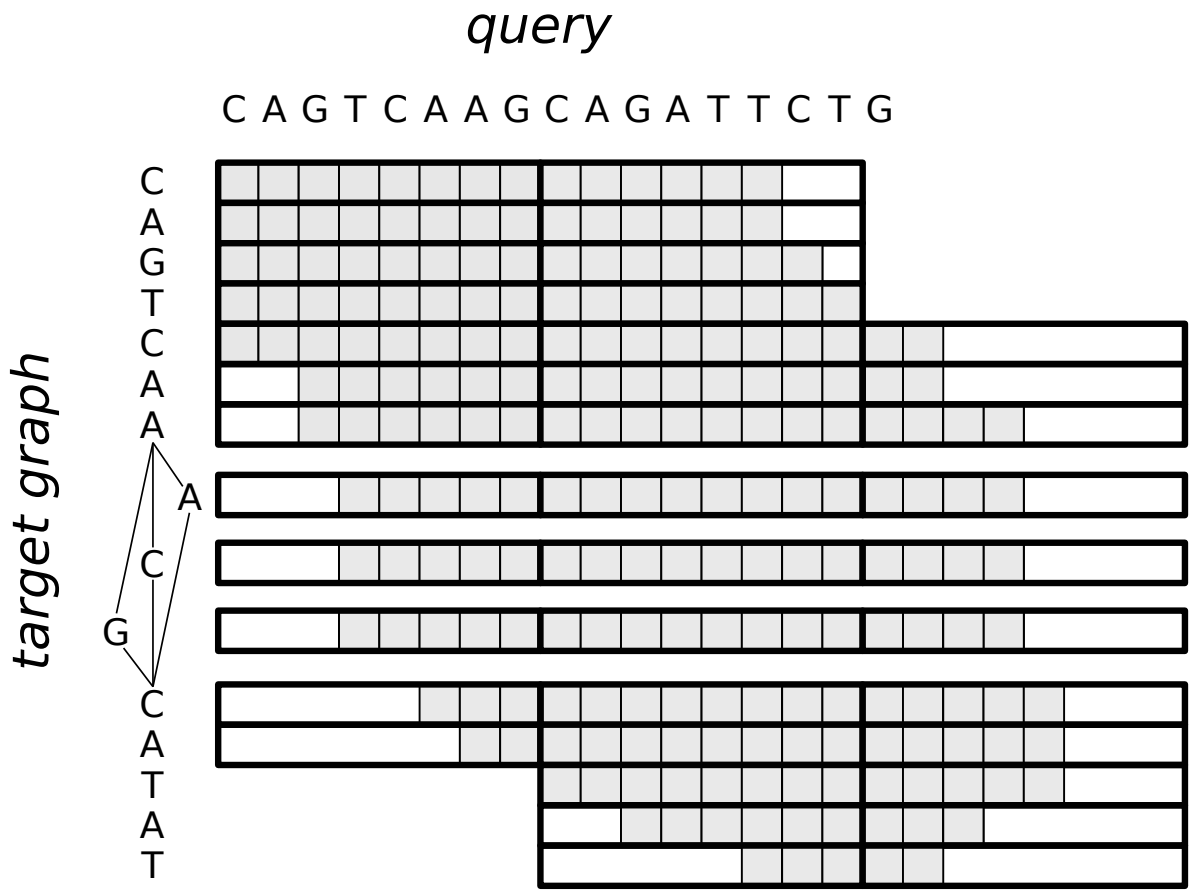


Fig. 2.15 Evaluation of the *dozeu* X-drop alignment algorithm for an example graph. Gray cells represent the part of the score matrix for which our X-drop parameter would allow evaluation if we were calculating the matrix one cell at a time. 8-cell wide black rectangles that contain the full set of gray cells represent the region of the score matrix for which we calculate scores when using a SIMD-based accelerated version of the algorithm. Traceback and per-cell scores are not shown. This figure is meant to illustrate the adaptive banding property of the X-drop algorithm and how that can be used in a SIMD-based acceleration of the algorithm. Adapted with permission from https://github.com/ocxtal/dozeu.

I have since worked with Suzuki to integrate his implementation of this algorithm *dozeu*[19] into `vg`. Due to difficulties in handling paired end rescue, the approach is not yet performing as well as GSSW for `vg map`. This remains a work in progress, but is a promising approach to enable the direct alignment of long sequences against the graph. It is orthogonal to the "chunked" alignment approach, and in principle, they can be applied together to build a SV-aware, chunked and banded alignment process. Future work in this direction may yield a new VG alignment algorithm, but this lies outside the scope of this thesis.

2.5.9 Chunked alignment

For long reads, where in the worst case the local dynamic programming can become prohibitively expensive, we break the reads into "bands" of a fixed width w (default 256 base pairs) with overlap between successive bands of $w/8$. Chunking the alignment process allows us to directly detect complex structural variation within our alignment, and provides a kind of split read alignment model for `vg`. We align these bands independently, trim the overlaps from the alignments, and build an alignment DAG model $G_{\text{AlignChain}} = (N, E)$ similar to that built for MEM chaining (as in 2.5.3). The only significant difference between these two models is that in $G_{\text{AlignChain}}$, we consider sub alignments as nodes in the model rather than MEMs.

In this model we put weights on transitions between alignments that relate to the estimated distance between the alignments in the graph versus their distance in the read, with the objective of making long co-linear chains be the highest-scoring walks through the chaining model. We take the max-sum path through the model to be the best alignment. Then, to obtain multiple alignments, we mask out this path, re-score, and take the next max-sum path to get the 2nd-, 3rd-, and ultimately Nth-best alignment.

After they have been extracted from the model, alignments are "patched" using local alignment of unaligned regions anchored in the graph near the end of previous mapped regions, so that sub-alignments which may have been misaligned due to repeats may be locally aligned correctly. This model allows `vg` to map noisy reads of arbitrary length, and is used as a core component in the long read progressive assembler `vg msga`.

Although the development of $G_{\text{AlignChain}}$ is very similar to G_{MemChain}, a number of important differences distinguish the two models. I will fully describe the chunked alignment chaining model here so that it may stand apart from the MEM chaining model.

[19] https://github.com/ocxtal/dozeu

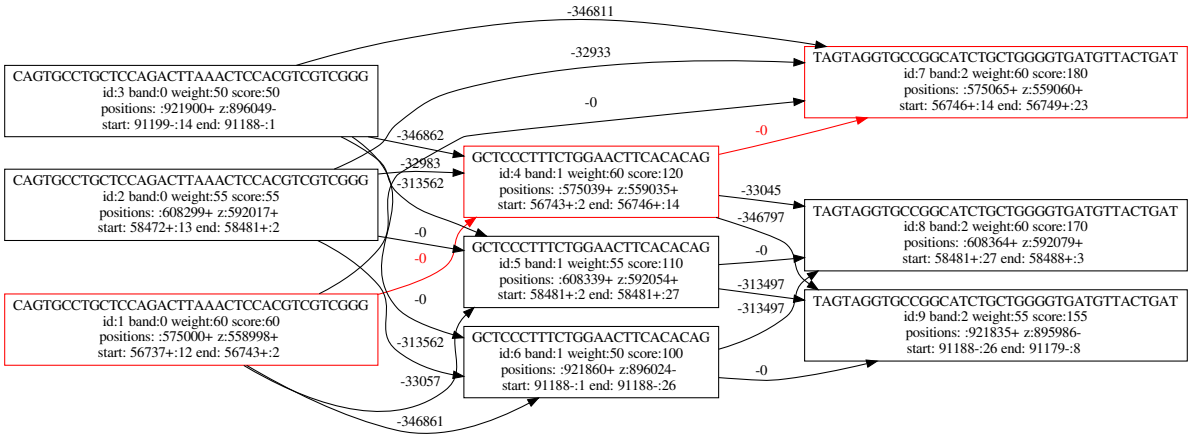


Fig. 2.16 The first evaluation of the alignment chain model for a set of alignment bands derived from a simulated query mapped against a test 1Mbp 1000GP graph included in the `vg` repository, using `vg map` parameters `-w 50 -O 25 -J 2` to simplify the model for exposition. Each node refers to a single mapping of an alignment band, its sequence, weight, its band index among bands taken from the original query, a node id in the graph, and the score derived from the first pass of the max-sum algorithm. Edges are labeled with their derived weight $\mathcal{W}_{e_{ij}}$, which can be seen to be 0 for edges connecting bands whose end and start positions directly connect. The maximum scoring chain is shown in red, and corresponds to the correct mapping location of the simulated query.

As each subalignment is part of the full alignment that we'll derive by finding the max-sum path through the model, we apply use the alignment score to set the initial weight for the node:

$$\mathcal{W}_{n_i} = alignment\_score(n_i) \tag{2.15}$$

Each alignment node n_i has a set of graph start and end positions $graph\_start\_pos(n_i) = \{b_1 \dots b_m\}$, $graph\_end\_pos(n_i) = \{b_1 \dots b_m\}$. We use these to provide weights for the edges in the alignment chain model. Edges e_{ij} in $G_{\text{AlignChain}}$ represent possible transitions between sub-alignments, and are weighted ($\mathcal{W}_{e_{ij}}$) by the minimum estimated distance between the given graph positions for each alignment end, times the gap open and extension costs. The alignment bands overlap during the alignment of each sub-alignment, but at the point when we establish the model, we have trimmed the overlaps between successive sub-alignments and re-calculated the alignment score, so we do not need to consider them here. Unlike the MEM chaining model, which is driven entirely by approximate distances, when we obtain an estimated distance less than the length of the alignment chunk, we walk the graph using a depth first search in order to obtain a precise minimum value for the distance.

$$dist_{e_{ij}} = \min_{dist(b,d)} \forall b \in graph\_end\_pos(n_i), \forall d \in graph\_start\_pos(n_j) \tag{2.16}$$

The chunked alignment model allows us to align through inversions, which we score using a basic heuristic that they are twice as costly as a gap of the same length as the inverted sequence. The inversion distance estimate walks forward from the positions in n_j and seeks n_i. In the case of a non-inversion, the resulting estimate will be ∞, as n_i will never be reached by walking forward from n_j.

$$dist\_inv_{e_{ij}} = dist_{e_{ji}} \tag{2.17}$$

$$\mathcal{W}_{e_{ij}} = \max \begin{cases} \omega_{\mathbf{extend}} dist_{e_{ij}} + \omega_{\mathbf{open}}[dist_{e_{ij}} \neq 0] \\ 2 \times \left(\omega_{\mathbf{extend}} dist\_inv_{e_{ij}} + \omega_{\mathbf{open}}[dist\_inv_{e_{ij}} \neq 0]\right) \end{cases} \tag{2.18}$$

These node and edge weights relate to the final alignment score that we would expect should we concatenate a particular series of sub-alignments. We cannot precisely estimate the final score of the read without the final patching step, but we can use our position index and the function $dist(b, d)$ to estimate gap lengths. We take the score of a concatenated alignment $\mathcal{P} = n_i \ldots n_j$ in the model to be:

$$\mathcal{S}_{\mathcal{P}} = \sum_{i=0}^{|\mathcal{P}|-2} \mathcal{W}_{\mathcal{P}[i]} + \mathcal{W}_{e_{\mathcal{P}[i]\mathcal{P}[i+1]}} \tag{2.19}$$

Given this definition, we expect the maximum sum walk $\mathcal{P}_{\text{MAX}} = n_i \ldots n_j$ through the graph to be likely to yield the series of sub-alignments and graph positions involved in the maximum scoring alignment we would obtain should we have aligned the entire query in one step. This concatenated alignment is approximate due to the incompleteness of our score estimate and the fact that our alignment set is not guaranteed to capture the optimal alignment due to the arbitrariness of the banding pattern that we have applied. Incompleteness due to our partitioning of the alignment problem means that to obtain a precise score we must locally realign unaligned portions of the concatenated alignment. This alignment patching process resolves problems that can occur due to induced soft clips, fully unaligned bands, or structural variations such as inversions or small CNVs that frustrate the complete alignment of a particular band.

As with the MEM chaining model, we derive partial alignments from $G_{\text{AlignmentChain}}$ by applying a standard max-sum dynamic programming algorithm to derive the maximum possible final alignment score at each node, and then, by working from the maximum scoring node, derive the maximum scoring path through the graph. The score for each node $\mathcal{S}_{n_i}$, combines the sum of own weight, the maximum score of any previous node it is connected to, and the weight of the edge connecting the maximum scoring inbound node and the current node:

$$\mathcal{S}_{n_i} = \mathcal{W}_{n_i} + \max_{\forall e_{ji} \in E} \left(\mathcal{W}_{e_{ji}} + \mathcal{S}_{n_j} \right) \tag{2.20}$$

To allow traceback of the maximum scoring path, we record the maximum inbound node for each node.

$$\mathcal{T}_{n_i} = \operatorname*{argmax}_{n_j} \left(\mathcal{W}_{e_{ji}} + \mathcal{S}_{n_j} \right) \tag{2.21}$$

At the end of the scoring phase, we find the highest scoring node.

$$n_{max} = \max_{\forall n_i \in N} \mathcal{S}_{n_i} \tag{2.22}$$

Walking back through the series of recorded traceback pointers yields the maximum scoring concatenated alignment under the alignment scoring and transition weight model we've applied. We define the series of nodes in the max-sum path by $n_{max-i-1} = \mathcal{T}_{n_{max-i}}$. The resulting path is expressed in reverse order relative to our traceback.

$$\mathcal{P}_{\text{MAX}} = n_{max-|\mathcal{P}_{\text{MAX}}|} \dots n_{max-1}, n_{max} \tag{2.23}$$

2.5.10 Alignment surjection

Alignments to graphs that include linear reference sequences as paths can be transformed into alignments against those paths. Alignment *surjection* projects alignments to the full graph *onto* the subgraph defined by a given set of paths in the graph. This transformation is used to project graph based alignments onto the linear reference, and is of great utility in the application of `vg` in resequencing based analyses, where it supports a lossy translation between `vg`'s native GAM format and BAM.

In surjection, portions of the alignment that already map to the path specific graph subset are left unchanged. However, regions of the original alignment to parts of the graph that are not in the selected path subset are mapped onto the nearest suitable node, with additional edits to specify the differences between the alignment and the

graph subset defined by the target paths. To produce a meaningful alignment, surjection requires that the alignment path matches the reference path for some portion of its length. Otherwise, the resulting alignment would be empty, or unaligned.

The name "surjection" is meant to be illustrative, and is not mathematically precise in the sense that it only applies to the alignments in terms of the node space of their paths. When we consider only the set of nodes that are traversed by a given alignment and its surjection, it is given that the function is surjective, as the domain of the function is the full graph, while the codomain is the subset defined by a given set of paths, with each node corresponding to one or more nodes in the original graph. (For instance, a mapping to a non-reference allele would be projected onto the nearest neighboring node in a reference path.) This surjective property would be violated in the case of a graph that only consisted of nodes in the paths that we are projecting our alignments into, as in this case the projection would bijective or the identity.

However, it is not clear that projecting the alignments into a graph subset defined by a given set of paths is surjective when we consider the full alignment data model. Information about the original alignment sequence is fully retained in the node mappings of the surjected alignment path, and this can be seen as violating the surjective property of the transformation.

The simplest surjection technique extracts the reference path region matching an alignment and realigns the read against it. Doing so without global alignment will often result in soft clipping, such as where non-reference alleles in the graph have allowed full length alignment. This can be resolved to some extent by applying global alignment of the alignment query sequence against the reference. But a more rigorous approach rebuilds the alignment in parts. For each piece that is not aligned to the reference, we extract the intervening reference sequence and align only the subset of the query that is no longer matching to this region. A kind of anchored semi-global alignment may be used on the ends of the reads, where the opposite reference-matching end is not defined. The resulting alignment may easily be expressed in the BAM format and thus be used by standard downstream variant calling and analysis methods.

2.5.11 Base quality adjusted alignment

Base qualities are typically reported on the Phred scale so that the probability of error for a given quality Q is $\epsilon = 10^{-Q/10}$. Assuming no bias in which bases are mistaken for each other, this defines a posterior distribution over bases b for a base call x.

$$P(b|x,\epsilon) = \begin{cases} 1-\epsilon & b = x \\ \frac{1}{3}\epsilon & b \neq x \end{cases} \tag{2.24}$$

We use this distribution to derive an adjusted score function. Normally, the match score for two bases is defined as the logarithm of the likelihood ratio between seeing two bases x and y aligned and seeing them occur at random according to their background frequencies.

$$s_{x,y} = \log\left(\frac{p_{x,y}}{q_y q_x}\right) \tag{2.25}$$

Next we marginalize over bases from the posterior distribution to obtain a quality adjusted match score.

$$\tilde{s}_{x,y}(\epsilon) = \log\left(\frac{(1-\epsilon)p_{x,y} + \frac{\epsilon}{3}\sum_{b\neq x} p_{b,y}}{q_y\left((1-\epsilon)q_x + \frac{\epsilon}{3}\sum_{b\neq x} q_b\right)}\right) \tag{2.26}$$

`vg` works backwards from integer scoring functions to the probabilistic alignment parameters in this equation. After doing so, the match scores are given by

$$\tilde{s}_{x,y}(\epsilon) = \frac{1}{\lambda}\log\left(\frac{(1-\epsilon)q_x q_y e^{\lambda s_{x,y}} + \frac{\epsilon}{3}\sum_{b\neq x} q_b q_y e^{\lambda s_{b,y}}}{q_y\left((1-\epsilon)q_x + \frac{\epsilon}{3}\sum_{b\neq x} q_b\right)}\right). \tag{2.27}$$

Here, λ is a scale factor that can be computed from the scoring parameters, and the background frequencies q_x are estimated by their frequency in the reference graph. Since base quality scores are already discretized, the adjusted scores can be precomputed and cached for all reasonable values of ϵ.

2.5.12 Mapping qualities

The algorithm for mapping qualities in `vg` is also motivated by a probabilistic interpretation of alignment scores. The score of an alignment A of two sequences X and Y is the sum of scores given in equation 2.25. This makes it a logarithm of a joint likelihood ratio across bases, where the bases are assumed independent (a more complete

justification including gap penalties involves a hidden Markov model, but it can be shown to approximate this formula). We denote this score $S(A|X,Y)$. Thus, assuming a uniform prior over alignments, we can use Bayes' Rule to motivate a formula for the Phred scaled quality of the optimal alignment, $\hat{A}$.

$$\begin{aligned} Q(\hat{A}|X,Y) &= -10\log_{10}(1 - P(\hat{A}|X,Y)) \\ &= -10\log_{10}\left(1 - \frac{P(X,Y|\hat{A})}{\sum_A P(X,Y|A)}\right) \\ &= -10\log_{10}\left(1 - \frac{e^{\lambda S(\hat{A}|X,Y)}}{\sum_A e^{\lambda S(A|X,Y)}}\right) \end{aligned} \tag{2.28}$$

Using the close approximation of the *LogSumExp* function by element-wise maximum, there is a fast approximation to this formula that does not involve transcendental functions.

$$Q(\hat{A}|X,Y) \approx \frac{10\lambda}{\log 10}\left(S(\hat{A}|X,Y) - \max_{A \neq \hat{A}} S(A|X,Y)\right) \tag{2.29}$$

In practice, we do not compare the optimal alignment to all possible alignments, but to the optimal alignments from other seeds. Thus, the mapping quality indicates the confidence that we have aligned the read to approximately the correct part of the graph rather than that the fine-grained alignment in that part of the graph is correct. Since this formula is based on alignment scores, it can incorporate base quality information through the base quality adjusted alignment scores.

2.6 Visualization

Visualization helps enormously to understand variation graphs and algorithms on them. While text-mode renderings are sufficient for evaluating results in resequencing against the linear reference, they are simply impractical when the reference is a graph. A set of dotplots can allow us to understand the relationship between many paths embedded in a graph. But this scales quadratically with the number of embedded paths and quickly becomes impossible to interpret. The alternative is to render graphs visually using a coherent set of visual motifs.

Here I describe several such techniques designed specifically for variation graphs. The simplest leverage standard utilities for graph drawing, and the most performant of these are hierarchical models that benefit from linear ordering which is often available in reference-ordered variation graphs. Force-directed layouts techniques developed for assembly graph interpretation allow us to interrogate larger-scale graphs. While it may be topologically complex, any graph is composed of sets which can be ordered linearly. By exploiting a linear sort of the graph I provide a linear-time layout algorithm that will scale to arbitrary data scales, allowing the visualization of both paths and read coverage against any graph.

2.6.1 Hierarchical layout

To develop a visualization method quickly, I relied most heavily on the four-phase hierarchical graph layout algorithm `dot` [89] that is part of the Graphviz package [88, 73]. This approach tries to generate a layout in which hierarchical structures in the graph are exposed, visual anomalies such as edge crossings and sharp edge bends are avoided, edges are short, and the layout is overall balanced or symmetric. It first uses a partial sort on the graph to derive a rank for each node. This aspect of the algorithm means it is best suited for DAGs. Then the unordered regions of the partially sorted graph are ordered to reduce edge crossings. Finally, the actual layout is derived and splines are drawn to show edges. The output of `dot` as well as other tools in Graphviz is a vector graphic, so the resulting renderings may be viewed in a number of ways.

To generate a visualization that captures the structure of the variation graph, I transform the graph into a visualization oriented structure in which the graph paths are rendered as nodes and edges. The layout is then driven entirely by the chosen algorithm in Graphviz, which is typically `dot`. This approach allows us to view rather large chunks of graphs, up to tens of kilobases, provided the graph is partially orderable.

The set of nodes may be rendered as boxes labeled by $id(n_i)$ and $seq(n_i)$. Edges have four types, and to indicate these we use the top and bottom of the node boxes. The top left corner of each node n_i receives incoming edges $e_{ji} \forall j : e_{ji} \in E$. While the bottom right corner of each node box represents edges arriving at $\overline{n_i}$ and thus $e_{j\bar{i}} \forall j : e_{j\bar{i}} \in E$. Similarly, the top right corner of each node box represents edges leaving n_i, and we add an edge for each $e_{ij} \forall j : e_{ij} \in E$. Finally, the bottom left corner represents the "end" of the reverse complement of the node $\overline{n_i}$, and, so we add edges for $e_{\bar{i}j} \forall j : e_{\bar{i}j} \in E$. As each edge implies its own reverse complement, we tend to replace edges $e_{\overline{ij}}$ with e_{ji}, and this is done both in normalization as in Graphviz based rendering.

Paths are not naturally supported in the Graphviz data model, and must be added as subgraphs with a different rendering style to identify them. In order to achieve a visually meaningful layout, these subgraphs must be also anchored appropriately into the graph. For each path, I hash the path name $name(p_i)$ into a set of colors and Unicode emoji, yielding $8 \times 766 = 6128$ possible color/symbol combinations. This generates a symbol for each path that is unlikely to collide with another given the typical application rendering a graph with tens of embedded paths. The hashing process also ensures the same rendering is returned as long as the same path names are given. For each mapping $m_i \dots m_{|p_j|} \in p_j$ I add invisible edges to the graph that link the mapping to the particular node it maps to as well as a visible edge in the path color from $m_{i-1} \rightarrow m_i$ when $i > 1$ and from $m_i \rightarrow m_{i+1}$ when $i < |p_j|$. A hint is given to `dot` to force the rank of each mapping to be the same as the node it maps to. Otherwise, the invisible edges encourage `dot` to render the path mappings close to the node they refer to. The resulting layout tends to look like a kind of multiple alignment matrix, as can be seen in figure 2.17.

2.6.2 Force directed models

Not all graphs yield easily to hierarchical layout algorithms. Graphviz also includes a force-directed layout algorithm `neato` that simulates the layout which would occur if connected nodes "pull" each other together and non-connected nodes "repel" each other apart. While the same input to `dot` may be used with `neato`, in practice the node labels become impossible to read and the edge types are confusing to infer, so a simplified rendering is produced without specific sequence labels on the nodes. This can still capture the overall structure of the graph as seen in figure 2.18.

While this rendering captures the path space of the graph even in arbitrary graphs, it cannot scale to graphs of significant size due to its approximately $O(|N|^3)$ scaling. The largest graphs I have visualized using this method contain tens of kilobases of sequence. `Bandage` [289] is an alternative method which is oriented towards visualizing assembly graphs. It reads GFA as input and provides an interactive rendering of the graph topology. This approach can render graphs of up to tens of megabases. Figure 2.19 shows the properties of this technique using the same region of H-3136.

2.6.3 Linear time visualization

Graph layout algorithms are computationally complex due to their need to iteratively relate all components of the graph to all others. In these layouts, we can observe large scale features about the topology and organization of the graph. These views are helpful

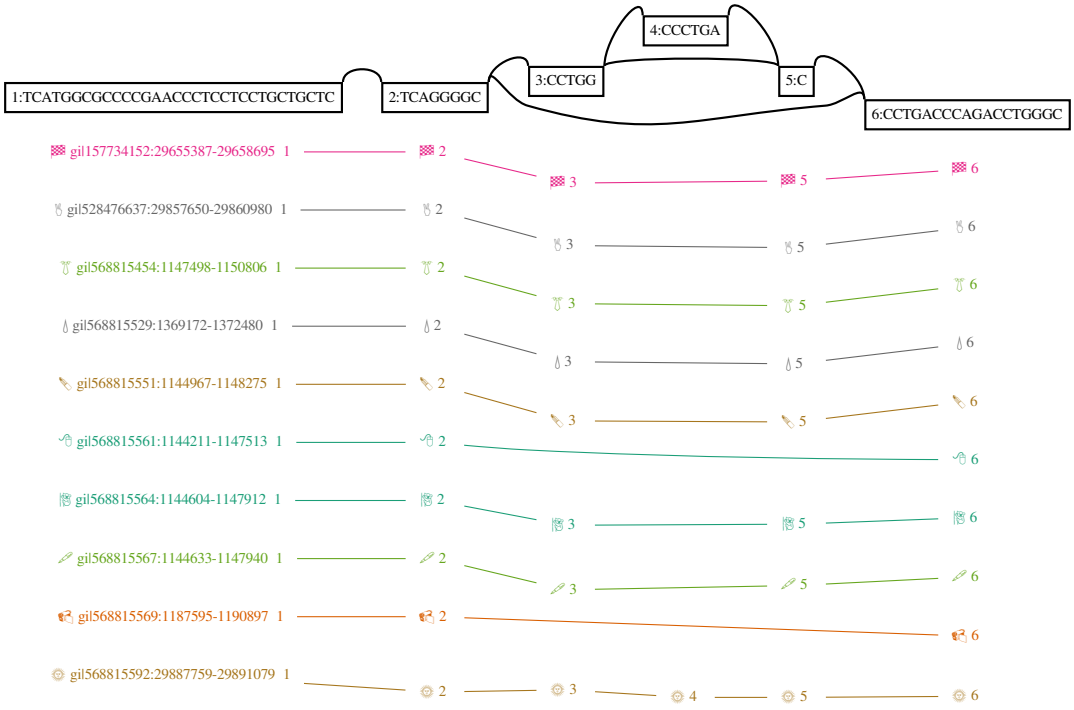


Fig. 2.17 The beginning of a variation graph built by progressive assembly of the GRCh38 haplotypes in HLA gene H-3136 visualized using `dot`. The graph topology of nodes and edges is represented above, with node ids and sequences labeled the rectangular nodes, and edges connecting the upper corners of nodes representing the edge topology relative to the forward strand of the graph. Below, paths in the graph are represented in a matrix-like format, with each mapping in the path represented by a node id and a colored emoji, and subsequent steps connected by a colored line. The emoji/color combination is determined by a hash function applied to the path name. A hidden edge connecting each path mapping step and the corresponding node in the graph topology is used to force the rendering to place them at the same horizontal position, increasingly legibility.

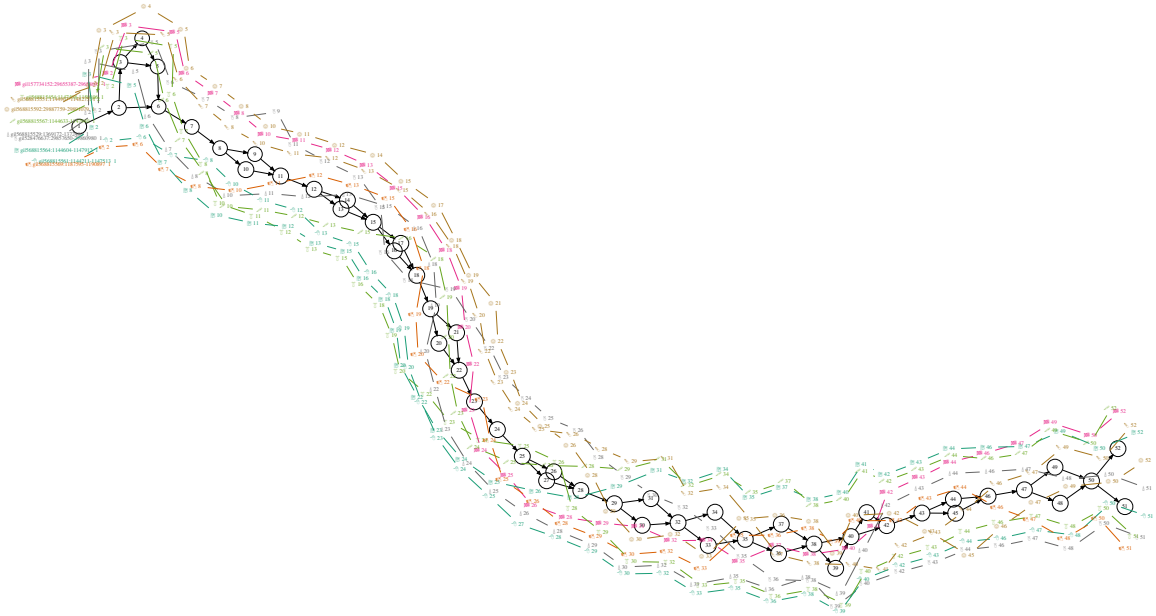


Fig. 2.18 A larger region of the same variation graph in figure 2.17 rendered using `neato`. Here, each path in the variation graph is represented by a colored path, with the node steps As in 2.17, paths in the graph are represented as colored paths connecting node ids and emoji. The same hash function is used to determine the color and emoji combination used to label each path. Each mapping in each path corresponds to one such node id / emoji label, and it is connected to the previous and subsequent steps by a line of the same color. A hidden edge connects each of these path steps to the node it corresponds to. This link causes the force directed layout algorithm to draw each path mapping node close to the node it refers to. Note that the graphviz data input here rendered by `neato` is the same as that given to `dot` in 2.17.

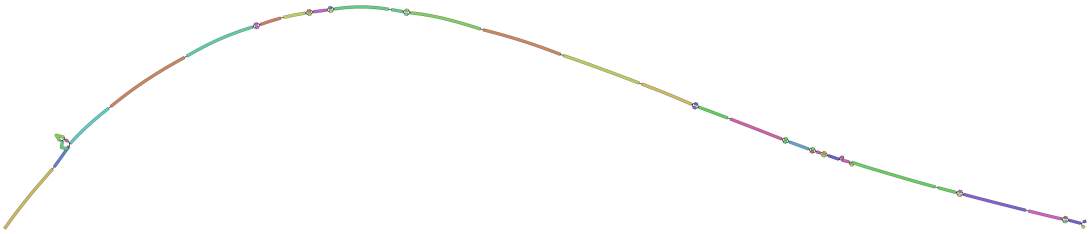


Fig. 2.19 The graph from figure 2.18 rendered using `Bandage`. This rendering algorithm does not include the graph's paths. Here, node colors have been randomly assigned, and serve to indicate where nodes start and end.

in many contexts, but the computational complexity of obtaining them prevents us from quickly visualizing larger graphs. Also, they do not scale efficiently to large path sets, and it is often difficult to understand alignments or other path-related on the graph using them.

To resolve these issues I developed a linear time rendering algorithm that projects a given VG, as indexed by XG, into a vector graphics format using the widely-available graphics library `libcairo`. This visualization algorithm uses the sequence basis vector $S_{\mathbf{iv}}$ as a coordinate system to position all elements of the graph. The graph topology itself is laid out in accordance with the node representation in $S_{\mathbf{iv}}$, flowing from left to right at the top of the rendering. Node lengths are shown using a black bar, with the sequence labels given below. The graph topology is rendered above the node set, and layout of these edges can be completed in linear time as they are rendered as simple splines connecting node ends. Paths, or other annotations such as coverage per read set, are displayed below as colored bars matching the subset of the node space that they cover. Where paths traverse a given node multiple times, an annotation is added to indicate the copy number. This technique is implemented as `vg viz`, and an example rendering is given in figure 2.20.

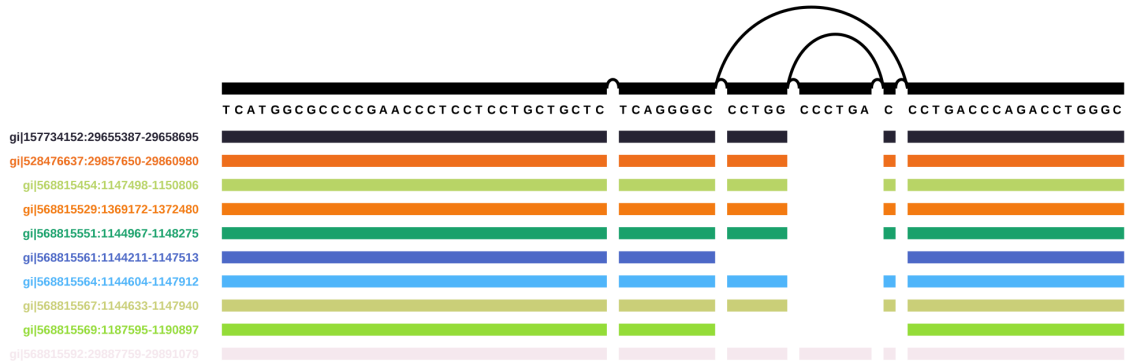


Fig. 2.20 The same variation graph in figure 2.17 rendered using `vg viz`. Above, we see the graph topology in terms of nodes, their lengths, and the edges that link their ends. Node sequence labels are shown below this topology, at the nodes they correspond to. Below, each path in the graph occupies a single position on the vertical axis. A hash function projects each path into a color, although the specific mapping is not the same as in figures 2.17 and 2.18. Each horizontal position corresponds to a unique graph position. Where a path touches this graph position, we applied a colored bar at the given vertical axis position corresponding to the path. This rendering thus represents paths as binary vectors over the sorted sequence space of the graph.

It is important to recognize that this approach is lossy. Path ordering is not clearly represented, as the paths are treated like masks over the sequence space of the graph. Furthermore, it can be difficult to interpret complex graphs as the topology of the graph is obscured in the simplistic rendering. `vg viz`'s linear layout treats the graph sequences as a basis space in which other paths or alignments may be interpreted. This simplistic view is central to many potential applications of `vg`, which I further discuss in section 2.8. The linear scaling of the algorithm should allow it to be applied to whole genomes, provided suitable front-end visualization software can be built, such as in a web interface.

2.7 Graph mutating algorithms

To use variation graphs as reference systems, we need to be able to modify them. Algorithms in graph theory are frequently based on graph transformations, and I have discussed some of them in the context of assembly and whole genome alignment. `vg` implements many algorithms that alter the graph, but a number are of importance to its development as a system for resequencing, and I detail them here.

2.7.1 Edit

As discussed in section 2.1.6, the extension of a variation graph to include new sequences and paths can be thought of as a transformation yielding a bijection between the new and old graphs $edit(G, A) \rightarrow (G', \Phi)$. The main issue which adds complexity to this process is that nodes represent more than a single character.

If nodes did represent a single character, then they would remain atomic through graph extension. It is simple to edit a "basepair" graph of this type: $G = (N, E, P)$, where $\forall_{n \in N} |seq(n)| = 1$. We walk through the alignments to it $A = a_1 \dots a_{|A|}$, adding novel bases represented in them ($seq(A) \notin G = N_A$) as new nodes. Then for each alignment a_i, we add edges connecting nodes in the order they would be traversed by the path of the alignment embedded in the graph $p_{a_i} \in G'$, yielding E_A. Finally, we add paths representing the embedded alignments P_A. The resulting graph unifies these additions $G' = (N \cup N_A, E \cup E_A, P \cup P_A)$.

To achieve lower representation costs in graphs with sparse variation, we usually compress the graph so that nodes cover multiple bases and thus we must consider the editing process in multiple phases. We first modify graph G so that every novel sequence will be added at the start or end of a node. We do this by breaking nodes into multiple derivative nodes when they overlap mappings that do not match the graph using

$break(G, A) \rightarrow (G', \Phi)$. For instance, if $seq(n_i) = \texttt{GATTACA}$ and we have a SNP $\texttt{T} \rightarrow \texttt{C}$ at offset 4, we would obtain $n_i \rightarrow n_i', n_i'', n_i''' : seq(n_i') = \texttt{GAT}$, $seq(n_i'') = \texttt{T}$, $seq(n_i'') = \texttt{ACA}$. We can then apply $translate(A, \Phi) \rightarrow A'$, and add edges to G' implied by the alignments as we would when editing a graph with single character nodes.

2.7.2 Pruning

The number of paths in a sequence graph grows exponentially with the number of variable sites. As I have discussed, this causes problems for alignment algorithms and graph sequence indexing. While we can use efficient disk-backed index construction algorithms like GCSA2 to mitigate the effects of this exponential scaling, only a handful of dense clusters of variation in the graph can increase the memory requirements of path enumeration beyond any reasonable level. To control this we restructure the graph to limit recombinations in the global sequence index.

We have explored two main techniques for the reduction of graph complexity. In the first, we prune regions of the graph which have high path complexity using a depth-first search (DFS). We can optionally add back known haplotypes, in order to mitigate the loss of information from the index. For VCF-based VGs, where we have haplotype panels, the performance of local alignment is unaffected by the topological complexity, so we only need to apply this pruning to the graph input to GCSA2 indexing. In assembly graphs, we find that nodes which represent repeats can sometimes have extremely high degree, which causes problems both for indexing and local alignment to the graph. There, we must remove these nodes in order to use the graph even for local alignment.

2.7.2.1 *k*-mer *m*-edge crossing complexity reduction

In k-mer complexity reduction, we enumerate the k-mers of the graph, removing edges when a given k-mer crossing them would cross $> m$ edges. We implement this filter, $prune(G, k, m) \rightarrow G_{\textbf{simple}}$, using the k-mer enumeration algorithm that generates a DBG from the variation graph, only that our walks through the graph are bounded at m edge crossings. For each offset in each node n_i and $\overline{n_i}$, we run a DFS forward until we have read k characters of the graph. During the pruning operation, instead of emitting the k-mers with their contexts, we stop the DFS when we have crossed m edges. We record the edge in $E_{\textbf{complex}}$. We thus derive the subgraph from the current graph by removing the complexity-inducing edges: $(N, E \setminus E_{\textbf{complex}}, P) \rightarrow G_{\textbf{simple}}$. It can be helpful to remove any small isolated components that result from this pruning, which can be done with a linear subgraph enumeration algorithm and the measurement of the sequence

length of each. A disjoint component $G_{\mathbf{sub}} \in G_{\mathbf{simple}} : \forall_{e_{ij} \in N_{\mathbf{sub}}} n_i \in N_{\mathbf{sub}} \wedge n_j \in N_{\mathbf{sub}}$ must have length $\sum_{\forall n \in N_{\mathbf{sub}}} |seq(n)| \geq \mathcal{J}$. By default, we use $k = 24$, $m = 3$, and $\mathcal{J} = 33$.

2.7.2.2 Filling gaps with haplotypes

Although removing complex regions will reduce the number of recombinant haplotypes represented by the graph, it is likely to also remove sections of known haplotypes. We can retain the complexity reduction without losing sequences in known haplotypes by replacing the pruned regions in $G_{\mathbf{simple}}$ with unfolded copies of each haplotype sequence. When we have a single reference path in $G_{\mathbf{ref}}$, we can accomplish this by overlaying $G_{\mathbf{simple}}$ and $G_{\mathbf{ref}}$. However, this will not achieve the desired result with even two overlapping paths, as where these differ they would reintroduce the re-combinatorial explosion that we hope to resolve with pruning. An alternative is to copy the haplotypes in the GBWT index that stretch from one border to the other of each removed region into the removed subgraphs in $G_{\mathbf{simple}}$. Doing so, we must preserve a mapping between the new nodes and the previous underlying ones in G. This allows matches to the haplotypes to be converted into matches in the base graph. The exact method by which this filling implemented is described in [259].

2.7.2.3 High degree filter

Because they separate dense variation into heterozygous bubbles, assembly graphs may feature greater "smoothness" than VCF-based graphs locally. But, in the context of repeats, they can contain nodes with exceptionally high degree. If these cluster together then they generate highly-connected regions that introduce degeneracy in the path space of the graph, and cause problems for k-mer enumeration and GCSA2 indexing. Furthermore, the local alignment methods in VG do not support efficient alignment through such dense regions. Preventing this is relatively simple, in that we remove nodes with more than $\mathcal{D}$ edges linking them to the graph, yielding $G_{\mathbf{prune}} : \forall_{n_i \in N_{\mathbf{prune}}} \mathcal{D} \geq |\{e_{*i} \cup e_{i*} \cup e_{\bar{i}*} \cup e_{*\bar{i}}\}|$.

It is not necessary that our local alignment suffers from high-degree nodes. The problem is that GSSW is provided an alignable graph that is an extracted subset of the full graph. If this subgraph is extracted using context expansion in the graph, then high-degree nodes will generate extremely large subgraphs. One solution would be to use the bit-parallel string to graph alignment approach in [227], as this achieves optimal bounds in the size of transformed graph to which we align. Alternatively, the graph exploration should be more directly linked to the alignment process. The X-drop aligner `dozeu` could be adapted to this approach, as the X-drop parameter would provide a natural limit to the graph exploration. Such approaches may allow us to tolerate a

larger $\mathcal{D}$, but it seems unlikely that they will allow alignment to be driven through the most-tangled areas of the graph without a large performance penalty relative to graphs with lower maximum degree.

2.7.3 Graph sorting

To achieve as much partial ordering as possible, we order and orient the nodes in the graph using a topological sort. The sort is guaranteed to be machine-independent given the initial graph's node and edge ordering. The algorithm is well-defined on non-DAG graphs, but in these cases the order is necessarily not a topological order. Our approach is a bidirected adaptation of Kahn's topological sort [126], which is extended to handle graph components with no heads or tails. This algorithm can be understood as a kind of seeded depth first search through the graph. Where the graph has nodes which are pure heads, it begins there. Otherwise, a set of seed nodes which are stably selected given a particular graph are used to begin the sort. The details of this procedure are provided in algorithm 3.

Sorting can provide a simple optimization during read alignment. If the reference graph has been sorted, then we can use the given order to generate node identifiers, embedding the rank of each node in its id. We can detect if a given subgraph is possibly non-acyclic if $\exists e_{ij} \in E : i > j$, and if so submit the graph to sorting, unfolding, and DAGification before applying local alignment.

Similarly, the sort allows us to project data in the context of the graph into a single dimension. Provided the graph is regionally partially ordered, this projection preserves local structures, which is a desirable property. This makes the sort applicable to visualization techniques as in figure 2.20.

2.7.4 Graph simplification

Assembly algorithms often employ a *bubble popping* phase, in which small bubbles, which are graph components connected to the rest of the graph through a single source and sink node, are replaced by linear components representing the most-likely path through the bubble given the read data. In `vg` we can carry out a similar operation based on the bubble decomposition of the graph. Unlike assembly graph bubble popping, we must retain information about the embedded paths and annotations in the variation graph. Simplification has a number of potential applications, for instance in reducing the complexity of visualizations of large variation graphs.

Algorithm 3 Pseudo-topological sort

$G = (N, E, P)$ ▷ A copy of our input graph which we will destructively modify
$L \leftarrow [\ldots]$ ▷ Stores the pseudo-topological order
$S \leftarrow \emptyset$ ▷ Set of nodes which have been oriented but not yet traversed
$V \leftarrow \{n_i\} \in N : \nexists e_{ji} \forall n_j \in N$ ▷ We start from the head nodes of the graph
if $V = \emptyset$ **then** ▷ If there are no head nodes, we use "seed" nodes
  $V \leftarrow$ Stably-selected seed nodes $\in N$
end if
while $V \neq \emptyset$ **do**
  $n \leftarrow n \in V$ ▷ Select a seed node
  $V \leftarrow V \setminus \{n\}$ ▷ Remove it from the input node set V
  $S \leftarrow S \cup \{n\}$ ▷ Store it in our working set S
  while $S \neq \emptyset$ **do**
    $n_i \leftarrow n_i \in S$ ▷ Remove an oriented node from S
    $S \leftarrow S \setminus \{n_i\}$
    $L \leftarrow [L[1] \ldots L[|L|], n_i]$ ▷ Append it to our output order L
    for $\forall n_j : e_{ij} \in E$ **do**
      $E \leftarrow E \setminus \{e_{ij}\}$ ▷ Remove the edge from our edge set
      if $\nexists e_{kj} \forall k \in N$ **then** ▷ n_j has no other edges to that side
        if $e_{i\bar{j}}$ **then**
          $n_j \leftarrow \overline{n_j}$ ▷ Orient n_j so the side the edge comes to is first
        end if
        $N \leftarrow N \setminus \{n_j\}$ ▷ Remove n_j from N
        $S \leftarrow S \cup \{n_j\}$ ▷ Insert n_j into S
      else ▷ This helps start at natural entry points to cycles
        $V \leftarrow V \cup \{n_j\}$ ▷ Record n_j as a place to start when S is empty
      end if
    end for
  end while
end while
return L ▷ Return our pseudo-topologically sorted order and orientation

2.8 Graphs as basis spaces for sequence data

If we can construct a graph which embeds all the sequences of all genomes which we are interested in, we resolve the separation between reference sequence and variation that is present in standard resequencing. This suggests that the intermediate steps in resequencing may be made redundant. If variation is already available during alignment then there is no need for a variant detection phase. However, if the graph does not include variation in our samples, then variant calling is required. In `vg` we have implemented several methods to do so. Similarly, we have implemented coverage summaries of read sets that may be used directly in downstream analyses.

2.8.1 Coverage maps

Numerous population genetic analyses are based on matrix representations of a collection of genomes. Such models can be used to infer population structure and phylogeny, as well as to associate phenotypes to genomic variants. If the variation graph used as a reference contains all sequences relevant to our analysis, then a matrix of per-base coverage of the graph by sample will provide highly representative information to downstream analyses. Exceptions include structural variation that does not result in coverage changes, such as balanced events like inversions. Also, some local patterns of variation between successive small variation will not be distinguishable. If we were to annotate edges with their coverage, this method would produce a result equivalent in information content to a Markov model. It is thus clear that any coverage based index will be lossy relative to the full read set. However, the lossiness reduces the information cost of storing and processing these coverage maps. As I described in section 2.4.6, in `vg` I developed an efficient method to accumulate coverage information of this kind across the graph.

2.8.2 Bubbles

In a sequence graph, a *bubble* is a pair of paths which start and end at the same nodes (s, t) but are otherwise disjoint in the graph [294]. Bubbles encompass our intuition about genetic variation in graphs. A homologous sequence corresponding to the common start and end nodes in the bubble flanks two or more alternative alleles in the middle. These structures were first considered in the context of finding small variation, and it was only in recent years that methods were developed to efficiently enumerate all bubbles of any size in DAGs [19].

Bubbles can nest and contain more complicated internal structures between the paths through them. The bubble may be generalized to the idea of a *superbubble*, which is a directed, acyclic component of a graph with a single head and tail node [207]. As for bubbles, efficient enumeration of superbubbles is possible in a DAG [24]. The optimal method relies on a recursive topological sort of the graph to structure the nested bubbles. Candidate node starts (s) and ends (t) are found following the definition of a superbubble. The set of candidates is then validated by range min queries (RMQ) to produce the set of superbubbles. As sort is linear $O(|N| + |E|)$, and the candidate enumeration and RMQ may be implemented on the sorted graph in $O(1)$ each, this yields a linear time algorithm for the enumeration of superbubbles.

The DAG requirement of this method is a significant limitation. In order to apply the superbubble enumeration to an arbitrary graph we must first DAGify it. In response, I worked with Benedict Paten and others to generalize the idea of a genetic site to support arbitrary bidirectional sequence graphs [213]. To formulate a generalization of superbubbles, we introduce the concept of a *snarl*, which is any graph component connected to the rest of the graph by two or fewer bordering nodes[20]. Snarls whose internal separated component is acyclic and does not contain any tips are *ultrabubbles*.

Paten observed that trees embedded in the Cactus graph transformation of a variation graph corresponded to the standard concept of superbubbles. Specifically, the cactus graph is transformed into a *cactus tree* in which each simple cycle in the cactus graph becomes a special kind of node. Various rootings of this tree may then be used to define a hierarchy of bubbles. The bidirectional nature of the variation graph mean that snarls can embed each other in a manner akin to how the twist used to generate an Möbius strip results in it having a single surface and border. In these cases the resulting cactus tree will support multiple alternative ultrabubble tree rootings, and so unlike the bubble and superbubble decompositions for DAGs the ultrabubble decomposition is not unique. We can enumerate a subset of these by identifying bridge edges (typically representing tips) in the "bridge forest", which is the result of a contraction of the cycles in the cactus graph into a set of top-level cycle representing nodes, and then use them to produce various rootings of the cactus tree.

Ultrabubbles and superbubbles provide a natural framework in which to reason about the hierarchy of variable genetic sites embedded in a graph. As such they provide the basis for generic models of genotype inference based on variation graphs which are capable

[20] In the paper the formulation is based on a bi-edged graph akin to the Enredo graph, but we note that a node-based formulation is equivalent and matches the other models in this section.

of genotyping any kind of genetic variation, including structural variation as well as nested variation, in the same model as SNPs and indels.

2.8.3 Variant calling and genotyping

Given a definition of variable genetic loci in the graph, we can build a genotyping system capable of generating genotype calls in the context of the graph. In `vg` we have explored two such methods. The first, originally implemented in `vg call`, generalizes the concepts first implemented in `samtools mpileup` to work on the graph. A set of alignments are first reduced to pointwise edits against the graph and per base coverage of the graph. This graph "pileup" is then processed by a genotyping algorithm that considers genetic sites using the ultrabubble model. A schematic overview of the method is shown in figure 2.21.

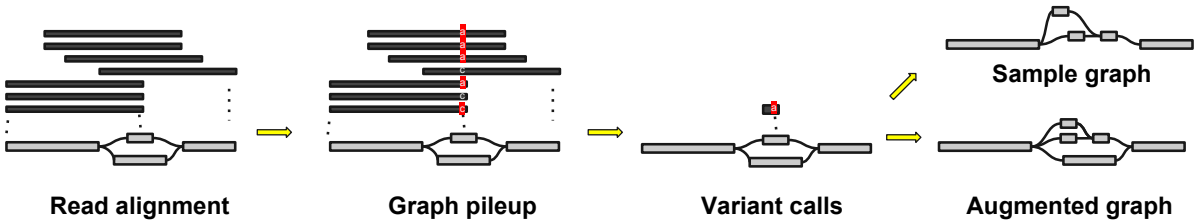


Fig. 2.21 Pileup variant calling with `vg call`

The second model, implemented in `vg genotype`, embeds the alignments in the variation graph using $edit(A, G_{\mathbf{base}}) \rightarrow (G_{\mathbf{aug}}, \Phi_{\mathbf{aug}\rightarrow\mathbf{base}})$, and then genotypes across the ultrabubbles of $G_{\mathbf{aug}}$ which are supported by reference paths in $G_{\mathbf{base}}$ or reads. As the resulting genotypes are represented as unordered sets of paths in $G_{\mathbf{aug}}$, they are projected back into the coordinate space of $G_{\mathbf{base}}$ using the translation $\Phi_{\mathbf{aug}\rightarrow\mathbf{base}}$. An overview of the process is shown in figure 2.22.

While principled, the full augmentation model in `vg genotype` is very expensive to compute. The pileup model has proven to be more efficient. Over time `vg call` has been adjusted to implement some features of the full graph augmentation model in `vg genotype` where the graph is augmented only with sequences supported by some number of alignments at a given quality threshold. Both methods employ a diploid specification of the genotyping model in freebayes [91] to develop their posterior estimates of variant quality. The generalization of SNP and indel calling to haplotype calling implemented in freebayes corresponds to the same allele model used in both `vg` variant calling methods. Alleles correspond to DNA sequences of arbitrary length, anchored at

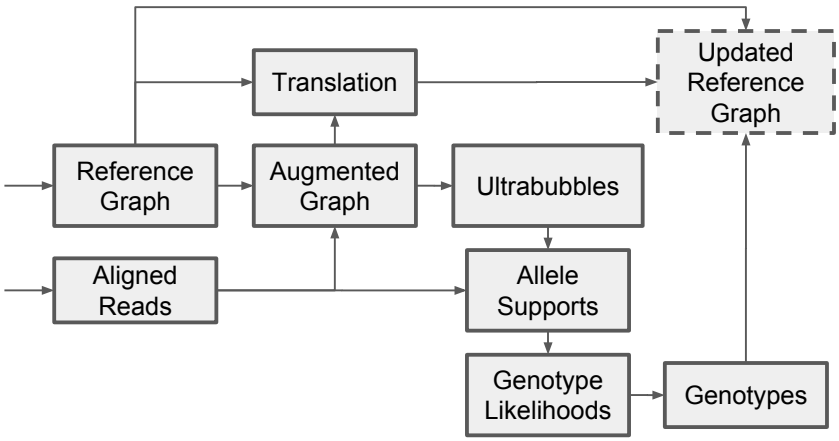


Fig. 2.22 Graph augmentation-based variant calling in `vg genotype`

.

the ends to the reference genome (in the case of freebayes) or to the rest of the graph (as in the ultrabubbles used in `vg`).

The complexity of running and evaluating genotyping in the graph has slowed development of these methods. Currently they are still outperformed by standard variant calling methods based on the linear reference, as indicated by our results in the PrecisionFDA variant calling challenge that I will describe in the next chapter.

Chapter 3

Applications

In the first two chapters I have provided an overview of the theoretical basis of my work and placed it within the history of related approaches. Here, I will demonstrate how the methods I develop can aid biological insight in a number of species domains. To do so, I will use methods described in chapter 2 as implemented in `vg`.

The small genome of *Saccharomyces cerevisiae* and ready availability of sources for pangenomic data models made it very useful to my development of `vg`. I begin by illustrating this for a variety of pangenome constructions and also a variety of read lengths.

However, much interest in bioinformatics is with larger genomes, specifically human. I use evaluations based on the human genome to validate the ability of `vg` to scale to large genomes. Through simulation and the analysis of real genomes, I show that the aligner I implement, `vg map`, yields the same quality of alignment as `bwa mem` against linear genomes. Although `vg map`'s runtime is between five to ten-fold slower than `bwa mem`, it provides improved, less biased alignment against variation graphs. I develop a variation graph for the reference-guided genome assemblies from the HGSVC project and demonstrate the strong effect of reference bias in ChIP-seq data.

One context where reference bias has very significant effects is in the analysis of ancient DNA (aDNA). Here short reads and high intrinsic error rates encourage a high rate of reference bias. I show that alignment against a pangenome graph ameliorates this issue.

`vg` can be applied to any kind of variation graph. To demonstrate the utility of this, I use de Bruijn assemblers to generate reference variation graphs from collections of raw sequencing reads in the absence of a prior reference. I recreate a classical pangenomic analysis of core and accessory pangenome by analyzing the coverage of alignments mapped to an assembly graph built from 10 *Escheria coli* strains. To illustrate the application of

vg to metagenomic data containing unknown source genomes, I show that vg enables the full length alignment of reads to a complex assembly graph built from an arctic viral metagenome, and similarly improves alignment to an assembly graph built from a human gut microbiome. Finally, I demonstrate that the data models and indexes in vg are capable of encoding splicing graphs, and that aligning to these splicing graphs allows the direct observation of the transcriptome.

3.1 Yeast

Saccharomyces cerevisiae, commonly known as baker's or brewer's yeast due to its gastronomic applications, has long been among the most important model organisms in biology, and its small genome attracted some of the first population scale whole genome surveys of variation to be undertaken using low-cost sequencing. The genome of *S. cerevisiae* was the first eukaryotic genome sequenced, in 1996 [94]. Resequencing studies followed that used *cerevisiae* as a model system to understand genome evolution. The Saccharomyces Genome Resequencing Project (SGRP) [169], which used low-coverage capillary Sanger sequencing to generate a population survey for *cerevisiae* can be seen as a precursor to the 1000GP, in its use of low-coverage sequencing and imputation to establish the panel[1]. A followup project, the SGRP2, used resequencing and whole genome assembly of high coverage, low cost Illumina sequencing to establish that the greater phenotypic diversity in *S. cerevisiae* relative to its wild relative *S. paradoxus* is likely due to structural variation (as measured by presence/absence and copy number) rather than SNP diversity [17]. Recently, whole genome *de novo* assembly with long single-molecule reads has further refined this conclusion by demonstrating that the structural diversity is non-uniformly distributed throughout the genomes of *S. cerevisiae*, concentrating in subtelomeric regions [292]. In this section, I use data from independent sequencing of the UK's National Center for Yeast Collections (NCYC) as well as long reads from [292] to demonstrate the capabilities of vg and compare the utility of various variation graph models built from these population surveys.

3.1.1 A SNP-based SGRP2 graph

The earliest rigorous testing of vg's alignment method was against a variation graph constructed from the SGRP2's released VCF for *S. cerevisiae*[2]. This early population resequencing project produced a VCF including only SNPs, yet using it already presented

[1] https://www.sanger.ac.uk/research/projects/genomeinformatics/assets/sgrp_manual.pdf
[2] http://www.moseslab.csb.utoronto.ca/sgrp/data/SGRP2-cerevisiae-freebayes-snps-Q30-GQ30.vcf.gz

problems typical even when working with larger scale genomes. The transposable elements in the genome generate rich patterns of repeats which make alignment difficult and require the development of mapping quality. Dense variation is also present in the results and this necessitates the application of pruning strategies to the graph to mask out high-complexity regions for indexing. Mistakes in the mapper could be readily observed and testing could easily be done on a laptop, whereas larger genomes require longer runtimes for indexing and larger servers in order to support the indexes during alignment.

The SGRP2 graph can be built and indexed in around 10 minutes on a commodity compute server, including the construction of the GBWT index and the generation of an order-256 GCSA2 index using a pruned and refilled version of the graph. It contains exactly the number of bases in the SGD_2010 reference plus the number of SNP alternate alleles in the SGRP2 VCF: 12163423 + 243629 = 12407052. The graph itself uses 23MB on disk, in contrast to that of the SGD_2010 reference, which takes only 7.6MB. Much of this difference is due to the larger number of entities required by to represent the variation-containing graph. The SGD_2010 graph is linear, with a gap for each chromosome, and contains 380,115 nodes and 380,097 edges after splitting into nodes of typical size 32bp, while the SGRP2 graph contains SNPs and is represented with 714,533 nodes and 969,690 edges. Note that by default, GCSA2 indexing works on nodes with a maximum length less than 1024, and `vg map` performs better if the maximum node size is limited further, with 32bp usually the standard maximum length in experiments I will present here. The resulting indexes also differ in size, with the SGRP2 graph's `xg` index requiring 71MB, while the SGD_2010 graph's only 38MB. The full GCSA index for the SGRP2 graph is substantially larger, at 220MB, in contrast to only 50MB for the linear reference, which reflects the greater complexity required to include all the recombination in the pruned and haplotype re-filled graph used for indexing.

To validate that the SGRP2 reference is a closer match to real read sets, I then mapped subsets of reads from *cerevisiae* samples that were in the NCYC collection but were not part of SGRP2. I aligned 100K read pairs from each of 12 samples ($N = 2.4$M total reads) to both the SGD_2010 reference graph and the SGRP2 pangenome graph. For each read, we can compare the alignment score and identity between the two graphs to evaluate the gain provided by using the pangenome as a reference. When we map the real reads from new strains not used to build the graph, 24.5% of the reads map better to the pangenome than to the linear reference (figure 3.1). A small fraction of reads (0.46%) map better to the linear than to the pangenome graph, which could result

from changes in paired alignment rescue, the effects of the pruning process, the slightly different minimum MEM size calculated for the two graphs, or errors in the SGRP2.

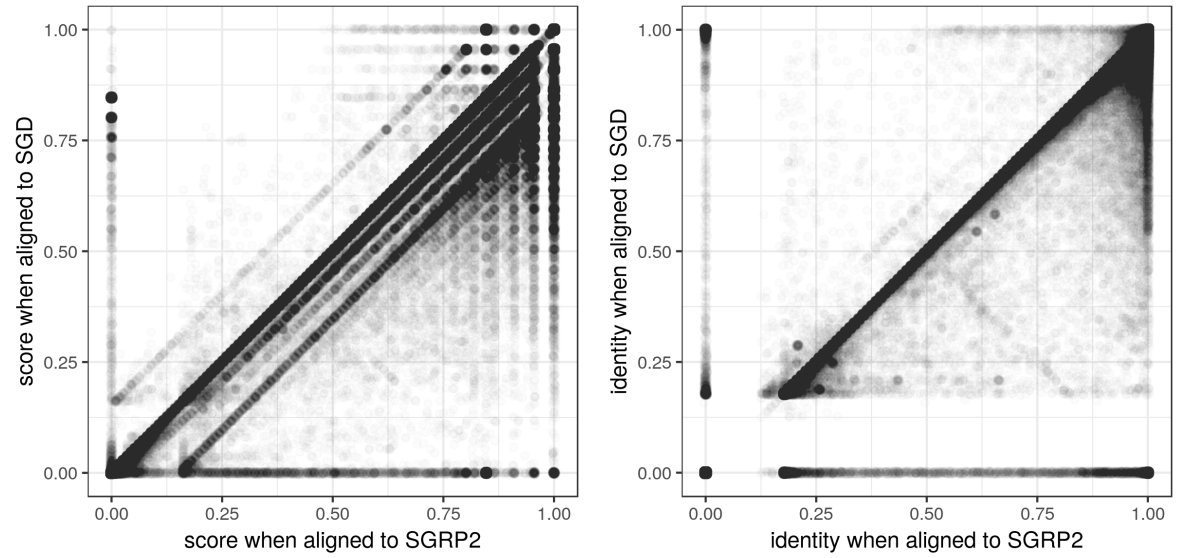
3.1.2 Cactus yeast variation graph

Variation graphs are generic objects capable of representing any kind of alignment between genomes or assembly of read data from them. To test the ability of `vg` to use graphs of complex topology, we constructed a variation graph from the whole genome alignment of *de novo* assemblies produced from PacBio sequencing of seven strains of *S. cerevisiae*. To build this graph, each chromosome of each assembly in [292] was aligned using LASTZ [107] according to a phylogenetic guide tree. The resulting alignment set was reduced to a variation graph using the Cactus reference-free whole genome multiple alignment method [212], which internally maintains a sequence graph equivalent to a VG. Its output was then converted to `vg` format using the hal2vg utility[3]. As illustrated in figure 3.2, this graph encodes a complex global topology that captures the structural variation between the species also reported in [292] as well as a local DAG-like topology which we expect when homologous sequences are represented compactly in a graph. This illustrates the ability of `vg` to represent paths corresponding to both collinear (inset) and structurally rearranged (main figure) regions of genomic variation.

A simulation study based on the SK1 strain provides some insight into the capabilities of `vg` and tradeoffs inherent in different graph designs. I compared four variation graphs: a linear reference graph from the standard S288c strain, a linear reference from the SK1 strain, a pangenome graph of all seven strains, and a "drop SK1" variation graph in which all sequence private to the strain SK1 was removed from the pangenome graph. The multiple genome graphs were based on that first constructed with Cactus, as described above, and then filtered down to the various subgraphs using path subsetting facilities in `vg mod`.

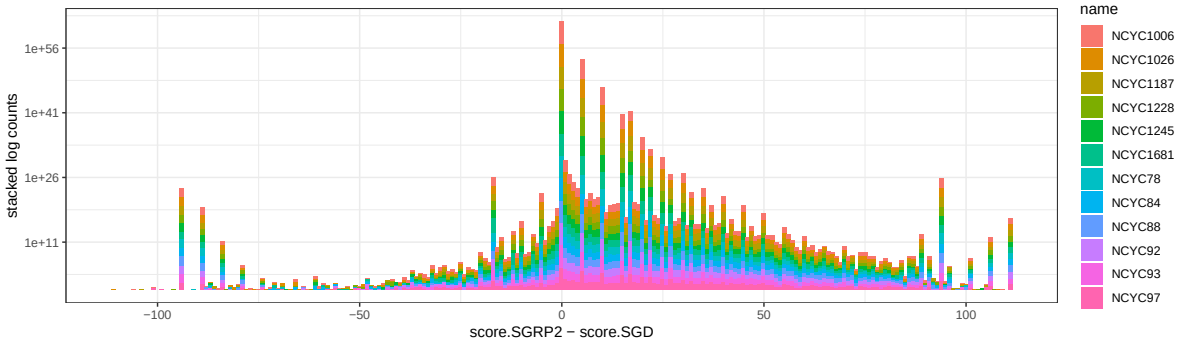
We simulated 100,000 150bp paired-end reads from the SK1 reference, modeling sequencing errors, and mapped them to the four references (ROC curves, Figure 3.3). Not surprisingly, the best performance was obtained by mapping to a linear reference of the SK1 strain from which the data were simulated, with substantially higher sensitivity and specificity compared to mapping to the standard linear reference from the strain S288c with either `vg` or `bwa mem`. Mapping to the variation graphs gave intermediate performance, with >1% more sensitivity and lower false-positive rates than mapping to the standard reference. There was surprisingly little difference between mapping to

[3] https://github.com/ComparativeGenomicsToolkit/hal2vg

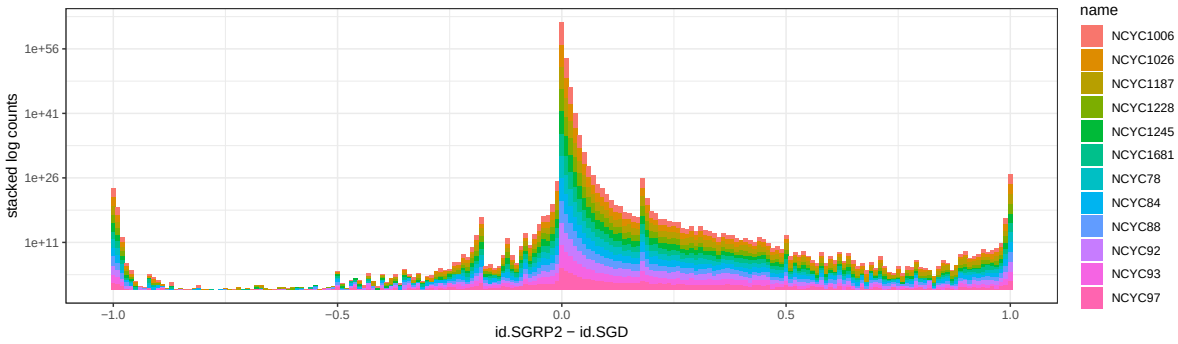


(a) Alignment score

(b) Alignment identity



(c) Difference in alignment score



(d) Difference in alignment identity

Fig. 3.1 Alignment of 100k read pairs from 12 NCYC *S. cerevisiae* strains against the reference genome (SGD) or the pangenome (SGRP2). In (3.1a) alignment scores are plotted for each read. The shift in density to the right relative to $y = x$ indicates improved alignments to the pangenome. In (3.1b) we observe the same pattern when using alignment identity rather than score. Subfigures 3.1c and 3.1d provide a stacked log-scaled histogram of the difference in score and identity between the two graphs.

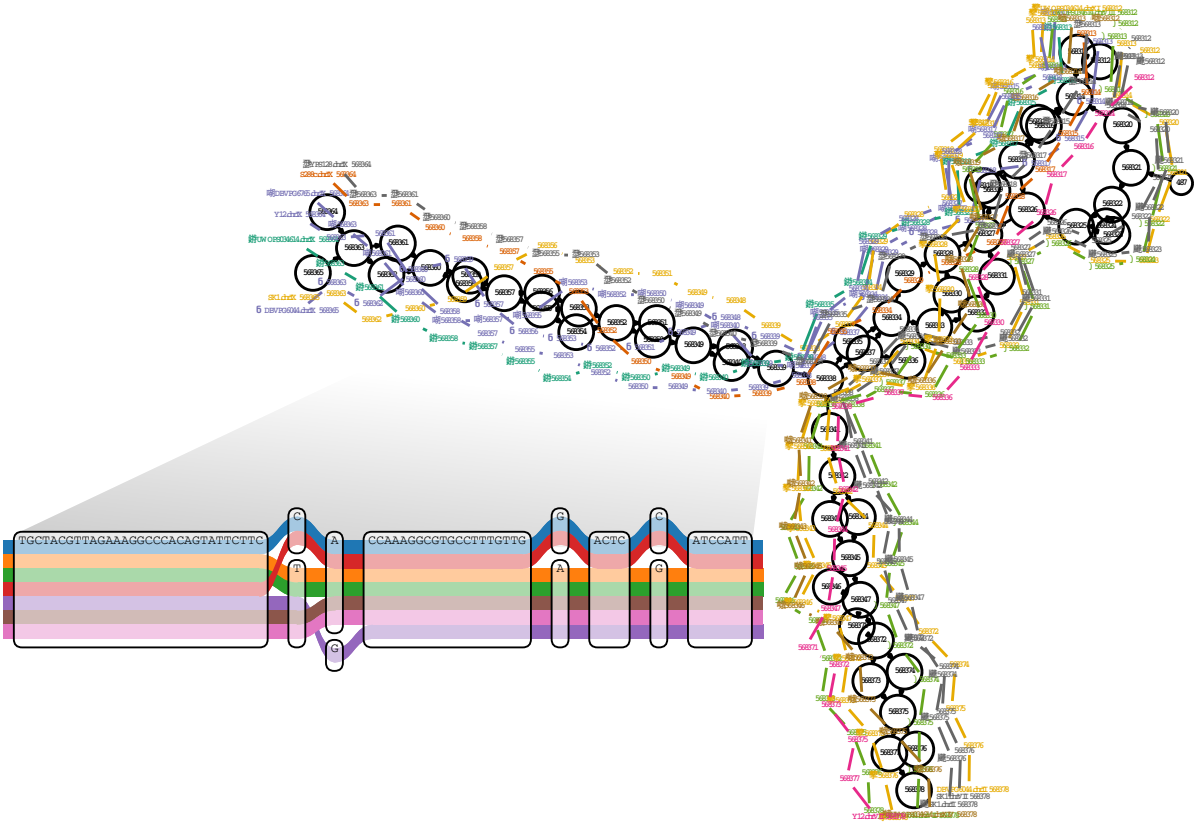


Fig. 3.2 A region of a yeast genome variation graph. This displays the start of the subtelomeric region on the left arm of chromosome 9 in a multiple alignment of the strains sequenced in [292] as assembled by Cactus [212]. The inset shows a subregion of the alignment at single-base level. The colored paths correspond to separate contiguous chromosomal segments of these strains. Reprinted from [92].

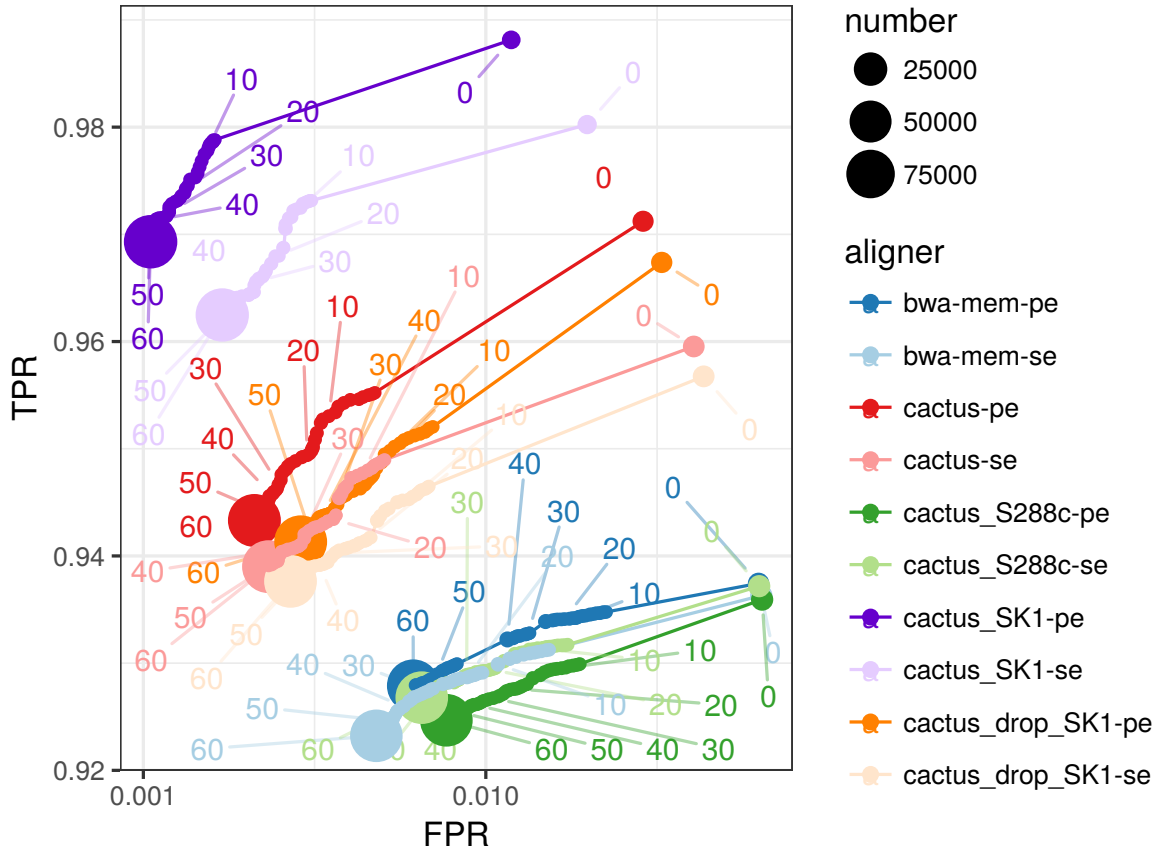


Fig. 3.3 Mapping short reads with vg to yeast genome references. ROC curves obtained by mapping 100,000 simulated SK1 yeast strain 150bp paired-end reads against a variety of references described in the text. Reprinted from [92].

graphs with and without the SK1 private variation, probably because much of what is novel in SK1 compared to the reference is also seen in other strains. Mapping to either graph had lower sensitivity compared to mapping just to the SK1 sequence, likely because of suppression of GCSA2 index k-mers in complex or duplicated regions, which our indexing strategy was not designed to address.

3.1.3 Constructing diverse *cerevisiae* variation graphs

With `vg`, our goal is to build a toolkit that allows the use of any genome graph as a reference. To validate this capability, I used data sources for *S. cerevisiae* to build seven variation graphs, whose dimensions are listed in table 3.1. In the next section (3.1.4) I present an evaluation of these graphs using long reads from the SK1 strain.

name	*size (MB)*	*length*	*nodes*	*edges*	*subgraphs*
SGD_2010	7.3	12163423	380115	380097	18
S288c	7.3	12249246	382797	382781	18
SGRP2	21	12407052	714533	969690	18
minia unitigs	15	14419206	1232804	1332994	46131
minia contigs	6.4	12233279	421125	425683	2961
Cactus	31	13243056	1059173	1304205	580
`vg msga`	42	13793955	1156295	1387903	2

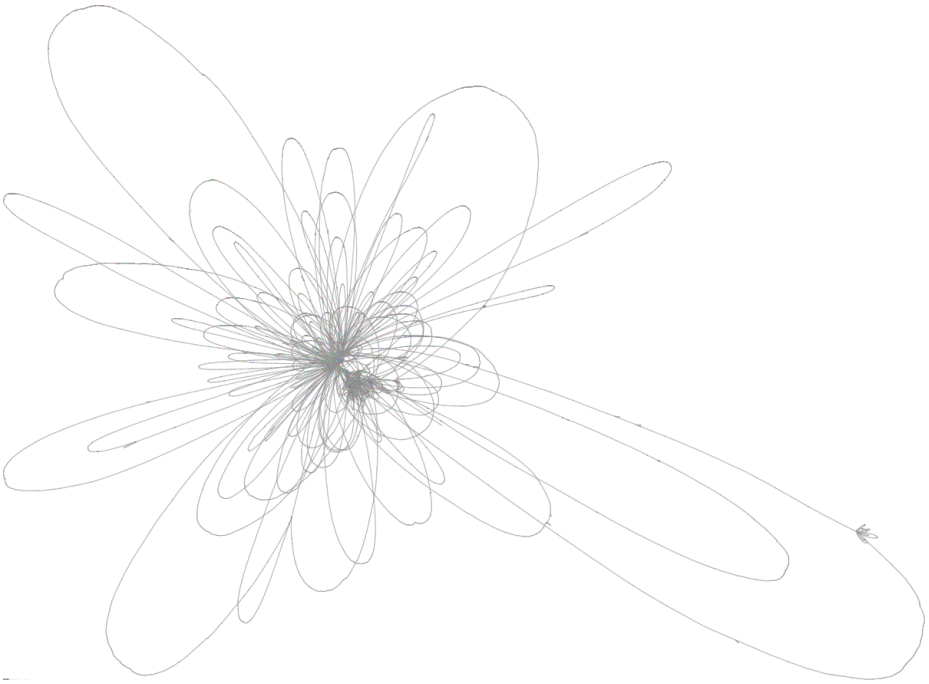
Table 3.1 A summary of seven different variation graphs constructed to represent variation in *S. cerevisiae*. As described in section 3.1.1, the SGD_2010 graph is built from the reference, while SGRP2 adds SNPs in the SGRP2 population survey. The Illumina data from [292] was used to build the minia unitigs and minia contigs graphs, while the whole genome, chromosome-resolved PacBio assemblies from the same work were used to build the Cactus and `vg msga` variation graphs. S288c is the *de novo* assembly of the reference strain produced in [292].

Three of the graphs are effectively linear or DAG-like, except for their mitochondria and plasmid chromosomes, which are included as circular components. SGD_2010 and S288c represent two assemblies of the reference genome, the former from the SGD genome sequencing project, and the latter is a *de novo* assembly from [292]. The difference in quality between the two approaches will be made apparent in the subsequent section. As described in section 3.1.1, the SGRP2 graph adds SNP variation from the population survey in [17] to build a pangenome reference. The SGD_2010 reference contains the mitochondria and 6kbp plasmid sequence, while the S288c assembly excludes them due to the sequencing protocol, where size selection in the library preparation stage removed

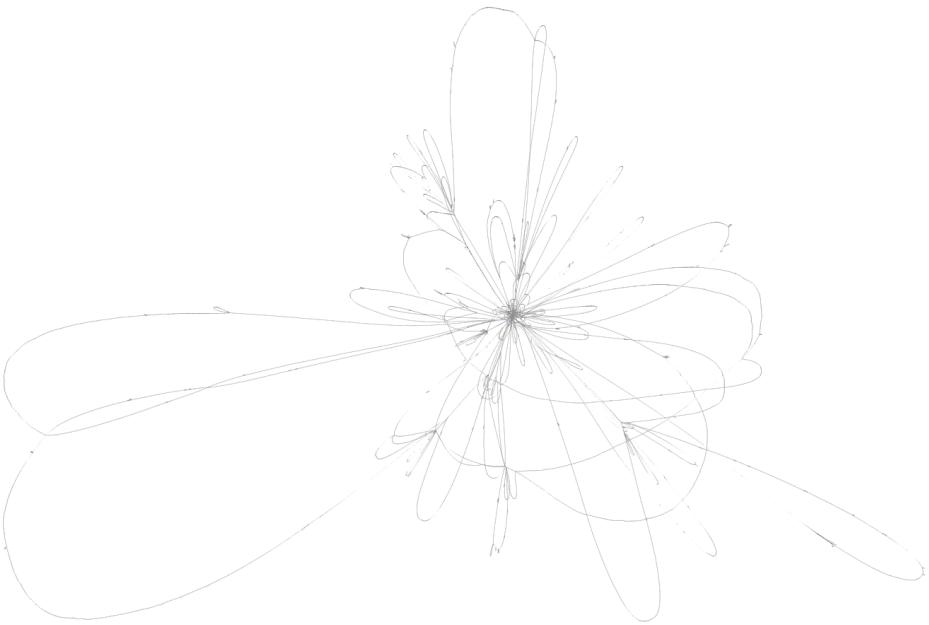
these short sequences, so to enable direct comparison in later tests these were added to the S288c graph.

The remaining four graphs are different forms of assembly graph. Using the Illumina data published in [292], I build two assemblies with minia3, using a k-mer size of 51 and a high abundance threshold (50) to limit the resulting graph complexity. In the first, I take the unitig graph that represents all non-branching paths in the compacted DBG as nodes. The second is the result of the minia contigification process that pops bubbles and cleans the graph to attempt to arrive at longer contigs. Both results are expressed as overlap graphs in GFA, and I can import them as variation graphs using bluntification and graph pseudotopological sorting. As shown in table 3.1, the unitig graph is considerably more complex in terms of node density than the contig graph. It also contains more sequence, presumably because some regions that are separated in the unitig graph are collapsed in the contig graph. The number of disjoint components in these graphs is very high (2961 for the contig graph and 46131 for the unitig graph), suggesting that pruning of the assembly graph has yielded a greatly fragmented result. The resulting graphs are difficult to align long reads to. I conclude that further tuning of the parameters used during *de novo* assembly from short reads will be required to use assemblies like this as reference graphs.

Finally, I used the whole genome *de novo* assemblies of long read data from [292] to build whole genome alignment graphs using Cactus (as described in section 3.1.2) and `vg msga`. The multiple sequence to graph alignment (MSGA) process implemented in `vg msga` is akin to the progressive POA method, but generalized to arbitrary graphs of any size. Rather than using a local alignment algorithm to expand the graph, the long read alignment algorithm described in section 2.5.9 allows the direct alignment of whole chromosomes to the graph. Where the Cactus variation graph uses a phylogenetic guide tree to structure its construction, `vg msga` simply aligns the chromosomes in order from longest to shortest to the growing graph. The long read alignment in `vg` is structured to enforce long range synteny, and the resulting graph is substantially different in structure than that of Cactus. I find that `vg msga` is less likely to collapse repeats than Cactus, at least in the configuration used for this assembly. We can see this in figure 3.4, where the Cactus variation graph (3.4a) shows two dense repeat structures in its core connected by loops of unique sequence, while the `vg msga` graph appears to have much longer loops, with collapsed repeats embedded in these loops. This observation is supported by the length statistics in table 3.1, with `vg msga` producing a graph that is 550,899bp longer than that of Cactus. At the same time, the node count of the `vg msga` graph is higher, which perhaps reflects a different local alignment result.



(a) Cactus variation graph



(b) `vg msga` variation graph

Fig. 3.4 Whole genome alignment graphs for *S. cerevisiae* visualized using Bandage.

3.1.4 Using long read mapping to evaluate *cerevisiae* graphs

In this section I evaluate the graphs I constructed in section 3.1.3 and simultaneously demonstrate the ability of `vg` to align long reads to graphs of any type. For each of the seven graphs I aligned a set of 43,337 Pacific Biosciences SK1 reads (mean length 4.7kbp) from [292] to the graph. We can then compare the alignment identity for each read across the various graphs. I do so using the same dot plot technique used to demonstrate alignment quality improvement using the Illumina data from the NCYC strains. In figure 3.5 I present a number of pairwise comparisons based on this read set.

I find that the SGD_2010 reference provides a better match for the SK1 PacBio reads than the S288c assembly (top left), which can be seen in a subset of reads that map nearly perfectly to the SGD_2010 graph but not to the S288c one. This may be due to the higher quality and curation of the SGD reference, which was initially based on BACs and capillary sequencing, but I have not determined the exact cause of this discrepancy. This same effect is clear in the comparison of S288c and the SGRP2 (top middle, figure 3.5), although there the SNPs in the SGRP2 graph tend to improve the overall match between the SK1 reads and the graph, which can be seen in a shift in density upwards from the diagonal. For other comparisons I focused on using the S288c reference, as it forms a part of the progressive alignments and the source data for the minia assemblies comes from the same paper.

The minia graphs appear to provide very low quality as a reference for the alignment of long reads (bottom left and middle, figure 3.5). The minia unitig graph is too fragmented for any practical use. In almost no case does it provide a better match for the long reads. However, while the minia contig graph is also outperformed by the S288c graph, for a notable subset of the reads it provides a perfect match, while the S288c graph fails to match them at all. This suggests that some contigs in the Illumina assembly match the SK1 strain, which is to be expected and demonstrates that in principle this kind of graph can represent multiple genomes.

Finally, the whole genome alignment graphs are notable in their similarity. Despite the fact that they were constructed using different algorithms, both provide a similar basis for alignment of the SK1 reads. It is notable that alignment time against the `vg msga` graph was the highest of the tested graphs, and significantly higher than that for the Cactus graph. This may relate to the un-collapsed state of the repeats in the graph. The alignment algorithm will attempt more alignments for each band where there is ambiguity, and the "patching" at the end of the alignment process will be more intensive.

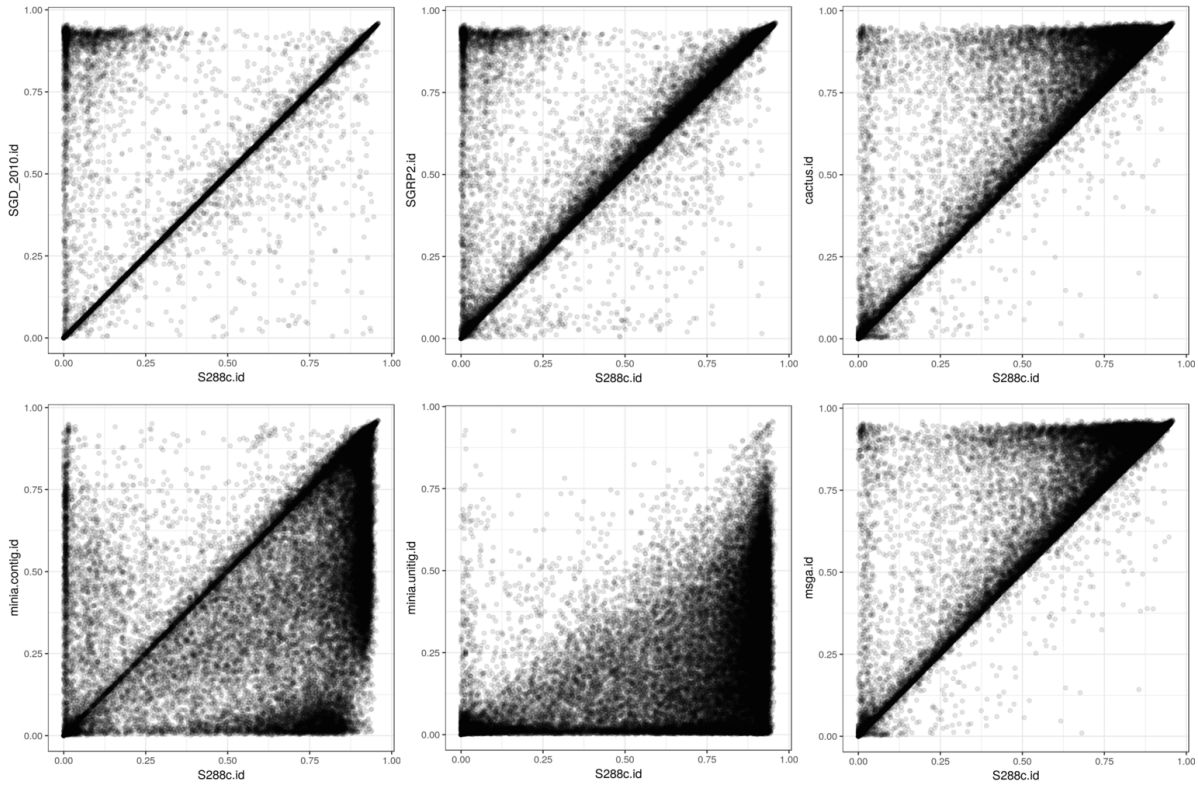


Fig. 3.5 Density plots of alignment identity when mapping 43,337 Pacific Biosciences long reads from the SK1 strain to different variation graphs. *Linear assemblies*: (Top left) the SGD_2010 reference vs. the S288c assembly from [292]. (Top middle) the S288c assembly vs. the SGRP2 SNP pangenome. *Assembly graphs from Illumina data*: (Bottom left) the minia contig graph vs. the S288c assembly. (Bottom middle) the minia unitig graph vs. the S288c assembly. *Whole genome alignment graphs*: (Top right) the Cactus variation graph vs. the S288c assembly. (Bottom right) the `vg msga` variation graph vs. the S288c assembly.

3.2 Human

For a species such as human, with only 0.1% nucleotide divergence on average between individual genome sequences, over 90% of 100bp reads will derive from sequence exactly matching the reference. Therefore, new mappers should perform at least as well for linear reference mapping as the current standard, which we take to be `bwa mem` with default parameters. We show that `vg` does this, and then that `vg` maps more informatively around divergent sites.

3.2.1 1000GP graph construction and indexing

The final phase of the 1000 Genomes Project (1000GP) produced a data set of ∼80 million variants in 2,504 humans [45]. We made a series of `vg` graphs containing all variants or those above minor allele frequency thresholds of 0.1%, 1%, or 10%, as well as a graph corresponding to the standard GRCh37 linear reference sequence without any variation. The full `vg` graph uses 3.92 GB when serialized to disk, and contains 3.181Gbp of sequence, which is exactly equivalent to the length of the input reference plus the length of the novel alleles in the VCF file. Complete file sizes including indices range from 25 GB to 63 GB, with details including build and mapping times given in table 3.2.

Reference set	*N vars*	*vg*		*index*		*search time*	
	(M)	*time*	*size*	*time*	*size*	*PE*	*SE*
GRCh37	0	1:09:54	1.76	23:30:41	25.11	33:34	28:33
1000GP AF0	84.8	3:42:01	3.92	51:05:07	63.28	45:10	39:46
1000GP AF0.001	30.2	2:00:08	2.58	31:45:12	38.10	39:33	32:53
1000GP AF0.01	14.3	1:35:02	2.17	27:18:53	30.94	33:13	27:09
1000GP AF0.1	6.8	1:23:04	1.97	26:06:38	27.79	32:35	28:43

Table 3.2 Numbers of variants, file sizes in gigabytes (GB) and build and search times in hours:minutes:seconds for various human `vg` graphs and associated indexes. Reference sets are the linear reference GRCh37, the full 1000 Genomes Project set 1000GP AF0, and subsets of 1000GP AF0 including only variants with allele frequency above thresholds 0.001 (0.1%), 0.01 (1%) and 0.1 (10%) respectively. The number of variants in millions for each of these data sets is shown. Search times are for 10 million 2x150bp read pairs simulated from NA24385. Reprinted from [92].

3.2.2 Simulations based on phased HG002

We next aligned ten million 150bp paired-end reads simulated with errors[4] from the parentally phased haplotypes of an Ashkenazim male NA24385, sequenced by the Genome in a Bottle (GIAB) Consortium [300] and not included in the 1000GP sample set, to each of these graphs as well as to the linear reference using `bwa mem`. Figure 3.6 shows the accuracy of these alignments compared with `bwa mem` for the full range of frequency thresholded graphs, in terms of receiver operating characteristic (ROC) curves.

Reads that come from parts of the sequence without differences from the reference (middle panels of Figure 3.6) mapped slightly better to the reference sequence (green) than to the 1000GP graph (red), which we attribute to a combination of the increase in options for alternative places to map reads provided by the variation graph, and the fact that we needed to prune some search index k-mers in the most complex regions of the graph. The best balance of performance appears at the threshold of 0.01. As expected, this difference increased as the allele frequency threshold was lowered and more variants were included in the graph.

For reads that were simulated from segments containing non-reference alleles (∼10% of reads), which are the reads relevant to variant calling, `vg` mapping to the 1000GP graph (red) gave better performance than either `vg` (green) or `bwa mem` (blue) mapping to the linear reference (right panels of Figure 3.6), because many variants present in NA24385 are already represented in the 1000GP graph. This is particularly clear for single-end mapping, since many paired-end reads are rescued by the mate read mapping. Overall, `vg` performed at least as well as `bwa mem`, even on reference-derived reads, and substantially better on reads containing non-reference variants.

3.2.3 Aligning and analyzing a real genome

We also mapped a real human genome read set with ∼50× coverage of Illumina 150bp paired-end reads from the NA24385 sample to the 1000GP graph. `vg` produced mappings for 98.7% of the reads, 88.7% with reported mapping quality score 30 on the Phred scale, and 76.8% with perfect, full-length sequence identity to the reported path on the graph. For comparison, we also used `vg` to map these reads to the linear reference. Similar proportions of reads mapped (98.7%) and with reported quality score 30 (88.8%), but considerably fewer with perfect identity (67.6%). Markedly different mappings were found for 1.0% of reads (0.9% mapping to widely separated positions on the two graphs, and 0.1% mapping to one graph but not the other). The reads mapping to widely

[4]SNP errors are introduced at a rate of 0.01 per base and indels at a rate of 0.002 per base.

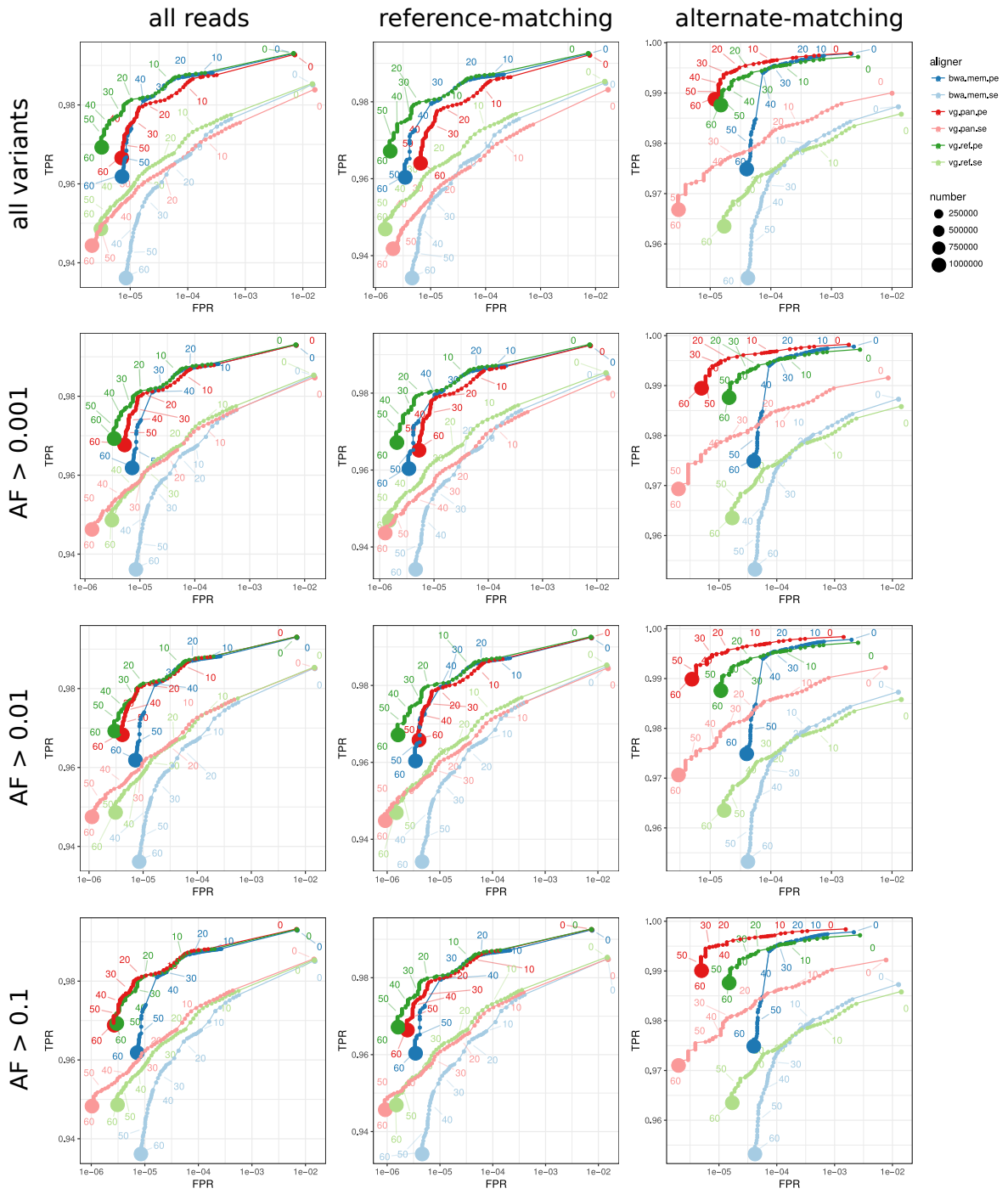


Fig. 3.6 ROC curves parameterized by mapping quality for 10M read pairs simulated from NA24385 as mapped by `bwa mem`, `vg` to various 1000GP pangenome references, and `vg` with a linear reference, using single end (se) or paired end (pe) mapping. Allele frequency thresholds are given to the left of each row. Within each panel, the left subpanel is based on all reads, middle on reads simulated from segments with no genetic variants from the linear reference, and the right on reads simulated from segments containing variants. Reprinted from [92].

separated positions were strongly enriched for repetitive DNA. For example, the linear reference mappings for 27.5% of these read pairs overlapped various types of satellite DNA identified by RepeatMasker, compared to 3.0% of all read pairs.

To illustrate the consequences of mapping to a reference graph rather than a linear reference, we stratified the sites independently called as heterozygous in NA24385 by deletion or insertion length (0 for single-nucleotide variants) and by whether the site was present in 1000GP, and measured the fraction of reads mapped to the alternate allele for each category. The results show that mapping with `vg` to the population graph when the variant was present in 1000GP (95.4% of sites) gave nearly balanced coverage of alternate and reference alleles independent of variant size, whereas mapping to the linear reference either with `vg` or `bwa mem` led to a progressively increasing bias with increasing deletion and (especially) insertion length (Figure 3.7), so that for insertions around 30bp, a majority of insertions containing reads were missing (there were over twice as many reference reads as alternate reads).

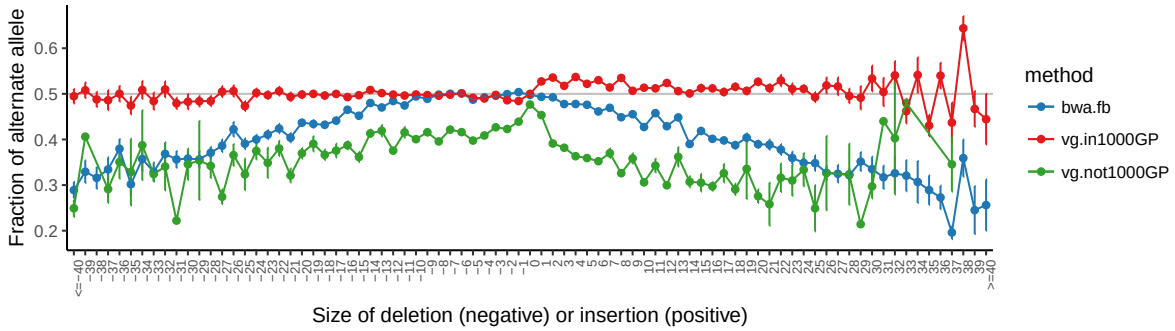


Fig. 3.7 The mean alternate allele fraction at heterozygous variants previously called in HG002/NA24385 as a function of deletion or insertion size (SNPs at 0). Error bars are ± 1 s.e.m. Reprinted from [92].

3.2.4 Whole genome variant calling experiments

During the development phase of `vg`, we explored its application to whole genome alignment and variant calling in the PrecisionFDA Truth Challenge, in which the team which developed the Genome in a Bottle truth set developed and held out a new sample. Methods were first tested against publicly available truth sets on NA12878 in the "consistency" challenge, in which the `vg` development team received a star for "heroic effort" in completing the first whole-genome graph based alignment and variant calling analysis. The results for this first iteration were very poor, with F-scores for indels and

SNPs around 95%. The computational costs were high, with the run consuming around $1000 in resources on Amazon's Elastic Compute cloud (AWS EC2).

In the final round of the challenge, we obtained results as described in table 3.3. We find that the `vg` pipeline had similar performance to the *de novo* assembly pipeline fermikit for SNPs. Methods that are not explicitly based on the GATK indel calling method perform notably worse on indels, including egarrison-hhga[5], which used Platypus, fermikit, and freebayes to generate candidate variants and implemented a genotyper using a machine learning method, and mlin-fermikit, which was a direct application of fermikit's standard pipeline to the data for HG002. However, `vg call`'s indel calling results were very poor, and likely caused by bugs in the variant caller and aligner at this stage rather than conceptual problems with graph based variant calling.

Submission	*SNPs*			*indels*		
	F-score	*recall*	*precision*	*F-score*	*recall*	*precision*
anovak-vg	98.4545	98.3357	98.5736	70.4960	69.7491	71.2591
astatham-gatk	99.5934	99.2091	99.9807	99.3424	99.2404	99.4446
bgallagher-sentieon	99.9296	99.9673	99.8919	99.2678	99.2143	99.3213
dgrover-gatk	99.9456	99.9631	99.9282	99.4009	99.3458	99.4561
egarrison-hhga	99.8985	99.8365	99.9607	97.4253	97.1646	97.6874
hfeng-pmm3	99.9548	99.9339	99.9756	99.3628	99.0161	99.7120
mlin-fermikit	98.8629	98.2311	99.5029	95.5997	94.8918	96.3183
rpoplin-dv42	99.9587	99.9447	99.9728	98.9802	98.7882	99.1728

Table 3.3 A selected subset of PrecisionFDA Truth Challenge results showing the best-performing methods as well as a number of other notable submissions.

We also explored integration of `vg` with the recently published GraphTyper [71] method, which calls genotypes by remapping reads to a local, partially ordered variation graph built from a VCF file, relying on initial global assignment to a region of the genome by mapping with bwa to a linear reference. Therefore, although GraphTyper also scales to the whole human genome because it is essentially a local method, its functionality is complementary to that of `vg`, which maps to a global variation graph and does not directly call genotypes. In experiments where we used `vg` rather than bwa as the primary mapper for GraphTyper, true positives increased marginally (0.02% for single-nucleotide polymorphisms (SNPs) and 0.06% for indels) while false positives increased for SNPs by 0.15% and decreased for indels by 0.03%. We note, however, that GraphTyper was developed by its authors for `bwa mem` mapping.

[5]This was my work along with Nicolas Della Penna, https://github.com/ekg/hhga. Unfortunately, it remains unpublished.

3.2.5 A graph of structural variation in humans

The Human Genome Structural Variation Consortium (HGSVC)[6] has continued the difficult process of cataloging structural variation in humans. Recently, the group has developed a set of haplotype-resolved structural variation callsets for the children in three parent-child trios from diverse populations: NA19240 (Yoruban Nigerian), HG00733 (Puerto Rican), and HG00514 (Han Chinese) [34]. These variant calls are derived from many sources, and are unified into a common framework in phased VCF files. I built a graph from these variants and the GRCh38 reference against which they are represented, then used `vg map` to align short reads from a sample in the HGSVC set (NA19240) as well as a sample that was not included (HG002/NA24385).

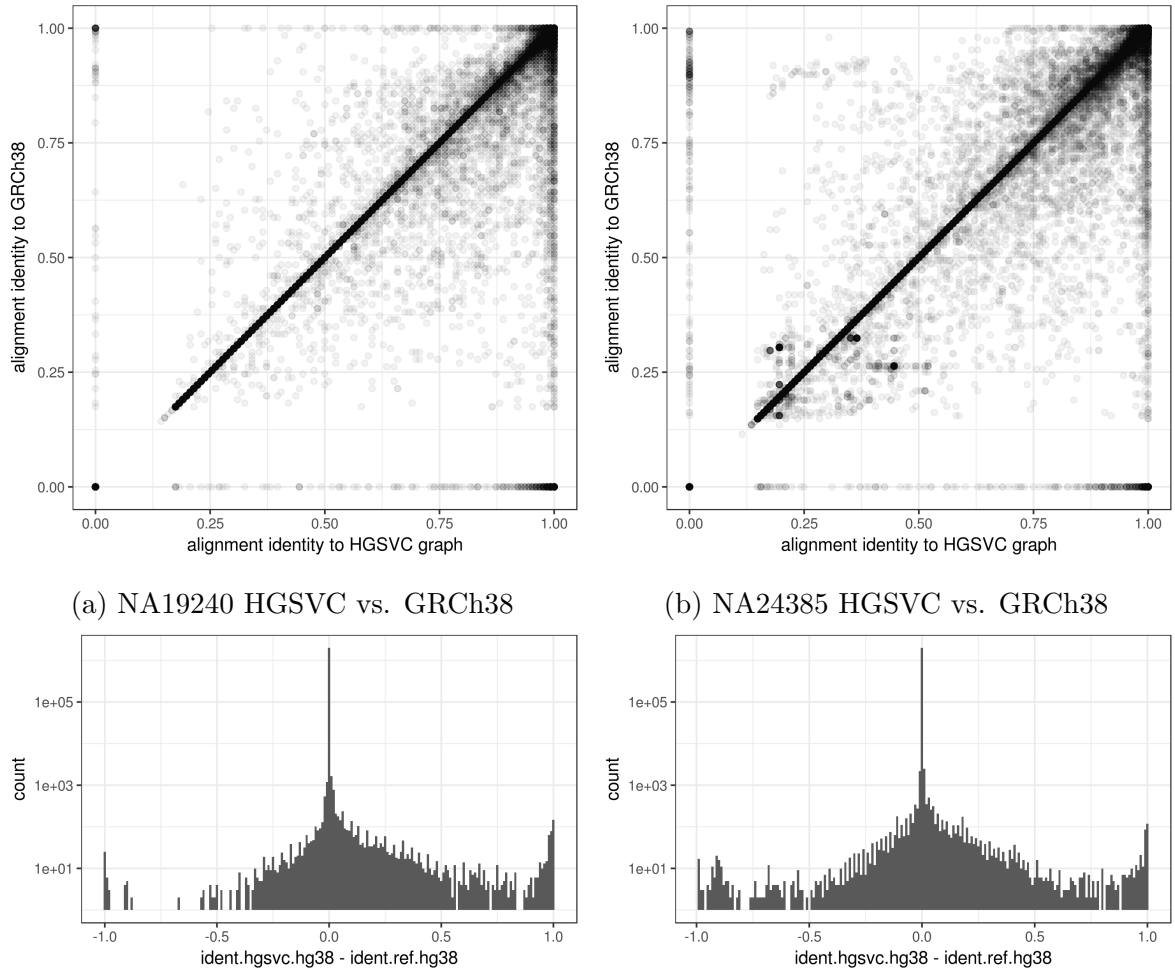
Although I mapped only 1M read pairs, alignments to the HGSVC graph were significantly better (when measured via the identity metric) than alignments to the GRCh38 linear reference (figure 3.8). This was much more significant in the case of NA19240 (two sample T-test p-value = 0.008529) than for NA24385 (p-value = 0.06813). Presumably the lower number of shared alleles with NA24385 means a larger set of reads would need to be mapped to obtain a clear result. The HGSVC graph did not significantly improve alignment scores for 1M random reads (only 2.4% align, with p-value = 0.9889), or for reads sampled without error (p-value = 0.4665) or with 0.5% SNP error and 0.1% indel error from GRCh38 (p-value = 0.9106).

These results are statistically significant and our negative controls validate that the representation in the reference of recurrent SV polymorphisms contributes to the improvement in alignment performance. However, it would be most interesting to see that the variant calling process implemented in `vg call` could genotype the structural variants in these samples. Investigations into this are ongoing, and although the results are promising they are not yet complete. As the HGSVC graph is produced from SV calls in a VCF, we retain the original problems of representing structural variation in VCF. We cannot represent nested variation, and so alleles that are only marginally different are represented as completely separate paths in the resulting graph. This adds ambiguity to the mapping and apparently causes problems when attempting to use the graph for variant calling.

3.2.6 Progressive alignment of human chromosomes

There are only a few truly *de novo* human genome assemblies which achieve near-complete chromosomes, and so a reference-guided variant detection approach has prevailed for

[6] http://www.internationalgenome.org/human-genome-structural-variation-consortium/



(a) NA19240 HGSVC vs. GRCh38

(b) NA24385 HGSVC vs. GRCh38

(c) NA19240 HGSVC - GRCh38

(d) NA24385 HGSVC - GRCh38

Fig. 3.8 1M 2x150bp Illumina read pairs from NA19240 (which is in the HGSVC graph) and NA24385/HG002, which is not, aligned against both the HGSVC graph and the GRCh38 reference, then compared. In panels 3.8a and 3.8b the difference in performance is seen by the alignments which have positive identity against the HGSVC graph but 0 identity against the GRCh38 reference. Panels 3.8c and 3.8d are log-scaled histograms of the difference in alignment score. In these we can observe a small subset of reads which map to the HGSVC graph but not GRCh38 as increased density at 1.0.

the discovery of novel structural variation [75]. Using `vg msga`, I explored if it would be possible to use the HGSVC reference-guided assemblies directly in progressive alignment. It was possible to produce the progressive alignment of the six haplotypes for chr20 on system I used at the Sanger for most experiments, which has 256 GB of RAM and 32 vCPUs. However, doing so took more than week of wall clock time, suggesting that this approach is untenable in its current form. I was not able to complete the progressive alignment of the six haplotypes of chromosome 2. Optimization may improve the performance of `vg msga`, but the linear nature of the approach suggests that aligning more than a handful of sequences will never be feasible.

The progressive assembly can be seen to compress the input. The resulting graph contains 76,875,262bp of sequence, while the input FASTA file of the six haplotypes contains 386,748,228bp, around a 5-fold compression. This result serves to demonstrate that the hierarchical alignment process can scale to many tens of megabases. The low performance of the method prevented its application to the whole genome.

3.2.7 Building graphs from the MHC

In [203] we explored methods to build graphs from the GRCh38's ALT sequences and reference genome. One of the most challenging regions is the Major Histocompatibility Complex (MHC) on chromosome 6. In this ∼5Mb region, balancing selection has generated a high level of genetic diversity, and today we observe up to 40 million years of divergence between alternative copies of the locus, dating far back into the primate lineage. Previous efforts to build reference graphs from this region have often required hand curation to achieve reliable results [63].

Automatically building a sensible MHC graph with tools in `vg` requires that the tools work correctly, and over the course of my work I have used this region as an important test case. It took nearly a year for `vg msga` to mature enough to build the MHC graph without crashing, and another two years before my long read alignment implementation became capable of developing a sensible result.

My further interest in the problem of building graphs from sequences yielded `seqwish` (section 2.2.6). `seqwish` losslessly induces the variation graph implied by a set of sequence alignments. It implements a disk-backed bidirectional alignment graph akin to that described in [131], and with a similar objective as tools developed in [123]. The goal is to build a pangenome graph in which pairwise alignments between the sequences define the graph. Unlike `vg msga`, it is fully dependent on an input alignment set, and is not progressive.

As an exposition of the differing solutions to this problem offered by these two methods, I built the MHC graph from the 9 haplotypes overlapping the MHC in GRCh38[7]. To build the `seqwish` graph, I used [157] to generate an all versus all alignment[8], and induce the graph from this[9]. With `vg msga` I supplied suitable parameters to encourage greater colliniearity in the alignment, but otherwise used defaults[10]. `seqwish` used very little memory, never more than around a gigabyte, but wrote 12 GB of intermediate files to disk during graph induction. `vg msga` used several times this much RAM at peak, indicating that it will be difficult to scale its application beyond small genomic regions. Both processes took around 2 hours to complete on the 256GB/32vCPU server I used for most of the experiments in this thesis.

The `seqwish` MHC graph contains 9,764,108bp of sequence, in 224,873 nodes and 321,990 edges, while the `vg msga` assembly is larger, with 10,900,412bp of sequence in 480,734 nodes and 536,592 edges. Unlike the `vg msga` graph, whose nodes are cut to be shorter than 32bp to enable GCSA2 indexing during progressive construction, the seqwish graph is fully compressed by default, and this results in its much lower node count. We can compare the aggregate results for `seqwish` (figure 3.9a) to those for `vg msga` (figure 3.10). It becomes clear from these visualizations that the `seqwish` graph compresses its repeats much more than `vg msga` (figure 3.9b). However, the alignment in general is sensible in that unique alignments between the input sequences generate linear components, as seen in panel 3.9c. This compression tends to confuse operations on the graph, as it reduces the distance between all positions in the graph and causes overlaps of many genomic regions in the pangenome graph. A different parameterization of the aligner might need to be used to reconstruct the approximately nature of this genomic locus.

To appreciate the effect of repeat collapse on the resulting graph, I applied `vg dotplot`, which generates path-coincidence dotplots from indexed variation graphs, to compare the in-graph alignments of the same pair of sequences in both the `seqwish` and `vg msga` graphs. These results are presented in figure 3.11.

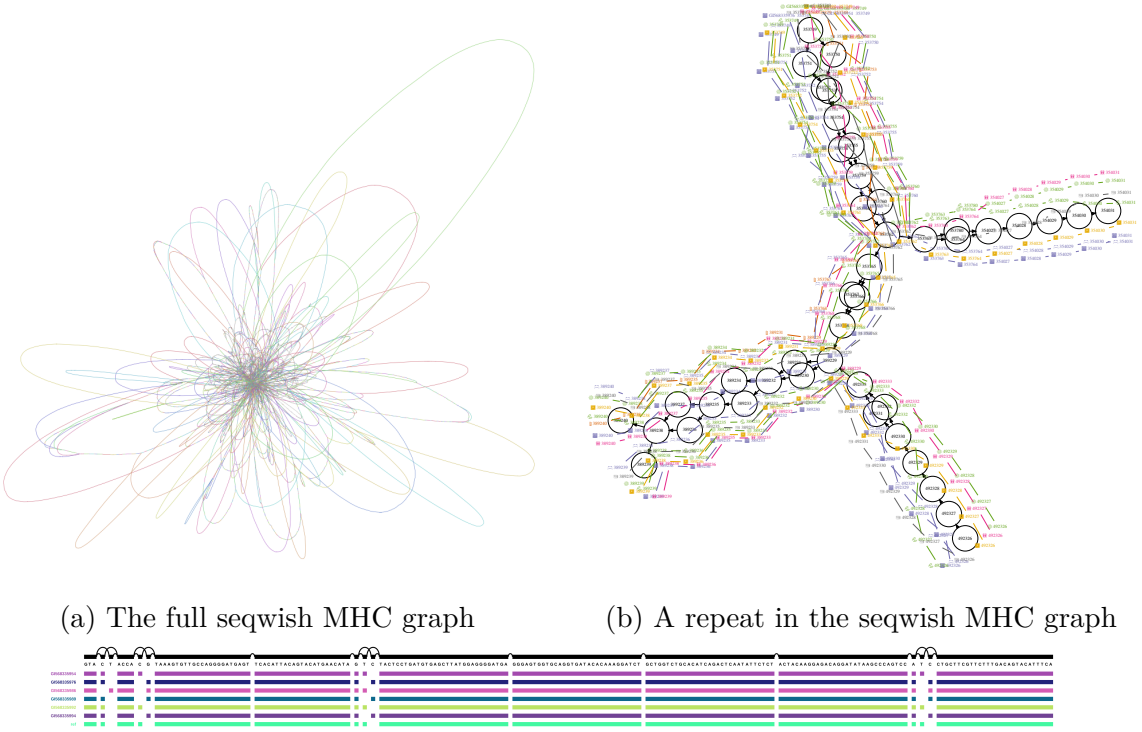
`vg dotplot` does not make a dotplot in the same manner as is done to represent pairwise alignments. Rather, it produces plots that represents the relationship between pairs of sequences as they are embedded in the variation graph. For a given pair of paths embedded in the graph, `vg dotplot` renders a dot for each instance where both

[7] These were collected for use in the GA4GH-DWG by the human genome variation map (HGVM) project [22].

[8] `minimap2 MHC.fa MHC.fa -c -X -x asm20 -t 32 >MHC.paf`

[9] `seqwish -s MHC.fa -a MHC.paf -b MHC.seqwish >MHC.gfa`

[10] `vg msga -f MHC.fa -w 512 -J 16384 -b ref -D >MHC.vg`



(a) The full seqwish MHC graph

(b) A repeat in the seqwish MHC graph

(c) A linear component in the seqwish MHC graph

Fig. 3.9 Seqwish assembly of the MHC in GRCh38. In panel 3.9a Bandage was used to gain a view of the full graph. The fine structure of the graph is similar as for the earlier yeast assemblies. The core of the graph forms a hairball of repeats, as in panel 3.9b flanked by low-copy loops which are locally directed and acyclic as in panel 3.9c, which adjoins the subgraph plotted in panel 3.9b.

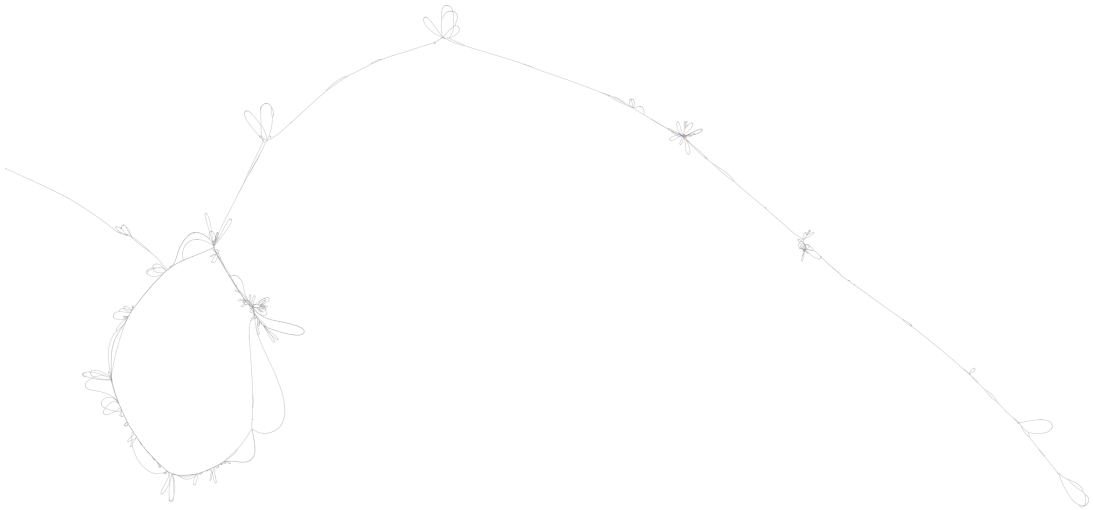
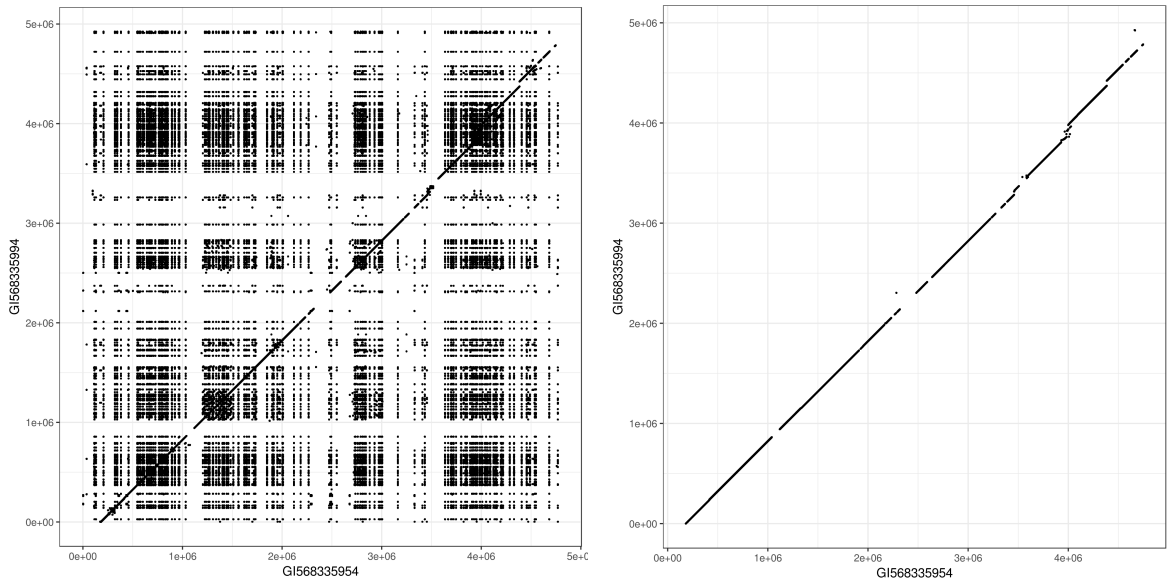


Fig. 3.10 Variation graph of the MHC in GRCh38 constructed by progressive alignment with `vg msga` and visualized by Bandage.

paths traverse the same graph position. The dot is shown at a position (x, y) where x corresponds to the position in one path, and y in the other. In the case of a graph built from the pairwise alignment of two sequences, the resulting path-coincidence dotplot would be equivalent to a traditional dotplot made from the alignments. However, for progressive alignments, we cannot be guaranteed the same structure for any given pair, due to the order dependence of the inclusion of each sequence.

It is clear (in 3.11a) that the `minimap2` based alignment process encorages the collapse of repeats, which results in the large off-diagonal blocks of graph positional matches between the sequences. `minimap2` allows for the multiple mapping of parts of its query sequences, but no filtering was completed on the input alignments to reduce the impact of multimapping on the resulting graph. In contrast, figure 3.11b shows that a different alignment model allow the sequences to align through the graph in an approximately linear fashion, with little repeat collapse. This linearity is enforced by setting a high alignment chain model "bandwidth" parameter (`-J 16384`) (described in section 2.5.9), which limits the size of an insertion or deletion that can be tolerated in collinear chaining. With a primary alignment chunk size of 512 and overlap of 64, this yields an alignment chain model bandwidth of approximately 6Mb, which roughly covers the entire MHC for any input sequence. Setting a shorter bandwidth allows repeat collapse to occur in a manner similar to what occurs when inducing the graph from the `minimap2` alignments.

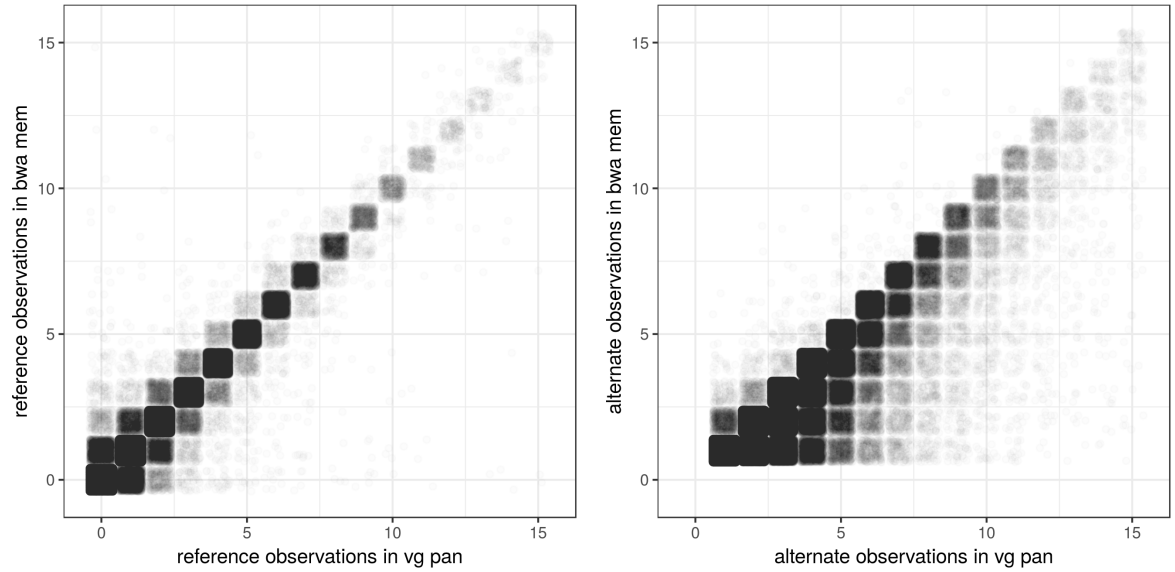


(a) `seqwish` (b) `vg msga`

Fig. 3.11 Comparing `seqwish` and `vg msga` variation graphs built for the MHC in GRCh38. In both panels we see a path-coincidence dotplot between GI568335954 (x-axis) and GI568335994 (y-axis) as they are embedded in each graph, with 3.11a showing the plot for `seqwish` and 3.11b that for `vg msga`. In these plots, we render a dot for each pair of path positions that coincide in the graph. The repeated positions in the `seqwish` plot, which appear as dark verticals banded regions in the plot, represent repeats that have triggered the collapse of multiple positions in each path onto the same graph position. In contrast, the `vg msga` plot shows a relatively linear relationship between the two sequences in the graph, with large indels and a few gaps, but no off-diagonal relationships that might suggest repeat collapse or structural variation.

3.2.8 CHiP-Seq

The removal of mapping bias is important when working with functional genomics data such as ChIP-seq data, where allele-specific expression analysis can reveal genetic variation that affects function but is confounded by reference mapping bias [181], especially given that read lengths are typically shorter for these experiments. We compared mapping with `bwa mem` and `vg` for data set ENCFF000ATK from the ENCODE project [47], which contains 14.9 million 51bp ChIP-seq reads for the H3K4me1 histone methylation mark from the NA12878 cell line. When mapping with bwa the ratio of reference to alternate allele matches at heterozygous sites was 1.20, whereas with `vg` to the 1000GP graph the ratio was 1.01, effectively eliminating reference bias.



(a) Reference observations (b) Alternate observations

Fig. 3.12 Resolving reference bias in 36bp CHiP-seq data from ENCODE sample ENCDO115AAA, using CHiP-seq targeting H3K4me1 (experiment ENCFF000ATK). We compare the counts of reference-supporting (panel 3.12a) or alternate-supporting (panel 3.12b) alignments between `vg map` and `bwa mem` at sites called as heterozygous in the sample which were also polymorphic in the 1000GP. To shown density we have jittered the points in the unit square centered on the integral values taken by the counts.

The effect is dramatically stronger when using 36bp reads. To illustrate, I aligned the ultra-short reads from ENCODE experiment ENCSR290YLQ[11] to the full 1000GP graph using `vg map` and to the GRCh37 reference genome using `bwa mem`. To simplify analysis, the `vg` mappings were surjected to the GRCh37 reference. This sample came from an

[11] https://www.encodeproject.org/experiments/ENCSR290YLQ/

adult donor, and so we have no "truth" set as with NA12878, but we can use a subset of sites called as variable using `freebayes` which also are known to be polymorphic in the 1000GP to examine the effect of reference bias. By plotting the reference and alternate counts obtained from each aligner relative to each other, we can observe the effect of reference bias on a per-site basis. I find that `bwa mem` and `vg map` obtain insignificantly different counts of reference observations at the heterozygous sites (panel 3.12a), but as can be seen in panel 3.12b, aligning to the graph uncovers dramatically more alternate observations at these loci, as is shown by the strong increase in density below the diagonal. The strength of this effect indicates that small variation will disrupt alignment of very short reads of relatively high quality and low error rates. The issue becomes even more dramatic when we consider data sources with higher error.

3.3 Ancient DNA

In suitable conditions, DNA can survive for tens or even hundreds of thousands of years *ex vivo*, providing a unique window into the past history of life [52]. However, the sequencing analysis of Ancient DNA poses several significant challenges. The amount of DNA available is often limited, and sequencing costs are increased in the presence of high rates of contamination by organisms that inhabit the sample after its death. Read lengths are limited by the degradation of DNA due to necrotic processes and subsequent environmental exposure. Post-mortem damage (PMD) of the DNA occurs at a high rate, introducing mutations in the tails of the short DNA molecules, which occur in a single-stranded and relatively unprotected state [167]. This manifests mostly as the conversion of cytosines to uracil, but also can lead to apurination [52]. Ancient DNA data is thus often of short length, low coverage, and high intrinsic error rate.

In combination these issues cause a strong bias against non-reference variation, which can have a significant effect on population genetic inference and implications for many aDNA studies. Ancient DNA may be treated with uracil-DNA-glycosylase (UDG) and endonuclease VIII to remove uracil residues and abasic sites, leaving undamaged portions of the DNA fragments intact [26]. However, this process results in a reduction of read length and library depth, which is disadvantageous, and it is not guaranteed to proceed without introducing new bias. There have also been many attempts to mitigate the effects of reference bias and low coverage, such as by implementing a model of reference bias in gentoyping [220], or by working with genotype likelihoods throughout all downstream population genetic analyses [167]. To avoid genotyping or even the generation of genotype

likelihoods, standard practice often involves modeling each genome as a collection of haploid chromosomes modeled by reads.

Here I report on the results of an ongoing collaboration with Rui Martiniano and Eppie Jones to explore the use of `vg` to mitigate reference bias in ancient DNA samples. They have completed most of the analyses, while I have supported their work and developed the alignment (and surjection) algorithms that are essential to them.

3.3.1 Evaluating reference bias in aDNA using simulation

Using simulation, Eppie Jones and I completed an exploratory analysis which demonstrates that the high degree of reference bias inherent in ancient DNA analysis may be mitigated at known sites by aligning against a pangenome graph. We simulated all possible 50 bp reads which spanned variant sites on chromosome 11 of the Human Origins SNP panel [214, 146], which is a set of SNPs designed to be highly informative about ancient genomes and human ancestors. In half of the simulated reads the SNP position was mutated to the alternate allele. We then mapped these reads back to the 1000GP graph or GRCh37 linear genome using `vg map` and `bwa aln` respectively, with `vg`'s surjection process used to produce a BAM file from the alignment for direct comparison. Different levels of ancient DNA damage estimated using 100 ancient genomes from [5] were simulated in these data using `gargammel` [228]. We filtered the resulting alignments for those above mapping quality 30, which has approximately the same significance in both `vg` and `bwa`. Our results are shown in figure 3.13.

At high levels of error, alignment against the linear reference prevents the observation of non-reference alleles in a large fraction of cases. This effect is notable at deamination rates as low as 10%, and with 30% PMD error the rate of alignment to non-reference alleles is reduced by nearly 15% relative to the total. While `bwa aln` suffers a significant reference bias with increasing simulated PMD rates (dashed dark blue versus orange fit lines representing alternate and reference alleles), we observe no such effect for `vg map` (light blue and yellow) (figure 3.13). Most of the alleles in the Human Origins panel are also in the 1000GP, which suggests that modern sources are useful when constructing a pangenomic reference for ancient samples.

3.3.2 Aligning ancient samples to the 1000GP pangenome

To evaluate whether pangenomic read mapping techniques could mitigate reference bias in real samples, we collected data from a number of recently published ancient DNA studies. This includes Iron Age, Roman, and Anglo-Saxon individuals sequenced

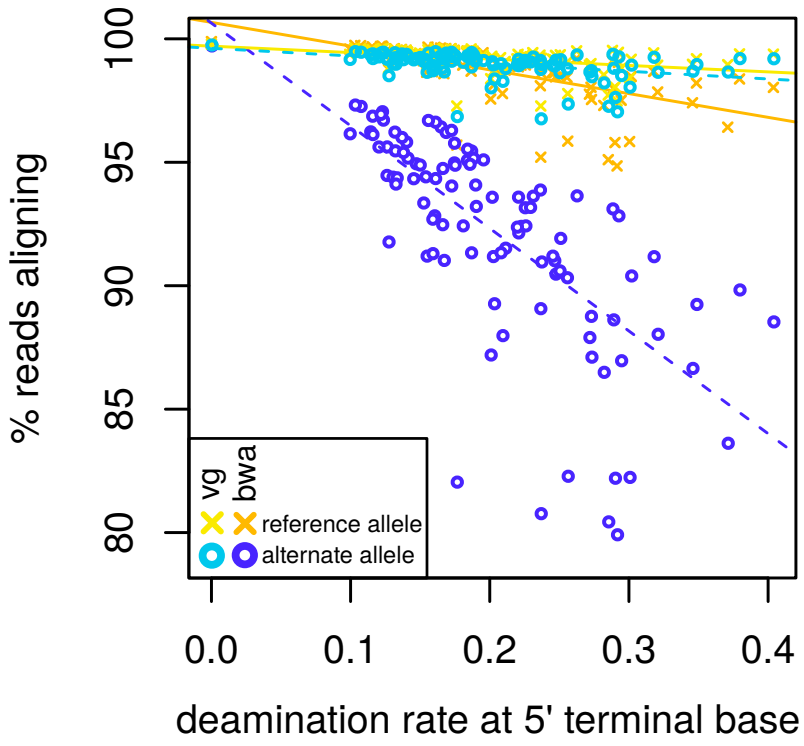


Fig. 3.13 Comparing `bwa aln` and `vg map` performance when aligning reads simulated from chromosome 11 of the Human Origins panel. Lines represent OLS regression results for the allele/aligner conditions corresponding to their colors.

at low coverage in [239] and [180], and high-coverage Yamnaya and Botai individuals from [55]. We used `vg map` to align the samples to the 1000GP pangenome of variants above 0.1% frequency, and `bwa aln` to align them to the GRCh37 reference plus decoys, using standard parameters for the analysis of ancient DNA (`-l1024 -q15 -n0.02`). For both alignment results we filtered the resulting BAMs using `samtools view -q30` to remove reads with mapping quality less than 30, and removed duplicates using `sambamba markdup` [269].

We first considered the high-coverage Yamnaya sample from [55], which provides approximately 20-fold coverage of the genome. This is very high for ancient samples and thus allows us to complete some experiments which would be otherwise very difficult. We called variants on chromosome 21 using bcftools [161] for both `vg` and `bwa` alignments. To evaluate the effect of reference bias with decreasing coverage, we then used these callsets as ground truth and measured our ability to recover the heterozygous variants in the full coverage set at coverage levels of 0.1, 0.2, 0.3, 0.4, 0.5 of the original. As seen in figure 3.14, at lower coverages common in ancient DNA sequencing, `bwa aln` recovers notably fewer heterozygous SNPs than `vg map` alignment to the 1000GP graph. At the sampling rate of 0.2, which corresponds to coverage ≈4, `vg map` recovers 8% more heterozygotes as a fraction of the total. `vg map` recovers more heterozygous SNPs than `bwa aln` at all the coverage levels, although at higher levels the difference is less pronounced. We expect this would have a significant effect on population genetic analyses depending on these alignments.

In an illustrative, if less well-controlled experiment, we simply examined the distribution of the ratio between reference and alternate allele observations in the set of variants called by `freebayes` from the surjected `vg map` Yamnaya alignments. We first subset the called alleles into heterozygous transversions, and further divided the resulting set of putative transversions into those alleles in the 1000GP graph, and those which are not. As shown in figure 3.15, we observe that if the variant is in the graph (N ≈ 1M) we have effectively removed bias towards the reference allele. This is not the case for those called alleles (N ≈ 300K) which are not in the graph.

We expect reference bias of the degree demonstrated in these experiments to affect population genetic inference. To evaluate this, we apply the ABBA BABA test of phylogenetic tree topology based on Patterson's *D*-statistic of population relationship [101]. To do so, we used the Human Origins dataset distributed with [146]. For each sample we ran samtools pileup yielding ∼1.2 million SNPs. We converted the pileup results to plink format, selecting one allele at random from the confidently mapped reads spanning each site, as is standard in the field. This was done 5 times, generating 5

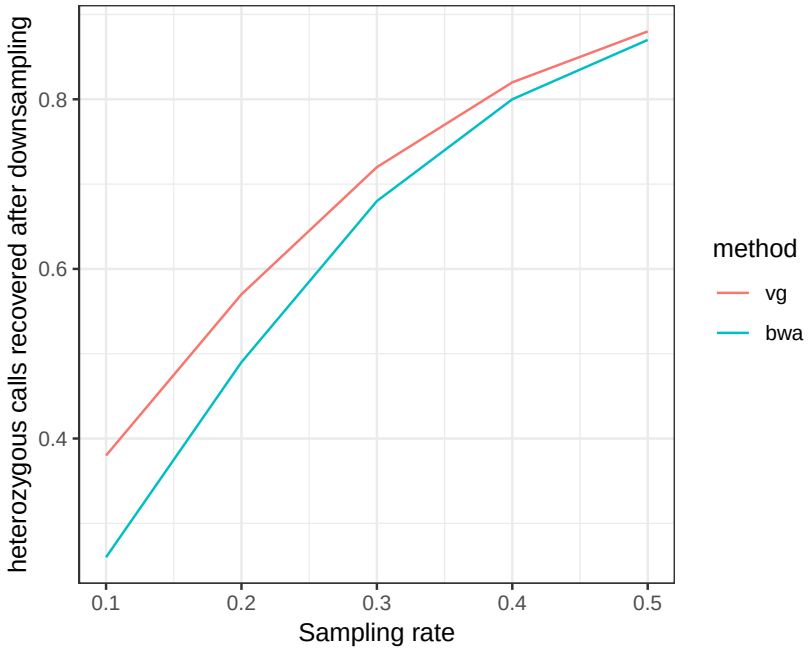


Fig. 3.14 The comparative effect of downsampling of an ancient DNA sample on `bwa aln` and `vg map` alignment.

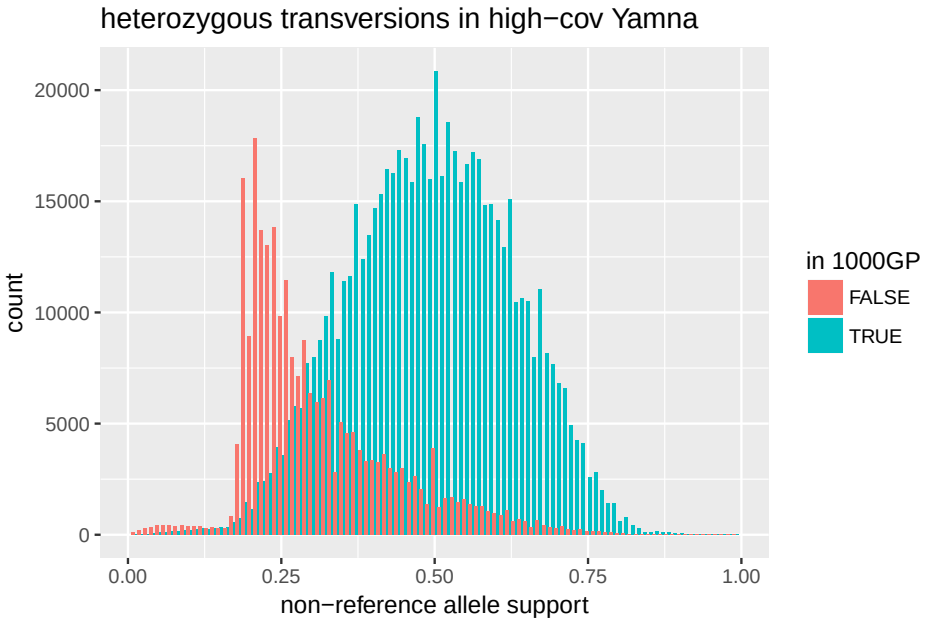
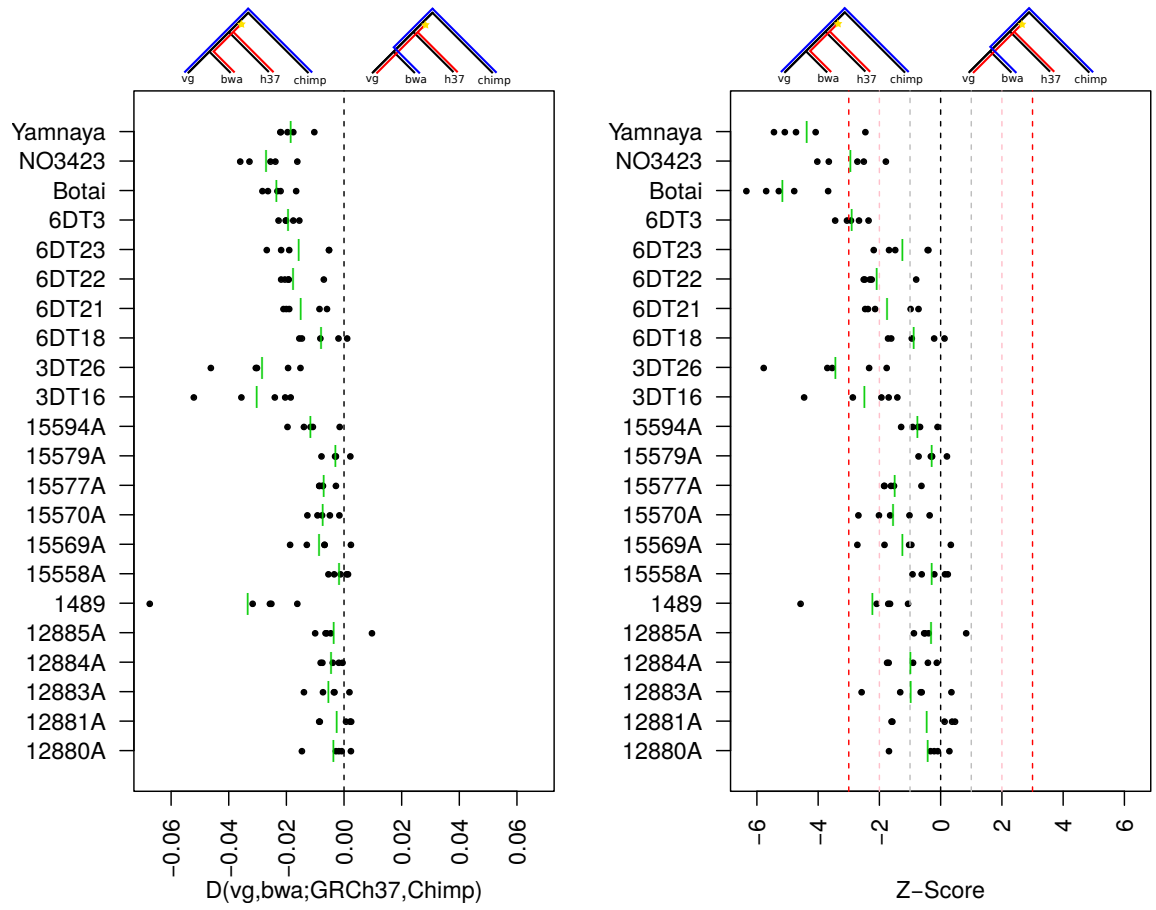


Fig. 3.15 Allele balance in the `vg` alignment of the Yamnaya sample to the 1000GP graph versus its presence in the 1000GP graph.



(a) *D*-statistic absolute values

(b) *D*-statistic z-scores

Fig. 3.16 *D*-statistic based ABBA-BABA test of reference bias in aDNA. Panel 3.16a provides the absolute values for the given *D*-statistic, calculated as described in the text, while panel 3.16b shows estimated significance level for the various samples. On both panels, points to the left of the dashed line at 0 indicate excess allele sharing between `bwa` alignments of the samples and GRCh37 (the ABBA pattern), while those to the right indicate an excess of sharing between `vg` alignments and GRCh37 (the BABA pattern). The two patterns are illustrated by the trees above the panels, where *A* alleles are given as blue lines on the tree and *B* alleles that arose between the split with the outgroup (Chimpanzee) are shown in red.

replicates of randomly selected alleles for each sample. We then filtered by genotyping rate, excluding poorly genotyped SNPs[12], resulting in 1,142,867 SNPs. Finally, we merged individual plink files with those for GRCh37 and Chimpanzee, converted plink to eigenstrat format using `convertf` and calculated D-statistics using qpDstat [214] of the form $D(\text{vg}, \text{bwa}; \text{GRCh37}, \text{Chimp})$ to test for excess of shared alleles between `vg` and `bwa`-aligned samples with the reference.

Our results, summarized in figure 3.16, indicate an excess of allele sharing between the `bwa`-aligned samples and GRCh37 relative to the `vg`-aligned ones and GRCh37. All but a handful of replicates have negative D-statistics, which implies that alignment against GRCh37 makes the samples appear more like the reference (subfigure 3.16a). We conclude that reference bias is strong enough to affect common population genetic analysis completed with ancient DNA, and this can be mitigated at least for known alleles by aligning against a pangenomic reference.

3.4 Neoclassical bacterial pangenomics

As I discussed in the introduction (section 1.3), during much of the past decade pangenomic concepts have been employed in microbiology to characterize the relationship between strains that frequently share DNA through horizontal gene transfer and exhibit wide variability in gene content. It is often helpful to consider the "core" and "accessory" pangenome of a given clade, with core genes being those present in all strains and accessory ones those present in only a subset. Techniques to evaluate these features of a pangenome are often based on gene counting approaches that can be driven by simple k-mer matching.

In this section I demonstrate that `vg` can be applied to derive the same kind of result by directly working on a pangenome graph reference built from diverse strains of *Escheria coli*[13]. The basic idea is to build a pangenome using a standard DBG assembler, then align all the strain level data we have available against the resulting assembly graph. The advance that `vg` offers is precision. Rather than considering gene-level counts, alignment with `vg` provides per-base, per-sample coverage information. The results can be used as in resequencing to find new variants, and to genotype existing ones in the graph. This approach is similar to colored de Bruijn graphs [119], but it allows for out-of-core

[12] We used the plink filter `--geno 0.05`.

[13] I developed this technique during the Computational Pangenomics (CPANG18) course at the Instituto Gulbenkian de Ciéncia in Oieras, Portugal. See https://gtpb.github.io/CPANG18/. This particular practical was explored on the third day of the course https://gtpb.github.io/CPANG18/pages/bacteria.html.

computation, retains full alignment information, and is consequently (in principle) lossless with respect to the input reads.

3.4.1 An *E. coli* pangenome assembly

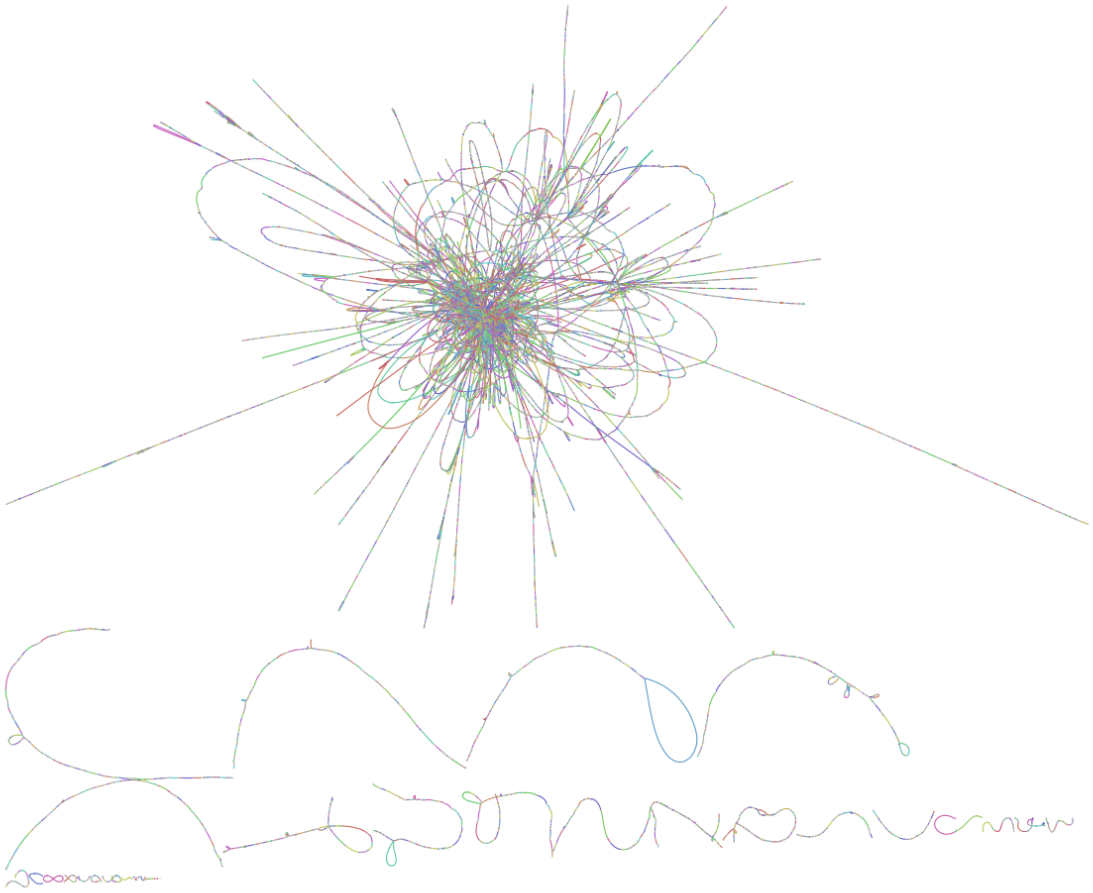
In line with early work on bacterial pangenomics [182], I used a handful of genomes to demonstrate the utility of pangenomic approaches in *E. coli*. I collected Illumina data from 10 strains published in [68][14]. These data total 2.3 GB of compressed FASTQ in 2x151bp paired end format. I built a compressed DBG using `minia` with $k = 51$ and an abundance minimum of 10, which was selected due to the very high genomic coverage offered by this data, although I did not observe large changes in assembly structure when varying it. I then processed the resulting graph with `vg` to decompress long nodes into nodes with a maximum length of 32bp, then sort it pseudotopologically and compact its node identifier space. These steps are required for efficient mapping given the absence of any annotated reference paths to provide distance estimates. As seen in figure 3.17, the assembly graph features a giant component representing most of the input sequencing data. Other fragments could represent plasmids or other contaminating elements. The normalized graph contains 10,456,557bp of sequence, in 429,377 nodes and 455,892 edges. The facts that the average node length is over 24bp and that there are only 1.06 edges per node suggest that despite its tangled appearance, it is mostly composed of linear components. It took less than an hour to assemble and index the graph, yielding a 6.3MB `vg` graph, a 37MB `xg` index, and a 47MB `GCSA2` index.

3.4.2 Evaluating the core and accessory pangenome

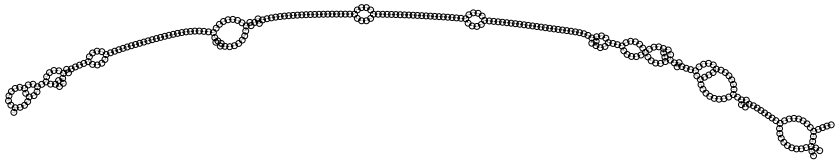
The generation of the compressed DBG loses the read-level relationships to the graph, but these may be obtained trivially with `vg` by aligning the read set back to the assembly graph. Doing so required around 5 minutes per sample, yielding 3.3 GB of GAM files in around an hour. To obtain coverage counts per sample across the graph I then applied the `vg pack` utility to project the alignment GAMs into coverage maps for each sample, which I collated for processing in R.

Figure 3.18a shows the coverage across a typical region of the pangenome. We see regions shared by all strains, as well as others which are particular to a single or small number of strains. This is exactly in line with our expectations based on observations from "classical" bacterial pangenomics. To quantify this over the whole pangenome, I

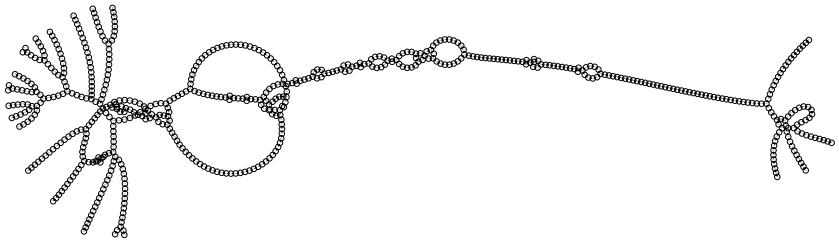
[14]SRA accessions SRR3050857, SRR3050919, SRR3050929, SRR3050978, SRR3050990, SRR3050992, SRR3050994, SRR3051002, SRR3051049, and SRR3051079.



(a) The full *E. coli* assembly graph.

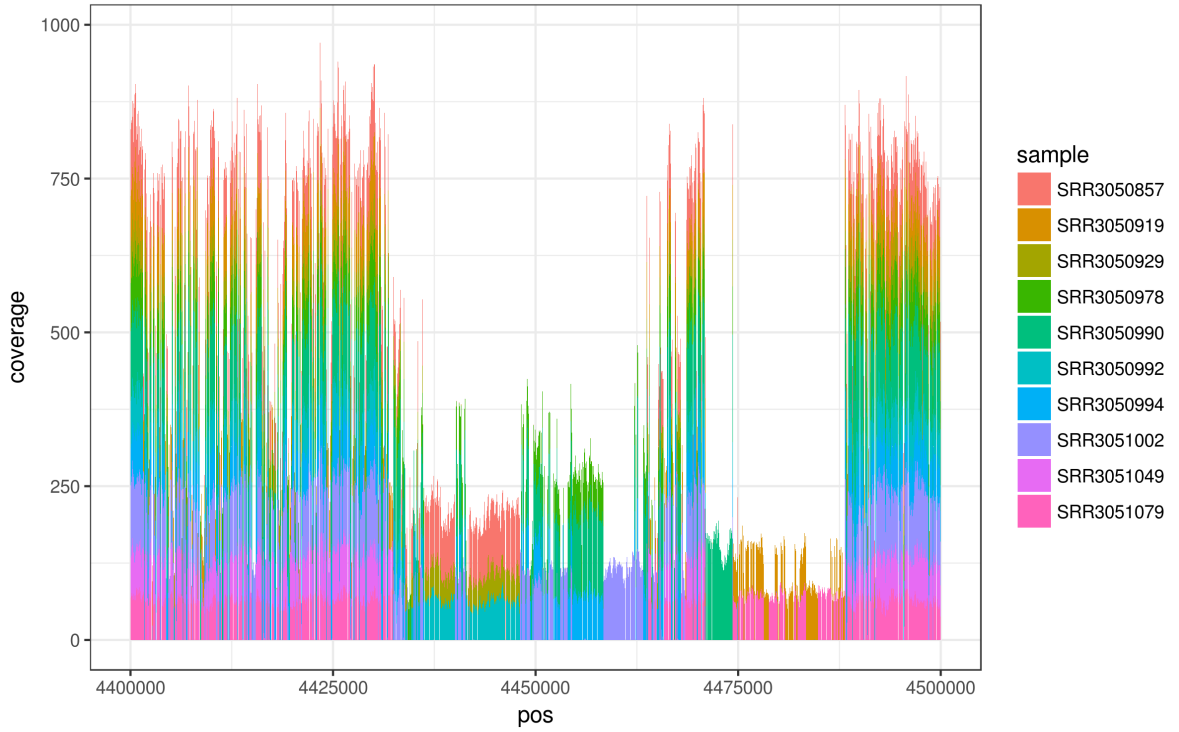


(b) A linear region of the assembly graph, with variation.

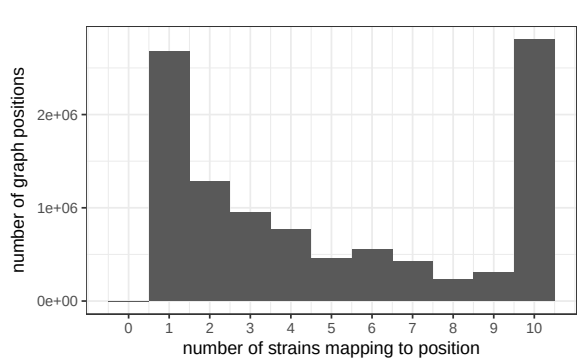


(c) A semi-tangled region of the assembly graph.

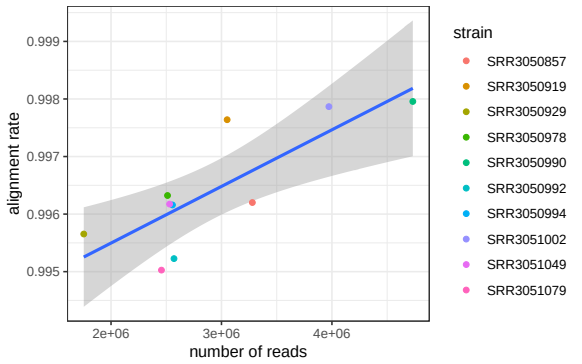
Fig. 3.17 An assembly graph built from 10 *E. coli* strains using minia. In the full graph we observe a single giant component (panel 3.17a). The graph is largely linear (3.17b). However, in some regions it is densely tangled (3.17c).



(a) Per-strain coverage across 100kbp of the *E. coli* pangenome



(b) Genomes per base of the pangenome



(c) Realignment rate versus input read count

Fig. 3.18 Evaluating alignment to the *E. coli* pangenome. Panel 3.18a shows per-strain coverage across a selected region of the ≈10MB pangenome, including several regions specific to a subset of the strains. Panel 3.18b provides a histogram of the number of genomes covering each base of the pangenome demonstrates that much of the genome is in all strains or in single strains. Although the alignment rate of the input reads to the pangenome graph is in all cases is above 99.5%, we see a significant correlation between this rate and the number of input reads from each strain used to construct the graph (panel 3.18c).

computed the histogram of graph positions by the number of strains mapping to each position, as summarized in figure 3.18b. I find that 26.9% of the pangenome is covered by all ten strains, while 25.6% is covered by only one strain. The remaining 47.5% of the pangenome has intermediate numbers of strains mapping to it. As such we can conclude that around $\frac{1}{4}$ of the pangenome is the conserved "core," and $\frac{1}{4}$ is purely "accessory," occuring in only a single strain.

Virtually all the graph is covered by some strain, with only 80bp receiving no mappings, which indicates that the pangenome object is complete. However, the read sets are not equivalently contained in the pangenome, and a significant ($p = 0.00534$) correlation exists between the realignment rate of the reads to the pangenome and the number of raw reads from each strain, as seen in figure 3.18c. This statistic suggests that a few strain-specific regions are not assembled. Naturally, the rate at which this failure occurs is dependent on read depth. It is possible that this effect is driven by the high abundance count (10) used in constructing the compressed DBG, and possibly it could be mitigated by tuning this parameter and the k-mer length.

3.5 Metagenomics

Metagenomics is conceptually close to pangenomics. But, where pangenomics considers a single clade, metagenomics considers the total DNA in an environmental context. This adds significant complexity to the analysis of metagenomic data. To obtain high coverage across all the extant genomes in a sample may be impossible, as many exist at extremely low copy. Furthermore, doing so could be extremely expensive. In response, reductive approaches focused on sequencing 16S ribosomal DNA or other conserved gene sequences have been popular [275]. These methods are only applicable where such homologous sequences are available, and so cannot be applied to viral contexts [70]. As sequencing costs have decreased, assembly techniques have become more important to metagenomics [204]. A typical workflow involves assembling contigs out of a DBG, mapping these contigs into annotated databases using sensitive alignment with BLAT or BLAST, and computing abundance by aligning the read set back to the contig set. Contigs, in being unary, suppress variation that could be useful in analysis. Furthermore, in graphs that are dense, contigs will be very short, and linear read alignment will fail to completely capture the set of contigs that a given read maps to. This will result in distortion in coverage estimates, and at very least loss of information due to the breakage of sequences that cross contigs boundaries. The information loss suggests that a technique based on alignment to the assembly graph itself could benefit downstream analysis. In this

section I use analyses of viral and bacterial metagenomes to demonstrate that `vg` enables contiguous alignments against topologically complex metagenomic assembly graphs.

3.5.1 Arctic viral metagenome

To explore this issue, I built an assembly graph from a single sequencing data set from a study of freshwater viruses in Svalbard [57][15] and compared the performance of alignment against the graph with alignment against the contig set. As with other assembly graph based variation graphs, I used `minia` to construct the graph, with $k = 51$ and a minimum abundance of 3. To enable efficient alignment against the graph, I chopped its long nodes into nodes of a maximum of 32bp. The resulting graph has 39,503,775bp of sequence in 1,387,287 nodes and 1,326,573 edges. The low ratio of edges to nodes suggests a locally linear structure, which enables direct use of the graph by `vg`. In fact there are fewer edges than nodes, which also means the graph contains many isolated components. To successfully index the graph I found I had to remove complex regions using `vg prune`, cutting edges that induced 3 bifurcations in any 16-mer. Indexing yielded a 119MB `xg` index and 158MB `GCSA2` index.

As expected given the small genome size of most viruses, the graph contains many small components (figure 3.19). The graph also contains a few large components, and many of the medium-sized components are circular, which is expected given the nature of many DNA viruses. Some components are topologically complex, which can be seen in figure 3.19 as "hairball" like structures. When using $k = 31$, indexing and alignment against the graph were impossible without a high-degree filter which removed nodes in such regions, however in the case of $k = 51$ it is possible to index with only standard pruning of the DBG.

I held out 100,000 reads from the input to the assembly graph. To evaluate the utility of using this graph as a reference for alignment, I aligned these 100,000 reads using `vg map` and `bwa mem` to the assembly graph and linear contig set respectively. Although both methods mapped ∼96% of the reads, `vg` had an average identity score of 95% compared to 87% for `bwa`, reflecting that the `bwa` alignments in many cases are not full length while those of `vg` are. Furthermore, many reads mapped with higher scores (longer matches) to the graph than to the contig set (figure 3.20).

[15] Data from https://www.ebi.ac.uk/ena/data/view/ERS396648

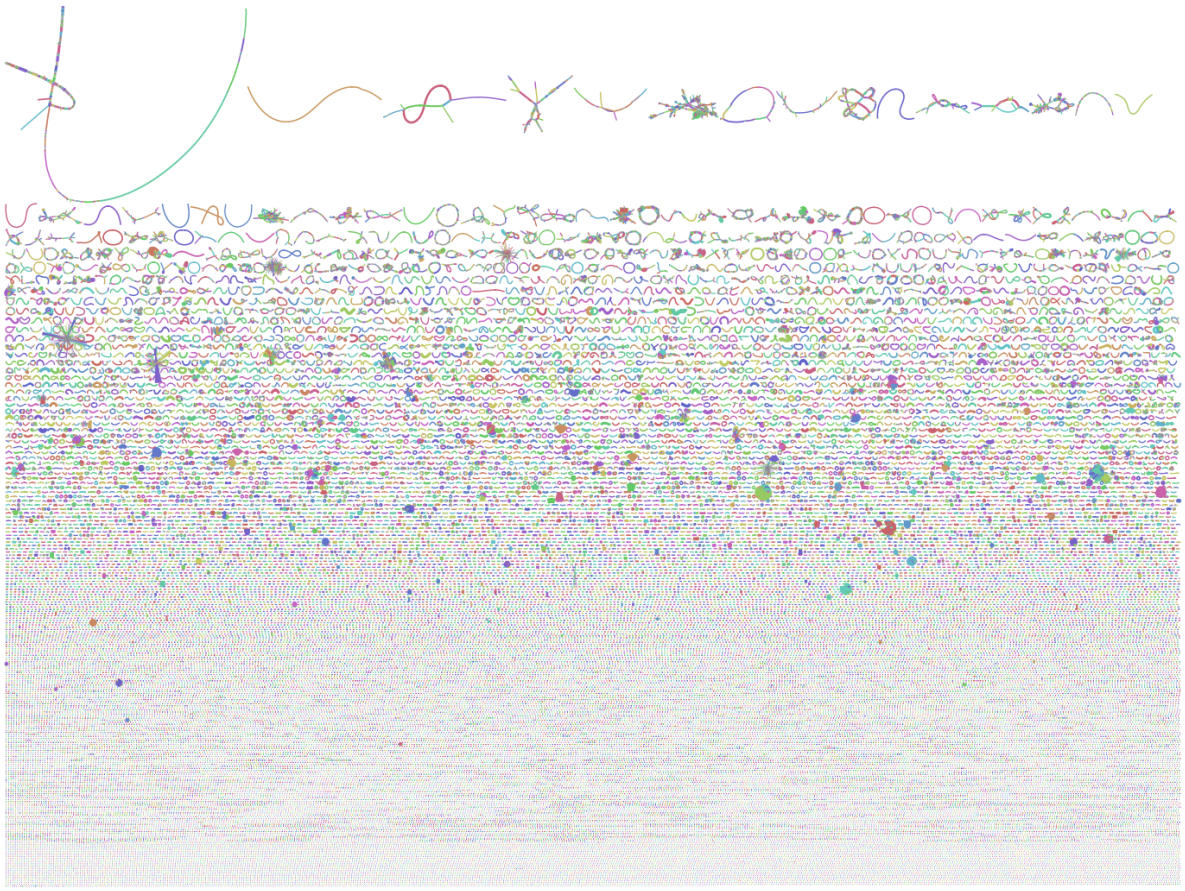


Fig. 3.19 An arctic freshwater viral metagenome graph, built with `minia` as described in the text. As in other Bandage visualizations, colored splines in this figure represent contigs or nodes in the assembly graph, with edges represented by junctions connecting the ends of these splines. The width of the nodes corresponds to the coverage measured by `minia` during the assembly process. Bandage renders graphs with many small contigs with a vertical order related to the size of each weakly connected component, so the nodes at the bottom are smaller than those at the top. The presence of many small, often circular contigs in this figure demonstrates the viral origin of the DNA sample used to build the graph. We should expect these to represent fully assembled viral genomes [57]. The fact that no giant component emerges in this graph indicates that there is sufficient sequence divergence between the viral species to prevent collapse in the $k = 51$ order de Bruijn graph. The widely varying width of the nodes in the graph indicate that some viral species are vastly more abundant than others. We can see that there are few very short contigs (at the bottom of the rendering) with high coverage, which may indicate that these represent partial assemblies, rare species, or contaminants from non-viral sources.

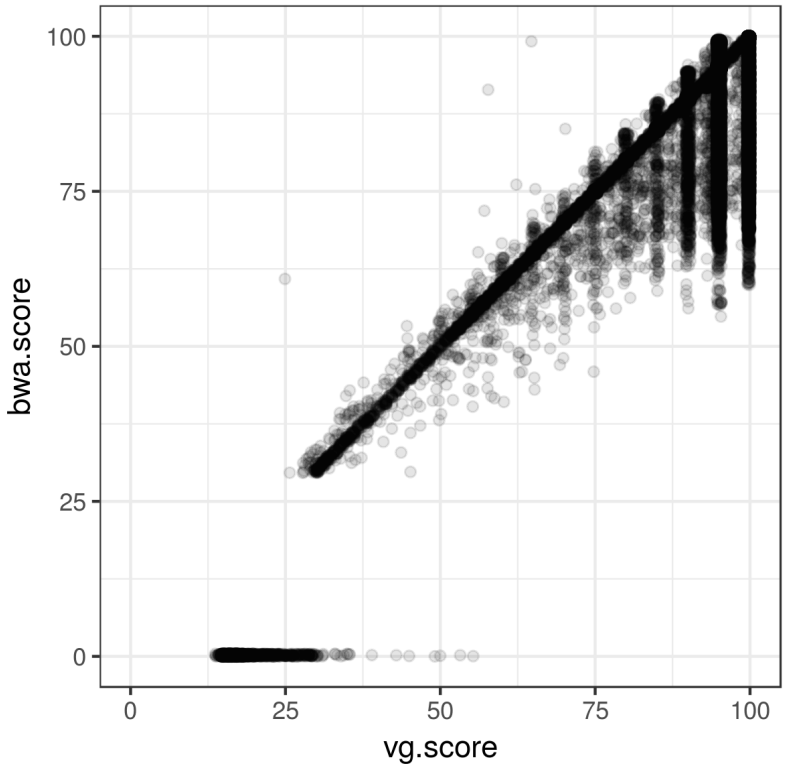


Fig. 3.20 Comparison of alignment score between `vg map` and `bwa mem` when aligning against the pangenome graph or assembled contig set of the graph. The region of higher density seen below the diagonal in the upper right of the plot indicates an increase in alignment score when mapping against the full assembly graph (with `vg map`) relative to mapping to the linearized contig set (with `bwa mem`).

3.5.2 Human gut microbiome

As a counterpoint to the viral metagenomic sample used in the previous section, here I present a similar analysis based on a predominantly bacterial human gut microbiome obtained from a stool sample[16] as part of the Human Microbiome Project (HMP) [276, 216].

As with other assembly graph based analyses, I applied `minia` with $k = 51$ and a minimum abundance filter of 3 to the 29 GB of compressed FASTQs from this sample. The resulting assembly graph is shown in figure 3.21. It contains 144,829,925bp of sequence in 4,788,652 nodes and 4,665,919 edges, and as with other DBG assemblies I have discussed is locally linear. Building and indexing the graph took around an hour on the 32vCPU/256GB server used in other experiments, and yielded a 74MB `vg` graph with a 403MB `xg` index and 606MB `GCSA2` index, which was built with pruning with a 3 edge crossing limit in $k = 16$.

Human gut flora are primarily comprised of two main phyla, the Firmicutes and the Bacteroidetes [174]. We can observe their presence in this sample in the structure of this assembly graph. By aligning the node sequences in the graph against a subset of RefSeq Genomes which are commonly found in the human git microbiome, I verify that the two ends of dumbbell-like main component of the graph (figure 3.21) represent these phyla. The connection between them may be generated by horizontal transfer of genes or spurious correlation due to recurrent k-mers in the assembly graph, and it is notable that many of the nodes in the bridge between the two components show a high depth in the assembly graph, as shown by the node width in the figure. *E. coli* can be found on a small component connected to this bridge, shown at the top middle of the rendering.

To evaluate the utility of this graph for read alignment, I aligned 100k of the held-out reads with `vg map` and `bwa mem` to the graph and the contig set respectively. This contig set would appear to be more suited to use as a linear reference and `bwa mem` alignment than the viral metagenome in described in the previous section. Alignment to the graph marginally improves the number of aligned reads, with `bwa mem` aligning 95.7% of them and `vg map` aligning 96.3%. Overall, we can observe a similar level of change in the alignment score, with `vg`'s mean score at 92.5 and `bwa`'s at 92.0, which is a small but significant difference (two-tailed T-test $p = 3.62 \times 10^{-7}$). The longer length of contigs in this graph and the lack of small circular contigs apparently contribute to the reduction in relative improvement offered by alignment to the graph in this bacterial context. A scatterplot of alignment scores between the two methods reveals a slight shift in density

[16] Sample described at https://portal.hmpdacc.org/cases/3674d95cd0d27e1de94ddf4d2e0398c4, with data from http://downloads.hmpdacc.org/data/Illumina/PHASEII/stool/SRS105153.tar.bz2.

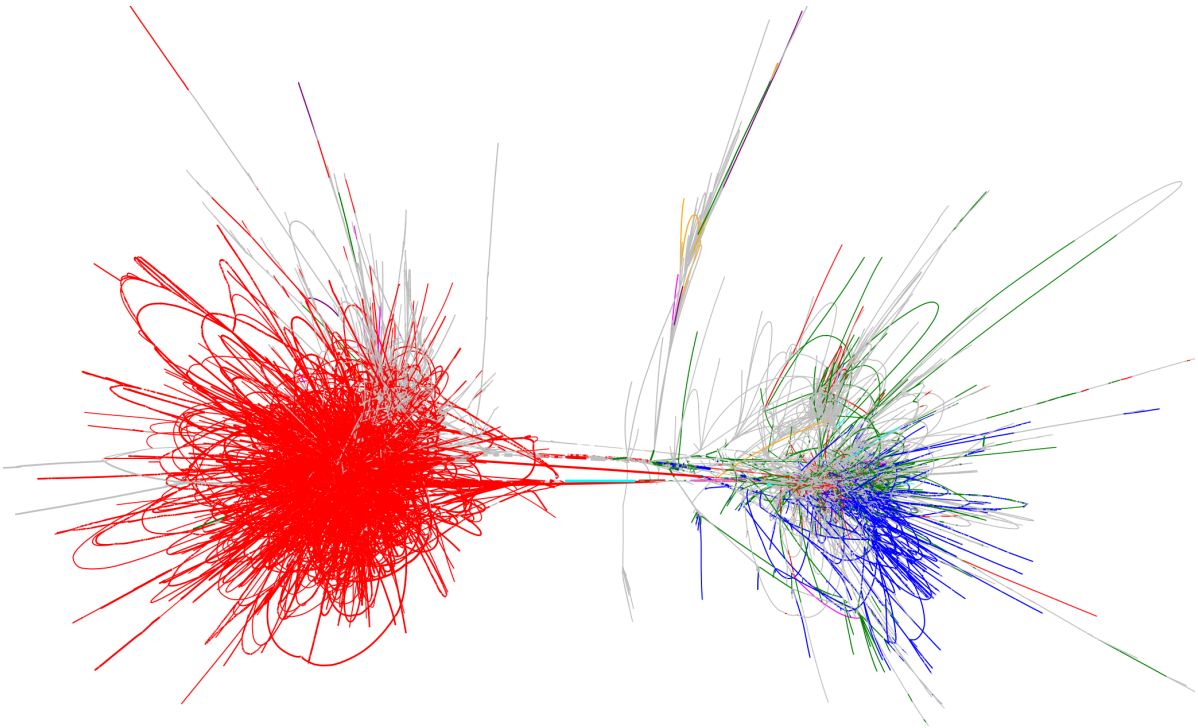


Fig. 3.21 The largest components in a human gut microbiome assembly graph. Node width shows coverage in the assembly. Nodes that match to RefSeq genomes for nine given phyla are colored in the graph as described here with abundances: *Bacteroides* (red) 34827, *Clostridium* (green) 9521, *Faecalibacterium* (blue) 5921, *Bifidobacterium* (magenta) 373, *Enterococcus* (cyan) 73, *Enterobacter* (orange) 25, *Klebsiella* (purple) 21, *Escherichia* (yellow) 17, and *Enterobacteriaceae* (brown) 2.

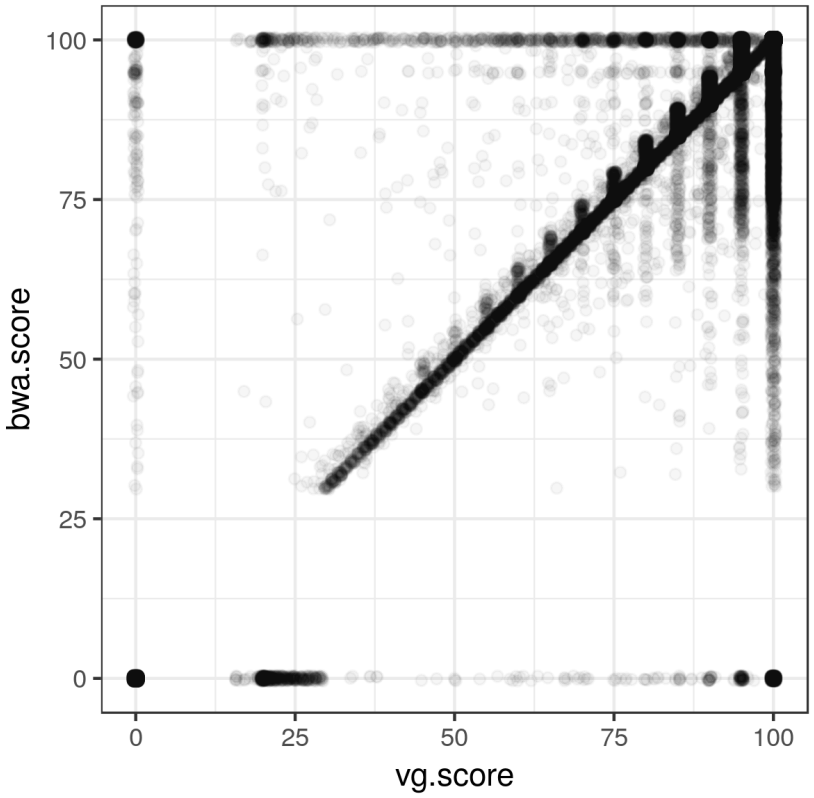


Fig. 3.22 Comparing alignment performance between `vg map` and `bwa mem` to the human gut microbiome assembly graph or its contigs.

below the diagonal in vg's favor (figure 3.22). It is possible that an assembly designed to capture more variation in the strains might benefit more greatly from the use of vg. A number of issues appear when inspecting these results, for instance that vg does not align some reads that bwa does. This could represent a bug in the aligner or the effect of pruning on the gut metagenome graph's GCSA2 index. We can conclude that while this method shows promise in a variety of metagenomic applications, improvements may be required for it to yield benefits in some contexts.

3.6 RNA-seq

As described in section 1.4.2.4, graphs have been used as a description of transcriptomes. A transcript graph can encode alternatively spliced transcripts in a DAG. By recording the transcripts as paths in this graph we can build a coherent annotation describing all the known transcripts and their sequences, and by indexing the graph with vg we can align RNA-seq reads directly to it. To build a splicing graph in vg requires a linear reference and a gene model encoded in GFF, and uses the *edit* function to augment the graph with alignments representing each gene annotation, thus incorporating edges for each spliced intron (section 2.2.3).

3.6.1 Yeast transcriptome graph

An obvious application of vg is to the direct alignment of reads to a splicing graph. I have only begun to explore this at the end of my studentship. Once again, I started with yeast, which has only a small number (∼280) of spliced genes. I built a gene model graph (SGD+CDS) using the yeast reference genome from the Saccharomyces Genome Database (SGD) and its gene annotations[17]. At 8.8MB, the serialized SGD+CDS vg graph itself is only slightly larger than the SGD linear reference graph, which is 7.3MB. However, its xg index is much larger, 642MB, in contrast to the linear index at 38MB. This index suffers a large penalty for the ∼5000 embedded transcript annotations, as each becomes a full reference path, usable in alignment for positional inference. To verify that the splicing graph would allow reads to align full length across splice junctions, I aligned 100k reads from a yeast mRNA sequencing data set[18].

The low rate of alternative splicing in yeast means that only a handful of reads map across splice junctions encoded in the graph, but those that do now align full length,

[17] I used the 20150113 release from https://downloads.yeastgenome.org/sequence/S288C_reference/genome_releases/S288C_reference_genome_R64-2-1_20150113.tgz.

[18] SRR2069949 in SRA, https://www.ncbi.nlm.nih.gov/sra/?term=SRR2069949

as seen in figure 3.23. The difference in aggregate alignment identity is small (0.91 vs 0.912) but marginally significant (two-tailed T-test $p = 0.03927$). This provides a basic proof of principle that the splicing graph can be represented as a variation graph and that `vg` can align reads to it without any particular RNA-specific considerations. Due to time constraints I was unable to further explore this topic. One major concern was the runtime and memory costs to construct the `xg` index. These suggest that it may not be possible to build a splicing graph for a large genome with the transcripts encoded as positional paths, which further implies that improvements in the approximate positional metric in the graph will be required to scale RNA-seq alignment with `vg`. Further, the results shown in figure 3.23 indicate that there may be a positional offset bug in the conversion of the GFF files representing the transcriptome into the graph. In summary, future work will be required to explore the use of `vg` for RNA-seq data.

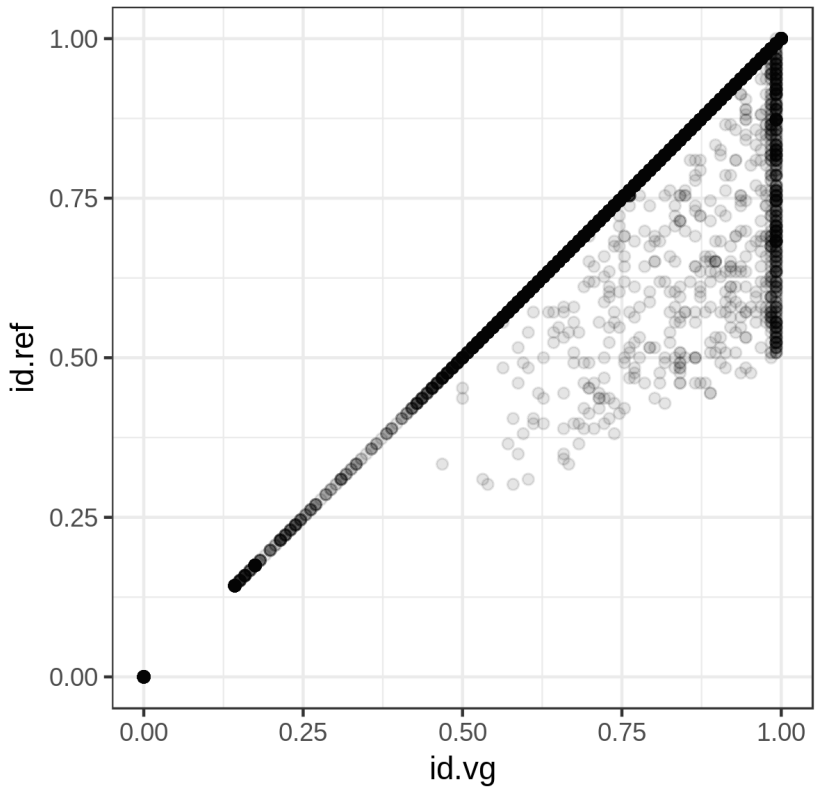


Fig. 3.23 Comparing alignment identity between `vg map` on the linear SGD reference (y-axis) and the SGD reference augmented by the transcripts in the annotated CDS (x-axis).

Chapter 4

Conclusions

Genomics is driven by comparison. It is rare that we have reason to consider a single genome in isolation. We apply alignment algorithms to infer the sequence-level relationship between genomes, or between additional sequence data and genomes. As data scales have increased, we have required incomplete methods to determine these relationships among many individuals. Contemporary resequencing techniques focus on the placement of new sequencing information into a reference system which is typically linear and representative of only a single copy of each genomic locus. A pangenomic reference system allows us to represent multiple versions of each locus, but until recently such techniques have been difficult to apply at scales commonly reached in current analyses.

I propose the use of variation graphs as reference systems in resequencing. These path-labeled, bidirectional DNA sequence graphs allow us to represent collections of genomes in a single, coherent structure which fully capture the sequences and variation between them. They support the direct representation of all kinds of genomic variation. By building a software system supporting the construction, manipulation, indexing, and alignment of new read sets and genomes to variation graphs, I am able to show that this model reduces bias towards the reference in alignment in a wide array of genomic contexts. These methods achieve a level of performance that will make them usable for large-scale resequencing analyses. As I have shown, their modular implementation, based around a handful of core data models, enables the rapid construction of novel graph-based analysis processes that provide conceptual unity to alignment, assembly, and variant calling. Although other methods for aligning sequences against pangenome data structures exist, `vg` is the first set of tools that does so in a completely coherent manner against arbitrary bidirectional sequence graphs. This is also the first framework to provide graph based analogs of many of the data types standardly used in resequencing.

In addition to developing methods to support alignment to variation graphs, we have explored a variety of related analyses. We developed new techniques for visualizing variation graphs that will help to build the genome browsers necessary to navigate data placed in the context of the graph. To record and interface with annotations embedded in a variation graphs, we have linked the variation graph data model to RDF. To simplify their construction, I provide a method to losslessly induce variation graphs from a set of aligned sequences. We built systems that allow for the efficient summarization of alignment data sets against variation graph. And we worked on methods to support genotyping known and novel variation in graphs. Throughout my work I have supported and worked with a growing group of researchers focused on these techniques, collaborating in the development of graph sequence and haplotype indexing techniques, the evaluation of diverse variation graph models, and the study of ancient DNA using variation graphs.

Graphical models are often regarded with apprehension by members of the bioinformatics community who are accustomed to working with linear reference genomes. I show that arbitrary variation graphs may be consistently linearized for visualization and analysis. Variation graphs built from related sequences tend to have a regional linear property despite the frequent presence of large scale variation. I show that this holds for graphs constructed from a variety of sources using alignment or assembly techniques. They retain relatively linear structures locally, and as such can be used for efficient alignment. The linearization of the graph suggests a projection of sequencing information in the graph into a basis vector space defined by the graph itself. Such an approach may greatly simplify genomic analyses by removing the complicated variant calling step. If the variation we want to consider is already embedded in the graph, we do not need to genotype novel variation or engage in filtering our results. As variation is now embedded in the graph, we can perhaps avoid variant calling altogether where downstream it is possible to work with a normalized coverage model across this graph basis vector. Doing so practically will require the development of techniques that can scale genetic analyses to the large matrix representations implied by such maps.

It is not clear how to build the best graph for a given analysis context. The results I present show that the addition of variation to a graph does not necessarily improve alignment performance in all contexts. Additional variation increases graph complexity, and this can make results more ambiguous. One important step is likely to be the use of haplotype information at the level of alignment. Ongoing work suggests that doing so may mitigate scaling issues that will occur as we build graphs from tens and hundreds of thousands of genomes, but there is still much work to be done. We can expect that, with time, practices will arise that capture the ideal patterns for constructing variation

graphs. I found a number of potential input sources unreliable in their current form, and I hope to explore them as variation graph analysis techniques mature. Progressive and multiple whole genome alignment algorithms look to be the most promising way to merge haplotype resolved genome assemblies that new genome inference technologies are enabling. However, as they have difficulty scaling to more than single human chromosomes, I am interested in exploring ways of building variation graphs from networks of pairwise alignments. Given improvement of the input alignment process, this technique could also serve as a scalable way to construct variation graphs in any context where collections of sequenced genomes exist.

I believe that reference genomes should be replaced with pangenomic structures. This is the clearest way to resolve representational issues that arise as we collect large collections of genomes in the species we examine. The variation graph is a natural model with which to do this. Its adoption is now a social as well as a technical question. Can the community generate a unified set of data structures that encapsulate the ideas I have presented here? Large distributed projects like the 1000GP gave rise to the current generation of genomic data formats. It seems natural that the next, graphical, pangenomic phase will require the same. At present, it is not clear what project might support this. Top-down approaches like that presented by the GA4GH have not proven as capable of promoting standards as analysis-oriented projects like the 1000GP, although they have served a coordinating role for the community of researchers interested in these topics. One obvious target for the widespread introduction of variation graph data models would be in the generation of a new reference genome system based on a collection of fully-resolved genomes. Motivation for such an advance increases as evidence mounts that a substantial and important fraction of genetic variation is neither small nor simple. I am hopeful that my work may support such an effort, and that the ideas which arise therein may follow at least in part from the generic graphical pangenomic models I have proposed and demonstrated here.

References

[1] The 1000 Genomes Project Participants. An integrated map of genetic variation from 1,092 human genomes. *Nature*, 491(7422):56–65, 2012.

[2] Mohamed Ibrahim Abouelhoda, Stefan Kurtz, and Enno Ohlebusch. Replacing suffix trees with enhanced suffix arrays. *Journal of Discrete Algorithms*, 2(1):53–86, 2004.

[3] JM Adams, PGN Jeppesen, F Sanger, and BG Barrell. Nucleotide sequence from the coat protein cistron of r17 bacteriophage RNA. *Nature*, 223(5210):1009–1014, 1969.

[4] Cornelis A Albers, Gerton Lunter, Daniel G MacArthur, Gilean McVean, Willem H Ouwehand, and Richard Durbin. Dindel: accurate indel calls from short-read data. *Genome Research*, 21(6):961–973, 2011.

[5] Morten E Allentoft, Martin Sikora, Karl-Göran Sjögren, Simon Rasmussen, Morten Rasmussen, Jesper Stenderup, Peter B Damgaard, Hannes Schroeder, Torbjörn Ahlström, Lasse Vinner, et al. Population genomics of bronze age Eurasia. *Nature*, 522(7555):167–172, 2015.

[6] Manuel Allhoff, Alexander Schönhuth, Marcel Martin, Ivan G Costa, Sven Rahmann, and Tobias Marschall. Discovering motifs that induce sequencing errors. *BMC Bioinformatics*, 14(5):S1, 2013.

[7] Carlos Alonso-Blanco, Jorge Andrade, Claude Becker, Felix Bemm, Joy Bergelson, Karsten M Borgwardt, Jun Cao, Eunyoung Chae, Todd M Dezwaan, Wei Ding, et al. 1,135 genomes reveal the global pattern of polymorphism in Arabidopsis thaliana. *Cell*, 166(2):481–491, 2016.

[8] Stephen F Altschul and Bruce W Erickson. Optimal sequence alignment using affine gap costs. *Bulletin of Mathematical Biology*, 48(5-6):603–616, 1986.

[9] Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3): 403–410, 1990.

[10] Alberto Apostolico. The myriad virtues of subword trees. In *Combinatorial algorithms on words*, pages 85–96. Springer, 1985.

[11] Oswald T Avery, Colin M MacLeod, and Maclyn McCarty. Studies on the chemical nature of the substance inducing transformation of pneumococcal types: induction of transformation by a desoxyribonucleic acid fraction isolated from pneumococcus type iii. *Journal of Experimental Medicine*, 79(2):137–158, 1944.

[12] Shankar Balasubramanian, David Klenerman, and David Bentley. Arrayed biomolecules and their use in sequencing, 2004. US Patent 6,787,308.

[13] Anton Bankevich, Sergey Nurk, Dmitry Antipov, Alexey A Gurevich, Mikhail Dvorkin, Alexander S Kulikov, Valery M Lesin, Sergey I Nikolenko, Son Pham, Andrey D Prjibelski, et al. SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing. *Journal of Computational Biology*, 19(5): 455–477, 2012.

[14] Markus J Bauer, Anthony J Cox, and Giovanna Rosone. Lightweight algorithms for constructing and inverting the BWT of string collections. *Theoretical Computer Science*, 483:134–148, 2013.

[15] Richard Bellman. On the theory of dynamic programming. *Proceedings of the National Academy of Sciences*, 38(8):716–719, 1952.

[16] 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.

[17] Anders Bergström, Jared T Simpson, Francisco Salinas, Benjamin Barré, Leopold Parts, Amin Zia, Alex N Nguyen Ba, Alan M Moses, Edward J Louis, Ville Mustonen, et al. A high-definition view of functional genetic variation from natural yeast genomes. *Molecular Biology and Evolution*, 31(4):872–888, 2014.

[18] Derek M Bickhart, Benjamin D Rosen, Sergey Koren, Brian L Sayre, Alex R Hastie, Saki Chan, Joyce Lee, Ernest T Lam, Ivan Liachko, Shawn T Sullivan, et al. Single-molecule sequencing and chromatin conformation capture enable de novo reference assembly of the domestic goat genome. *Nature Genetics*, 49(4):643–650, 2017.

[19] Etienne Birmelé, Pierluigi Crescenzi, Rui Ferreira, Roberto Grossi, Vincent Lacroix, Andrea Marino, Nadia Pisanti, Gustavo Sacomoto, and Marie-France Sagot. Efficient bubble enumeration in directed graphs. In *International Symposium on String Processing and Information Retrieval*, pages 118–129. Springer, 2012.

[20] Inanç Birol, Shaun D Jackman, Cydney B Nielsen, Jenny Q Qian, Richard Varhol, Greg Stazyk, Ryan D Morin, Yongjun Zhao, Martin Hirst, Jacqueline E Schein, et al. De novo transcriptome assembly with ABySS. *Bioinformatics*, 25(21):2872–2877, 2009.

[21] Mathieu Blanchette, W James Kent, Cathy Riemer, Laura Elnitski, Arian FA Smit, Krishna M Roskin, Robert Baertsch, Kate Rosenbloom, Hiram Clawson, Eric D Green, et al. Aligning multiple genomic sequences with the threaded blockset aligner. *Genome Research*, 14(4):708–715, 2004.

[22] Nathan Blow. Decoding the unsequenceable, 2015. URL https://www.future-science.com/doi/pdf/10.2144/000114252.

[23] Alexander Bowe, Taku Onodera, Kunihiko Sadakane, and Tetsuo Shibuya. Succinct de Bruijn graphs. In *International Workshop on Algorithms in Bioinformatics*, pages 225–235. Springer, 2012.

[24] Ljiljana Brankovic, Costas S Iliopoulos, Ritu Kundu, Manal Mohamed, Solon P Pissis, and Fatima Vayani. Linear-time superbubble identification algorithm for genome assembly. *Theoretical Computer Science*, 609:374–383, 2016.

[25] Nicolas L Bray, Harold Pimentel, Páll Melsted, and Lior Pachter. Near-optimal probabilistic RNA-seq quantification. *Nature Biotechnology*, 34(5):525–527, 2016.

[26] Adrian W Briggs, Udo Stenzel, Matthias Meyer, Johannes Krause, Martin Kircher, and Svante Pääbo. Removal of deaminated cytosines and detection of in vivo methylation in ancient DNA. *Nucleic Acids Research*, 38(6):e87–e87, 2009.

[27] S. R. Browning and B. L. Browning. Rapid and accurate haplotype phasing and missing-data inference for whole-genome association studies by use of localized haplotype clustering. *Am. J. Hum. Genetics*, 81(5):1084–1097, 2007.

[28] Michael Burrows and David J Wheeler. A block-sorting lossless data compression algorithm. *Digital Equipment Corporation technical reports*, 124, 1994.

[29] 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(5):810–820, 2008.

[30] Bruno Canard and Robert S Sarfati. DNA polymerase fluorescent substrates with reversible 3'-tags. *Gene*, 148(1):1–6, 1994.

[31] Jun Cao, Korbinian Schneeberger, Stephan Ossowski, Torsten Günther, Sebastian Bender, Joffrey Fitz, Daniel Koenig, Christa Lanz, Oliver Stegle, Christoph Lippert, et al. Whole-genome sequencing of multiple Arabidopsis thaliana populations. *Nature Genetics*, 43(10):956–963, 2011.

[32] Humberto Carrillo and David Lipman. The multiple sequence alignment problem in biology. *SIAM Journal on Applied Mathematics*, 48(5):1073–1082, 1988.

[33] John Castiblanco. *A primer on current and common sequencing technologies.* El Rosario University Press, 2013. URL https://www.ncbi.nlm.nih.gov/books/NBK459463/.

[34] Mark JP Chaisson, Ashley D Sanders, Xuefang Zhao, Ankit Malhotra, David Porubsky, Tobias Rausch, Eugene J Gardner, Oscar Rodriguez, Li Guo, Ryan L Collins, et al. Multi-platform discovery of haplotype-resolved structural variation in human genomes. *bioRxiv:193144*, 2018.

[35] Mahul Chakraborty, Nicholas W VanKuren, Roy Zhao, Xinwen Zhang, Shannon Kalsow, and JJ Emerson. Hidden genetic variation shapes the structure of functional elements in drosophila. *Nature Genetics*, 50(1):20–25, 2018.

[36] Danny Challis, Lilian Antunes, Erik Garrison, Eric Banks, Uday S Evani, Donna Muzny, Ryan Poplin, Richard A Gibbs, Gabor Marth, and Fuli Yu. The distribution and mutagenesis of short coding indels from 1,128 whole exomes. *BMC Genomics*, 16(1):143, 2015.

[37] Zheng Chang, Guojun Li, Juntao Liu, Yu Zhang, Cody Ashby, Deli Liu, Carole L Cramer, and Xiuzhen Huang. Bridger: a new framework for de novo transcriptome assembly using RNA-seq data. *Genome Biology*, 16(1):30, 2015.

[38] J Michael Cherry, Caroline Adler, Catherine Ball, Stephen A Chervitz, Selina S Dwight, Erich T Hester, Yankai Jia, Gail Juvik, TaiYun Roe, Mark Schroeder, et al. SGD: Saccharomyces genome database. *Nucleic Acids Research*, 26(1):73–79, 1998.

[39] Rayan Chikhi and Guillaume Rizk. Space-efficient and exact de Bruijn graph representation based on a bloom filter. *Algorithms for Molecular Biology*, 8(1):22, 2013.

[40] Rayan Chikhi, Antoine Limasset, and Paul Medvedev. Compacting de Bruijn graphs from sequencing data quickly and in low memory. *Bioinformatics*, 32(12): i201–i208, 2016.

[41] Chen-Shan Chin, Paul Peluso, Fritz J Sedlazeck, Maria Nattestad, Gregory T Concepcion, Alicia Clum, Christopher Dunn, Ronan O'Malley, Rosa Figueroa-Balderas, Abraham Morales-Cruz, et al. Phased diploid genome assembly with single-molecule real-time sequencing. *Nature Methods*, 13(12):1050–1054, 2016.

[42] Deanna M Church. Genomes for all. *Nature Biotechnology*, 36(9):815–816, 2018.

[43] Richard M Clark, Gabriele Schweikert, Christopher Toomajian, Stephan Ossowski, Georg Zeller, Paul Shinn, Norman Warthmann, Tina T Hu, Glenn Fu, David A Hinds, et al. Common sequence polymorphisms shaping genetic diversity in Arabidopsis thaliana. *Science*, 317(5836):338–342, 2007.

[44] Jonathan Cohen. Graph twiddling in a mapreduce world. *Computing in Science & Engineering*, 11(4):29–41, 2009.

[45] 1000 Genomes Project Consortium et al. A global reference for human genetic variation. *Nature*, 526(7571):68–74, 2015.

[46] Computational Pan-Genomics Consortium. Computational pan-genomics: status, promises and challenges. *Briefings in Bioinformatics*, 19(1):118–135, 2018.

[47] ENCODE Project Consortium et al. An integrated encyclopedia of DNA elements in the human genome. *Nature*, 489(7414):57–74, 2012.

[48] International Human Genome Sequencing Consortium et al. Initial sequencing and analysis of the human genome. *Nature*, 409(6822):860–921, 2001.

[49] UK10K Consortium et al. The UK10K project identifies rare variants in health and disease. *Nature*, 526(7571):82–90, 2015.

[50] Francis HC Crick. On protein synthesis. *Symposia of the Society for Experimental Biology*, 12:138–163, 1958.

[51] Francis HC Crick. Central dogma of molecular biology. *Nature*, 227(5258):561–563, 1970.

[52] Jesse Dabney, Matthias Meyer, and Svante Pääbo. Ancient DNA damage. *Cold Spring Harbor Perspectives in Biology*, page a012567, 2013.

[53] Petr Danecek, Adam Auton, Goncalo Abecasis, Cornelis A Albers, Eric Banks, Mark A DePristo, Robert E Handsaker, Gerton Lunter, Gabor T Marth, Stephen T Sherry, et al. The variant call format and VCFtools. *Bioinformatics*, 27(15): 2156–2158, 2011.

[54] Matei David, Lewis Jonathan Dursi, Delia Yao, Paul C Boutros, and Jared T Simpson. Nanocall: an open source basecaller for oxford nanopore sequencing data. *Bioinformatics*, 33(1):49–55, 2016.

[55] Peter de Barros Damgaard, Rui Martiniano, Jack Kamm, J Víctor Moreno-Mayar, Guus Kroonen, Michaël Peyrot, Gojko Barjamovic, Simon Rasmussen, Claus Zacho, Nurbol Baimukhanov, et al. The first horse herders and the impact of early Bronze Age steppe expansions into Asia. *Science*, 360(6396):1422–1442, 2018.

[56] Nicolaas Govert De Bruijn. A combinatorial problem. *Koninklijke Nederlandse Akademie v. Wetenschappen*, 49(49):758–764, 1946.

[57] Daniel Aguirre de Cárcer, Alberto López-Bueno, David A Pearce, and Antonio Alcamí. Biodiversity and distribution of polar freshwater DNA viruses. *Science Advances*, 1(5):e1400127, 2015.

[58] David Deamer, Mark Akeson, and Daniel Branton. Three decades of nanopore sequencing. *Nature Biotechnology*, 34(5):518–524, 2016.

[59] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[60] O. Delaneau, J. Marchini, and J. F. Zagury. A linear complexity phasing method for thousands of genomes. *Nature Methods*, 9(2):179–181, 2012.

[61] Arthur L Delcher, Simon Kasif, Robert D Fleischmann, Jeremy Peterson, Owen White, and Steven L Salzberg. Alignment of whole genomes. *Nucleic Acids Research*, 27(11):2369–2376, 1999.

[62] Mark A DePristo, Eric Banks, Ryan Poplin, Kiran V Garimella, Jared R Maguire, Christopher Hartl, Anthony A Philippakis, Guillermo Del Angel, Manuel A Rivas, Matt Hanna, et al. A framework for variation discovery and genotyping using next-generation DNA sequencing data. *Nature Genetics*, 43(5):491–498, 2011.

[63] Alexander Dilthey, Charles Cox, Zamin Iqbal, Matthew R Nelson, and Gil McVean. Improved genome inference in the MHC using a population reference graph. *Nature Genetics*, 47(6):682–688, 2015.

[64] Olga Dudchenko, Sanjit S Batra, Arina D Omer, Sarah K Nyquist, Marie Hoeger, Neva C Durand, Muhammad S Shamim, Ido Machol, Eric S Lander, Aviva Presser Aiden, et al. De novo assembly of the Aedes aegypti genome using Hi-C yields chromosome-length scaffolds. *Science*, 356(6333):92–95, 2017.

[65] Richard Durbin. Efficient haplotype matching and storage using the positional Burrows–Wheeler transform (PBWT). *Bioinformatics*, 30(9):1266–1272, 2014.

[66] Richard Durbin, Sean R Eddy, Anders Krogh, and Graeme Mitchison. *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge University Press, 1998.

[67] Dent A Earl, Keith Bradnam, John St John, Aaron Darling, Dawei Lin, Joseph Faas, Hung On Ken Yu, Buffalo Vince, Daniel R Zerbino, Mark Diekhans, et al. Assemblathon 1: a competitive assessment of de novo short read assembly methods. *Genome Research*, 21(12):2224–2241, 2011.

[68] Sarah G Earle, Chieh-Hsi Wu, Jane Charlesworth, Nicole Stoesser, N Claire Gordon, Timothy M Walker, Chris CA Spencer, Zamin Iqbal, David A Clifton, Katie L Hopkins, et al. Identifying lineage effects when controlling for population structure improves power in bacterial association studies. *Nature Microbiology*, 1(5):16041, 2016.

[69] MA Eberle, M Kallberg, HY Chuang, P Tedder, S Humphray, D Bentley, and E Margulies. Platinum genomes: A systematic assessment of variant accuracy using a large family pedigree. In *60th Annual Meeting of The American Society of Human Genetics*, pages 22–26, 2013.

[70] Robert A Edwards and Forest Rohwer. Viral metagenomics. *Nature Reviews Microbiology*, 3(6):504–510, 2005.

[71] Hannes P Eggertsson, Hakon Jonsson, Snaedis Kristmundsdottir, Eirikur Hjartarson, Birte Kehr, Gisli Masson, Florian Zink, Kristjan E Hjorleifsson, Aslaug Jonasdottir, Adalbjorg Jonasdottir, et al. Graphtyper enables population-scale genotyping using pangenome graphs. *Nature Genetics*, 49(11):1654–1660, 2017.

[72] John Eid, Adrian Fehr, Jeremy Gray, Khai Luong, John Lyle, Geoff Otto, Paul Peluso, David Rank, Primo Baybayan, Brad Bettman, et al. Real-time DNA sequencing from single polymerase molecules. *Science*, 323(5910):133–138, 2009.

[73] John Ellson, Emden Gansner, Lefteris Koutsofios, Stephen C North, and Gordon Woodhull. Graphviz—open source graph drawing tools. In *International Symposium on Graph Drawing*, pages 483–484. Springer, 2001.

[74] Ester Falconer, Mark Hills, Ulrike Naumann, Steven SS Poon, Elizabeth A Chavez, Ashley D Sanders, Yongjun Zhao, Martin Hirst, and Peter M Lansdorp. DNA template strand sequencing of single-cells maps genomic rearrangements at high resolution. *Nature Methods*, 9(11):1107–1112, 2012.

[75] Xian Fan, Mark Chaisson, Luay Nakhleh, and Ken Chen. HySA: A Hybrid Structural variant Assembly approach using next generation and single-molecule sequencing technologies. *Genome Research*, 27(5):793–800, 2017.

[76] Michael Farrar. Striped Smith–Waterman speeds database searches six times over other SIMD implementations. *Bioinformatics*, 23(2):156–161, 2007.

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

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

[79] Paolo Ferragina and Giovanni Manzini. An experimental study of an opportunistic index. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete Algorithms*, pages 269–278. Society for Industrial and Applied Mathematics, 2001.

[80] Paolo Ferragina and Giovanni Manzini. Indexing compressed text. *Journal of the ACM (JACM)*, 52(4):552–581, 2005.

[81] Paolo Ferragina, Giovanni Manzini, Veli Mäkinen, and Gonzalo Navarro. An alphabet-friendly FM-index. In *String Processing and Information Retrieval*, pages 150–160. Springer, 2004.

[82] Paolo Ferragina, Fabrizio Luccio, Giovanni Manzini, and S Muthukrishnan. Structuring labeled trees for optimal succinctness, and beyond. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*, pages 184–193. IEEE, 2005.

[83] Paolo Ferragina, Fabrizio Luccio, Giovanni Manzini, and Shan Muthukrishnan. Compressing and indexing labeled trees, with applications. *Journal of the ACM (JACM)*, 57(1):4, 2009.

[84] Walter Fiers, Roland Contreras, Fred Duerinck, Guy Haegeman, Dirk Iserentant, Jozef Merregaert, W Min Jou, Francis Molemans, Alex Raeymaekers, A Van den Berghe, et al. Complete nucleotide sequence of bacteriophage ms2 RNA: primary and secondary structure of the replicase gene. *Nature*, 260(5551):500–507, 1976.

[85] Walter M Fitch. An improved method of testing for evolutionary homology. *Journal of Molecular Biology*, 16(1):9–16, 1966.

[86] Robert D Fleischmann, Mark D Adams, Owen White, Rebecca A Clayton, Ewen F Kirkness, Anthony R Kerlavage, Carol J Bult, Jean-Francois Tomb, Brian A Dougherty, Joseph M Merrick, et al. Whole-genome random sequencing and assembly of haemophilus influenzae Rd. *Science*, 269(5223):496–512, 1995.

[87] Travis Gagie, Giovanni Manzini, and Jouni Sirén. Wheeler graphs: A framework for BWT-based data structures. *Theoretical Computer Science*, 698:67–78, 2017.

[88] Emden R Gansner and Stephen C North. An open graph visualization system and its applications to software engineering. *Software: practice and experience*, 30(11): 1203–1233, 2000.

[89] Emden R Gansner, Eleftherios Koutsofios, Stephen C North, and K-P Vo. A technique for drawing directed graphs. *IEEE Transactions on Software Engineering*, 19(3):214–230, 1993.

[90] Richard C Gardner, Alan J Howarth, Peter Hahn, Marianne Brown-Luedi, Robert J Shepherd, and Joachim Messing. The complete nucleotide sequence of an infectious clone of cauliflower mosaic virus by M13mp7 shotgun sequencing. *Nucleic Acids Research*, 9(12):2871–2888, 1981.

[91] Erik Garrison and Gabor Marth. Haplotype-based variant detection from short-read sequencing. *arXiv:1207.3907*, 2012.

[92] Erik Garrison, Jouni Sirén, Adam M Novak, Glenn Hickey, Jordan M Eizenga, Eric T Dawson, William Jones, Shilpa Garg, Charles Markello, Michael F Lin, et al. Variation graph toolkit improves read mapping by representing genetic variation in the reference. *Nature Biotechnology*, 36(9):875–879, 2018.

[93] Jay Ghurye, Arang Rhie, Brian P Walenz, Anthony Schmitt, Siddarth Selvaraj, Mihai Pop, Adam M Phillippy, and Sergey Koren. Integrating Hi-C links with assembly graphs for chromosome-scale assembly. *bioRxiv:261149*, 2018.

[94] André Goffeau, Bart G Barrell, Howard Bussey, RW Davis, Bernard Dujon, Heinz Feldmann, Francis Galibert, JD Hoheisel, Cr Jacq, Michael Johnston, et al. Life with 6000 genes. *Science*, 274(5287):546–567, 1996.

[95] Simon Gog, Timo Beller, Alistair Moffat, and Matthias Petri. From theory to practice: Plug and play with succinct data structures. In *13th International Symposium on Experimental Algorithms, (SEA 2014)*, pages 326–337, 2014.

[96] David Gordon, Chris Abajian, and Phil Green. Consed: a graphical tool for sequence finishing. *Genome Research*, 8(3):195–202, 1998.

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

[98] Osamu Gotoh. Optimal sequence alignment allowing for long gaps. *Bulletin of Mathematical Biology*, 52(3):359–373, 1990.

[99] Manfred G Grabherr, Brian J Haas, Moran Yassour, Joshua Z Levin, Dawn A Thompson, Ido Amit, Xian Adiconis, Lin Fan, Raktima Raychowdhury, Qiandong Zeng, et al. Full-length transcriptome assembly from RNA-seq data without a reference genome. *Nature Biotechnology*, 29(7):644–652, 2011.

[100] Catherine Grasso and Christopher Lee. Combining partial order alignment and progressive multiple sequence alignment increases alignment speed and scalability to very large alignment problems. *Bioinformatics*, 20(10):1546–1556, 2004.

[101] Richard E Green, Johannes Krause, Adrian W Briggs, Tomislav Maricic, Udo Stenzel, Martin Kircher, Nick Patterson, Heng Li, Weiwei Zhai, Markus Hsi-Yang Fritz, et al. A draft sequence of the Neandertal genome. *Science*, 328(5979):710–722, 2010.

[102] Stuart J. Green, Reigh P. Monreal, Alan T. White, Thomas G. Bayer, Stuart J. Green, Reigh P. Monreal, Alan T. White, Thomas G. Bayer, Yasmin D. Arquiza, Alan T. White, Stuart J. Green, R. Buenaflor, and Jr. Nd Y. D. Arquiza. Phrap documentation, 1999.

[103] Roberto Grossi and Jeffrey Scott Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *SIAM Journal on Computing*, 35(2):378–407, 2005.

[104] Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. High-order entropy-compressed text indexes. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 841–850. Society for Industrial and Applied Mathematics, 2003.

[105] Sarah Guthrie, Abram Connelly, Peter Amstutz, Adam F Berrey, Nicolas Cesar, Jiahua Chen, Radhika Chippada, Tom Clegg, Bryan Cosca, Jiayong Li, et al. Tiling the genome into consistently named subsequences enables precision medicine and machine learning with millions of complex individual data-sets. Technical report, PeerJ PrePrints, 2015. URL https://peerj.com/preprints/1426/.

[106] Ronald Haentjens Dekker, Dirk Van Hulle, Gregor Middell, Vincent Neyt, and Joris Van Zundert. Computer-supported collation of modern manuscripts: Collatex and the beckett digital manuscript project. *Digital Scholarship in the Humanities*, 30(3):452–470, 2014.

[107] Robert S Harris. *Improved Pairwise Alignment of Genomic DNA*. PhD thesis, The Pennsylvania State University, 2007.

[108] Timothy D Harris, Phillip R Buzby, Hazen Babcock, Eric Beer, Jayson Bowers, Ido Braslavsky, Marie Causey, Jennifer Colonell, James DiMeo, J William Efcavitch, et al. Single-molecule DNA sequencing of a viral genome. *Science*, 320(5872): 106–109, 2008.

[109] Steffen Heber, Max Alekseyev, Sing-Hoi Sze, Haixu Tang, and Pavel A Pevzner. Splicing graphs and EST assembly problem. *Bioinformatics*, 18(suppl_1):S181–S188, 2002.

[110] Desmond G Higgins and Paul M Sharp. Clustal: a package for performing multiple sequence alignment on a microcomputer. *Gene*, 73(1):237–244, 1988.

[111] Jonathon T Hill, Bradley L Demarest, Brent W Bisgrove, Yi-Chu Su, Megan Smith, and H Joseph Yost. Poly peak parser: Method and software for identification of unknown indels using sanger sequencing of polymerase chain reaction products. *Developmental Dynamics*, 243(12):1632–1636, 2014.

[112] B. Howie, J. Marchini, and M. Stephens. Genotype imputation with thousands of genomes. *G3 (Bethesda)*, 1(6):457–470, 2011.

[113] Lin Huang, Victoria Popic, and Serafim Batzoglou. Short read alignment with populations of genomes. *Bioinformatics*, 29(13):i361–i370, 2013.

[114] Xiaoqiu Huang. A contig assembly program based on sensitive detection of fragment overlaps. *Genomics*, 14(1):18–25, 1992.

[115] Yao-Ting Huang and Chen-Fu Liao. Integration of string and de Bruijn graphs for genome assembly. *Bioinformatics*, 32(9):1301–1307, 2016.

[116] Julian Huxley. *Evolution: the modern synthesis*. George Allen and Unwin, 1942.

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

[118] Z. Iqbal, M. Caccamo, I. Turner, P. Flicek, and G. McVean. De novo assembly and genotyping of variants using colored de Bruijn graphs. *Nature Genetics*, 44(2): 226–232, 2012.

[119] Z. Iqbal, I. Turner, and G. McVean. High-throughput microbial population genomics using the Cortex variation assembler. *Bioinformatics*, 29(2):275–276, 2013.

[120] Zamin Iqbal, Mario Caccamo, Isaac Turner, Paul Flicek, and Gil McVean. De novo assembly and genotyping of variants using colored de Bruijn graphs. *Nature Genetics*, 44(2):226–232, 2012.

[121] Marten Jäger, Max Schubach, Tomasz Zemojtel, Knut Reinert, Deanna M Church, and Peter N Robinson. Alternate-locus aware variant calling in whole genome sequencing. *Genome Medicine*, 8(1):130, 2016.

[122] Miten Jain, Sergey Koren, Karen H Miga, Josh Quick, Arthur C Rand, Thomas A Sasani, John R Tyson, Andrew D Beggs, Alexander T Dilthey, Ian T Fiddes, et al. Nanopore sequencing and assembly of a human genome with ultra-long reads. *Nature Biotechnology*, 36(4):338–345, 2018.

[123] Christine Jandrasits, Piotr W Dabrowski, Stephan Fuchs, and Bernhard Y Renard. seq-seq-pan: Building a computational pan-genome data structure on whole genome alignment. *BMC Genomics*, 19(1):47, 2018.

[124] Stefan Jänicke, Annette Geßner, Greta Franzini, Melissa Terras, Simon Mahony, and Gerik Scheuermann. Traviz: A visualization for variant graphs. *Digital Scholarship in the Humanities*, 30(suppl_1):i83–i99, 2015.

[125] W Min Jou, G Haegeman, M Ysebaert, and W Fiers. Nucleotide sequence of the gene coding for the bacteriophage MS2 coat protein. *Nature*, 237(5350):82–88, 1972.

[126] Arthur B Kahn. Topological sorting of large networks. *Communications of the ACM*, 5(11):558–562, 1962.

[127] Rolf G Karlsson and Patricio V Poblete. An $O(m \log \log D)$ algorithm for shortest paths. *Discrete Applied Mathematics*, 6(1):91–93, 1983.

[128] John J Kasianowicz, Eric Brandin, Daniel Branton, and David W Deamer. Characterization of individual polynucleotide molecules using a membrane channel. *Proceedings of the National Academy of Sciences*, 93(24):13770–13773, 1996.

[129] John Kececioglu. The maximum weight trace problem in multiple sequence alignment. In *Annual Symposium on Combinatorial Pattern Matching*, pages 106–119. Springer, 1993.

[130] John D Kececioglu and Eugene W Myers. Combinatorial algorithms for DNA sequence assembly. *Algorithmica*, 13(1-2):7–51, 1995.

[131] Birte Kehr, Kathrin Trappe, Manuel Holtgrewe, and Knut Reinert. Genome alignment with graph data structures: a comparison. *BMC Bioinformatics*, 15(1): 99, 2014.

[132] W James Kent. BLAT—the BLAST-like alignment tool. *Genome Research*, 12(4): 656–664, 2002.

[133] W James Kent and David Haussler. Assembly of the working draft of the human genome with GigAssembler. *Genome Research*, 11(9):1541–1548, 2001.

[134] W James Kent, Charles W Sugnet, Terrence S Furey, Krishna M Roskin, Tom H Pringle, Alan M Zahler, and David Haussler. The human genome browser at UCSC. *Genome Research*, 12(6):996–1006, 2002.

[135] Paul Julian Kersey, James E Allen, Irina Armean, Sanjay Boddu, Bruce J Bolt, Denise Carvalho-Silva, Mikkel Christensen, Paul Davis, Lee J Falin, Christoph Grabmueller, et al. Ensembl genomes 2016: more genomes, more complexity. *Nucleic Acids Research*, 44(D1):D574–D580, 2015.

[136] Daehwan Kim, Ben Langmead, and Steven L Salzberg. HISAT: a fast spliced aligner with low memory requirements. *Nature Methods*, 12(4):357–360, 2015.

[137] Daehwan Kim, B Langmead, and S Salzberg. HISAT2: graph-based alignment of next-generation sequencing reads to a population of genomes. https://github.com/infphilo/hisat2, 2017.

[138] Daniel C Koboldt, Ken Chen, Todd Wylie, David E Larson, Michael D McLellan, Elaine R Mardis, George M Weinstock, Richard K Wilson, and Li Ding. VarScan: variant detection in massively parallel sequencing of individual and pooled samples. *Bioinformatics*, 25(17):2283–2285, 2009.

[139] Yuichi Kodama, Martin Shumway, and Rasko Leinonen. The sequence read archive: explosive growth of sequencing data. *Nucleic Acids Research*, 40(D1):D54–D56, 2011.

[140] Klaus-Peter Koepfli, Benedict Paten, Genome 10K Community of Scientists, and Stephen J O'Brien. The Genome 10K Project: a way forward. *Annual Review of Animal Biosciences*, 3(1):57–111, 2015.

[141] Sergey Koren, Brian P Walenz, Konstantin Berlin, Jason R Miller, Nicholas H Bergman, and Adam M Phillippy. Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation. *Genome Research*, 27(5): 722–736, 2017.

[142] Jonas Korlach, Patrick J Marks, Ronald L Cicero, Jeremy J Gray, Devon L Murphy, Daniel B Roitman, Thang T Pham, Geoff A Otto, Mathieu Foquet, and Stephen W Turner. Selective aluminum passivation for targeted immobilization of single DNA polymerase molecules in zero-mode waveguide nanostructures. *Proceedings of the National Academy of Sciences*, 105(4):1176–1181, 2008.

[143] Anna Kuosmanen, Topi Paavilainen, Travis Gagie, Rayan Chikhi, Alexandru Tomescu, and Veli Mäkinen. Using minimum path cover to boost dynamic programming on DAGs: co-linear chaining extended. In *International Conference on Research in Computational Molecular Biology*, pages 105–121. Springer, 2018.

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

[145] Ben Langmead, Cole Trapnell, Mihai Pop, and Steven L Salzberg. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biology*, 10(3):R25, 2009.

[146] Iosif Lazaridis, Nick Patterson, Alissa Mittnik, Gabriel Renaud, Swapan Mallick, Karola Kirsanow, Peter H Sudmant, Joshua G Schraiber, Sergi Castellano, Mark Lipson, et al. Ancient human genomes suggest three ancestral populations for present-day Europeans. *Nature*, 513(7518):409–413, 2014.

[147] Christopher Lee, Catherine Grasso, and Mark F Sharlow. Multiple sequence alignment using partial order graphs. *Bioinformatics*, 18(3):452–464, 2002.

[148] Wan-Ping Lee, Michael P Stromberg, Alistair Ward, Chip Stewart, Erik P Garrison, and Gabor T Marth. Mosaik: A hash-based algorithm for accurate next-generation sequencing short-read mapping. *PLoS ONE*, 9(3):e90581, 2014.

[149] Rasko Leinonen, Hideaki Sugawara, Martin Shumway, and International Nucleotide Sequence Database Collaboration. The sequence read archive. *Nucleic Acids Research*, 39(suppl_1):D19–D21, 2010.

[150] Dinghua Li, Chi-Man Liu, Ruibang Luo, Kunihiko Sadakane, and Tak-Wah Lam. MEGAHIT: an ultra-fast single-node solution for large and complex metagenomics assembly via succinct de Bruijn graph. *Bioinformatics*, 31(10):1674–1676, 2015.

[151] Heng Li. A statistical framework for SNP calling, mutation discovery, association mapping and population genetical parameter estimation from sequencing data. *Bioinformatics*, 27(21):2987–2993, 2011.

[152] Heng Li. Exploring single-sample SNP and INDEL calling with whole-genome de novo assembly. *Bioinformatics*, 28(14):1838–1844, 2012.

[153] Heng Li. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. *arXiv:1303.3997*, 2013.

[154] Heng Li. Fast construction of FM-index for long sequence reads. *Bioinformatics*, 30(22):3274–3275, 2014.

[155] Heng Li. Fermikit: assembly-based variant calling for Illumina resequencing data. *Bioinformatics*, 31(22):3694–3696, 2015.

[156] Heng Li. Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *Bioinformatics*, 32(14):2103–2110, 2016.

[157] Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18):3094–3100, 2018.

[158] Heng Li and Richard Durbin. Fast and accurate short read alignment with Burrows–Wheeler transform. *Bioinformatics*, 25(14):1754–1760, 2009.

[159] Heng Li and Richard Durbin. Fast and accurate long-read alignment with Burrows–Wheeler transform. *Bioinformatics*, 26(5):589–595, 2010.

[160] Heng Li, Jue Ruan, and Richard Durbin. Mapping short DNA sequencing reads and calling variants using mapping quality scores. *Genome Research*, 18(11):1851–1858, 2008.

[161] Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor Marth, Goncalo Abecasis, and Richard Durbin. The sequence alignment/map format and SAMtools. *Bioinformatics*, 25(16):2078–2079, 2009.

[162] Ruiqiang Li, Yingrui Li, Karsten Kristiansen, and Jun Wang. SOAP: short oligonucleotide alignment program. *Bioinformatics*, 24(5):713–714, 2008.

[163] Ruiqiang Li, Yingrui Li, Hancheng Zheng, Ruibang Luo, Hongmei Zhu, Qibin Li, Wubin Qian, Yuanyuan Ren, Geng Tian, Jinxiang Li, et al. Building the sequence map of the human pan-genome. *Nature Biotechnology*, 28(1):57–63, 2010.

[164] Erez Lieberman-Aiden, Nynke L Van Berkum, Louise Williams, Maxim Imakaev, Tobias Ragoczy, Agnes Telling, Ido Amit, Bryan R Lajoie, Peter J Sabo, Michael O Dorschner, et al. Comprehensive mapping of long-range interactions reveals folding principles of the human genome. *Science*, 326(5950):289–293, 2009.

[165] DJ Lightfoot, David Erwin Jarvis, T Ramaraj, R Lee, EN Jellen, and PJ Maughan. Single-molecule sequencing and Hi-C-based proximity-guided assembly of amaranth (Amaranthus hypochondriacus) chromosomes provide insights into genome evolution. *BMC Biology*, 15(1):74, 2017.

[166] Henry W Lin and Max Tegmark. Critical behavior in physics and probabilistic formal languages. *Entropy*, 19(7):299, 2017.

[167] Vivian Link, Athanasios Kousathanas, Krishna Veeramah, Christian Sell, Amelie Scheu, and Daniel Wegmann. ATLAS: analysis tools for low-depth and ancient samples. *bioRxiv:105346*, 2017.

[168] David J Lipman, Stephen F Altschul, and John D Kececioglu. A tool for multiple sequence alignment. *Proceedings of the National Academy of Sciences*, 86(12): 4412–4415, 1989.

[169] Gianni Liti, David M Carter, Alan M Moses, Jonas Warringer, Leopold Parts, Stephen A James, Robert P Davey, Ian N Roberts, Austin Burt, Vassiliki Koufopanou, et al. Population genomics of domestic and wild yeasts. *Nature*, 458 (7236):337–341, 2009.

[170] Bo Liu, Hongzhe Guo, Michael Brudno, and Yadong Wang. deBGA: read alignment with de Bruijn graph-based seed and extension. *Bioinformatics*, 32(21):3224–3232, 2016.

[171] Nicholas J Loman, Joshua Quick, and Jared T Simpson. A complete bacterial genome assembled de novo using only nanopore sequencing data. *Nature Methods*, 12(8):733–735, 2015.

[172] Sorina Maciuca, Carlos del Ojo Elias, Gil McVean, and Zamin Iqbal. A natural encoding of genetic variation in a Burrows-Wheeler Transform to enable mapping and genome inference. In *International Workshop on Algorithms in Bioinformatics*, pages 222–233. Springer, 2016.

[173] Tanja Magoč and Steven L Salzberg. FLASH: fast length adjustment of short reads to improve genome assemblies. *Bioinformatics*, 27(21):2957–2963, 2011.

[174] Michael A Mahowald, Federico E Rey, Henning Seedorf, Peter J Turnbaugh, Robert S Fulton, Aye Wollam, Neha Shah, Chunyan Wang, Vincent Magrini, Richard K Wilson, et al. Characterizing a model human gut microbiota composed of members of its two dominant bacterial phyla. *Proceedings of the National Academy of Sciences*, 106(14):5859–5864, 2009.

[175] BWJ Mahy, JJ Esposito, and JC Venter. Sequencing the smallpox virus genome: prelude to destruction of a virus species. *ASM News*, 57:577–580, 1991.

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

[177] Marcel Margulies, Michael Egholm, William E Altman, Said Attiya, Joel S Bader, Lisa A Bemben, Jan Berka, Michael S Braverman, Yi-Ju Chen, Zhoutao Chen, et al. Genome sequencing in microfabricated high-density picolitre reactors. *Nature*, 437(7057):376–380, 2005.

[178] Gabor T Marth, Ian Korf, Mark D Yandell, Raymond T Yeh, Zhijie Gu, Hamideh Zakeri, Nathan O Stitziel, LaDeana Hillier, Pui-Yan Kwok, and Warren R Gish. A general approach to single-nucleotide polymorphism discovery. *Nature Genetics*, 23 (4):452–456, 1999.

[179] Jeffrey A Martin and Zhong Wang. Next-generation transcriptome assembly. *Nature Reviews Genetics*, 12(10):671–682, 2011.

[180] Rui Martiniano, Anwen Caffell, Malin Holst, Kurt Hunter-Mann, Janet Montgomery, Gundula Müldner, Russell L McLaughlin, Matthew D Teasdale, Wouter Van Rheenen, Jan H Veldink, et al. Genomic signals of migration and continuity in Britain before the Anglo-Saxons. *Nature Communications*, 7:10326, 2016.

[181] Ryan McDaniell, Bum-Kyu Lee, Lingyun Song, Zheng Liu, Alan P Boyle, Michael R Erdos, Laura J Scott, Mario A Morken, Katerina S Kucera, Anna Battenhouse, et al. Heritable individual-specific and allele-specific chromatin signatures in humans. *Science*, 328(5975):235–239, 2010.

[182] Duccio Medini, Claudio Donati, Herve Tettelin, Vega Masignani, and Rino Rappuoli. The microbial pan-genome. *Current Opinion in Genetics & Development*, 15(6): 589–594, 2005.

[183] Gregor Mendel. Versuche über Pflanzenhybriden. *Verhandlungen des naturforschenden Vereines in Brunn*, 44:1–47, 1866.

[184] Androniki Menelaou and Jonathan Marchini. Genotype calling and phasing using next-generation sequencing reads and a haplotype scaffold. *Bioinformatics*, 29(1): 84–91, 2012.

[185] Alexander S Mikheyev and Mandy MY Tin. A first look at the oxford nanopore minion sequencer. *Molecular Ecology Resources*, 14(6):1097–1102, 2014.

[186] Jason R Miller, Arthur L Delcher, Sergey Koren, Eli Venter, Brian P Walenz, Anushka Brownley, Justin Johnson, Kelvin Li, Clark Mobarry, and Granger Sutton. Aggressive assembly of pyrosequencing reads with mates. *Bioinformatics*, 24(24): 2818–2824, 2008.

[187] Ilia Minkin, Son Pham, and Paul Medvedev. TwoPaCo: An efficient algorithm to build the compacted de Bruijn graph from many complete genomes. *Bioinformatics*, 33(24):4024–4032, 2016.

[188] Robi D Mitra, Jay Shendure, Jerzy Olejnik, George M Church, et al. Fluorescent in situ sequencing on polymerase colonies. *Analytical Biochemistry*, 320(1):55–65, 2003.

[189] Anthony P Monaco and Zoia Larin. YACs, BACs, PACs and MACs: artificial chromosomes as research tools. *Trends in Biotechnology*, 12(7):280–286, 1994.

[190] Burkhard Morgenstern, Andreas Dress, and Thomas Werner. Multiple DNA and protein sequence alignment based on segment-to-segment comparison. *Proceedings of the National Academy of Sciences*, 93(22):12098–12103, 1996.

[191] Yulia Mostovoy, Michal Levy-Sakin, Jessica Lam, Ernest T Lam, Alex R Hastie, Patrick Marks, Joyce Lee, Catherine Chu, Chin Lin, Željko Džakula, et al. A hybrid approach for de novo human genome sequence assembly and phasing. *Nature Methods*, 13(7):587–590, 2016.

[192] Martin D Muggli, Alexander Bowe, Noelle R Noyes, Paul S Morley, Keith E Belk, Robert Raymond, Travis Gagie, Simon J Puglisi, and Christina Boucher. Succinct colored de Bruijn graphs. *Bioinformatics*, 33(20):3181–3187, 2017.

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

[194] Eugene W Myers. The fragment assembly string graph. *Bioinformatics*, 21 (suppl_2):ii79–ii85, 2005.

[195] Eugene W Myers. Efficient local alignment discovery amongst noisy long reads. In *International Workshop on Algorithms in Bioinformatics*, pages 52–67. Springer, 2014.

[196] Eugene W Myers and Webb Miller. Optimal alignments in linear space. *Bioinformatics*, 4(1):11–17, 1988.

[197] Eugene W Myers and Webb Miller. Approximate matching of regular expressions. *Bulletin of Mathematical Biology*, 51(1):5–37, 1989.

[198] Eugene W Myers, Granger G Sutton, Art L Delcher, Ian M Dew, Dan P Fasulo, Michael J Flanigan, Saul A Kravitz, Clark M Mobarry, Knut HJ Reinert, Karin A Remington, et al. A whole-genome assembly of drosophila. *Science*, 287(5461): 2196–2204, 2000.

[199] Eugene W Myers, Granger G Sutton, Hamilton O Smith, Mark D Adams, and J Craig Venter. On the sequencing and assembly of the human genome. *Proceedings of the National Academy of Sciences*, 99(7):4145–4146, 2002.

[200] Saul B Needleman and Christian D Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970.

[201] Cédric Notredame, Desmond G Higgins, and Jaap Heringa. T-coffee: a novel method for fast and accurate multiple sequence alignment1. *Journal of Molecular Biology*, 302(1):205–217, 2000.

[202] Adam M Novak, Erik Garrison, and Benedict Paten. A graph extension of the positional Burrows–Wheeler transform and its applications. *Algorithms for Molecular Biology*, 12:18, 2017.

[203] Adam M Novak, Glenn Hickey, Erik Garrison, Sean Blum, Abram Connelly, Alexander Dilthey, Jordan Eizenga, MA Saleh Elmohamed, Sally Guthrie, André Kahles, et al. Genome graphs. *bioRxiv:101378*, 2017.

[204] Sergey Nurk, Dmitry Meleshko, Anton Korobeynikov, and Pavel A Pevzner. metaSPAdes: a new versatile metagenomic assembler. *Genome Research*, 27(5):824–834, 2017.

[205] Genome 10K Community of Scientists. Genome 10K: a proposal to obtain whole-genome sequence for 10 000 vertebrate species. *Journal of Heredity*, 100(6):659–674, 2009.

[206] Yukiteru Ono, Kiyoshi Asai, and Michiaki Hamada. PBSIM: PacBio reads simulator—toward accurate genome assembly. *Bioinformatics*, 29(1):119–121, 2012.

[207] Taku Onodera, Kunihiko Sadakane, and Tetsuo Shibuya. Detecting superbubbles in assembly graphs. In *International Workshop on Algorithms in Bioinformatics*, pages 338–348. Springer, 2013.

[208] R Padmanabhan, Ernest Jay, and Ray Wu. Chemical synthesis of a primer and its use in the sequence analysis of the lysozyme gene of bacteriophage t4. *Proceedings of the National Academy of Sciences*, 71(6):2510–2514, 1974.

[209] Andrew J Page, Carla A Cummins, Martin Hunt, Vanessa K Wong, Sandra Reuter, Matthew TG Holden, Maria Fookes, Daniel Falush, Jacqueline A Keane, and Julian Parkhill. Roary: rapid large-scale prokaryote pan genome analysis. *Bioinformatics*, 31(22):3691–3693, 2015.

[210] 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.

[211] Benedict Paten, Mark Diekhans, Dent Earl, John St John, Jian Ma, Bernard Suh, and David Haussler. Cactus graphs for genome comparisons. *Journal of Computational Biology*, 18(3):469–481, 2011.

[212] Benedict Paten, Dent Earl, Ngan Nguyen, Mark Diekhans, Daniel Zerbino, and David Haussler. Cactus: Algorithms for genome multiple sequence alignment. *Genome Research*, 21(9):1512–1528, 2011.

[213] Benedict Paten, Jordan M Eizenga, Yohei M Rosen, Adam M Novak, Erik Garrison, and Glenn Hickey. Superbubbles, ultrabubbles, and cacti. *Journal of Computational Biology*, 25(7):649–663, 2018.

[214] N. Patterson, P. Moorjani, Y. Luo, S. Mallick, N. Rohland, Y. Zhan, T. Genschoreck, T. Webster, and D. Reich. Ancient admixture in human history. *Genetics*, 192(3): 1065–1093, 2012.

[215] William R Pearson and David J Lipman. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences*, 85(8):2444–2448, 1988.

[216] Jane Peterson, Susan Garges, Maria Giovanni, Pamela McInnes, Lu Wang, Jeffery A Schloss, Vivien Bonazzi, Jean E McEwen, Kris A Wetterstrand, Carolyn Deal, et al. The NIH human microbiome project. *Genome Research*, 19(12):2317–2323, 2009.

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

[218] Ryan Poplin, Dan Newburger, Jojo Dijamco, Nam Nguyen, Dion Loy, Sam S Gross, Cory Y McLean, and Mark A DePristo. Creating a universal snp and small indel variant caller with deep neural networks. *bioRxiv:092890*, 2017.

[219] David Porubskỳ, Ashley D Sanders, Niek Van Wietmarschen, Ester Falconer, Mark Hills, Diana CJ Spierings, Marianna R Bevova, Victor Guryev, and Peter M Lansdorp. Direct chromosome-length haplotyping by single-cell sequencing. *Genome Research*, 26(11):1565–1574, 2016.

[220] Kay Prüfer. snpAD: an ancient DNA genotype caller. *Bioinformatics*, 2018. doi: 10.1093/bioinformatics/bty507.

[221] Robert F Purnell, Kunal K Mehta, and Jacob J Schmidt. Nucleotide identification and orientation discrimination of DNA homopolymers immobilized in a protein nanopore. *Nano Letters*, 8(9):3029–3034, 2008.

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

[223] Goran Rakocevic, Vladimir Semenyuk, James Spencer, John Browning, Ivan Johnson, Vladan Arsenijevic, Jelena Nadj, Kaushik Ghose, Maria C Suciu, Sun-Gou Ji, et al. Fast and accurate genomic analyses using genome graphs. *bioRxiv:194530*, 2018.

[224] Rajeev Raman, Venkatesh Raman, and S Srinivasa Rao. Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 233–242. Society for Industrial and Applied Mathematics, 2002.

[225] Benjamin Raphael, Degui Zhi, Haixu Tang, and Pavel Pevzner. A novel method for multiple alignment of sequences with repeated and shuffled elements. *Genome Research*, 14(11):2336–2346, 2004.

[226] Tobias Rausch, Anne-Katrin Emde, David Weese, Andreas Döring, Cedric Notredame, and Knut Reinert. Segment-based multiple sequence alignment. *Bioinformatics*, 24(16):i187–i192, 2008.

[227] Mikko Rautiainen, Veli Mäkinen, and Tobias Marschall. Bit-parallel sequence-to-graph alignment. *bioRxiv:323063*, 2018.

[228] Gabriel Renaud, Kristian Hanghøj, Eske Willerslev, and Ludovic Orlando. gargammel: a sequence simulator for ancient DNA. *Bioinformatics*, 33(4):577–579, 2016.

[229] Anthony Rhoads and Kin Fai Au. PacBio sequencing and its applications. *Genomics, Proteomics & Bioinformatics*, 13(5):278–289, 2015.

[230] Andy Rimmer, Hang Phan, Iain Mathieson, Zamin Iqbal, Stephen RF Twigg, Andrew OM Wilkie, Gil McVean, Gerton Lunter, WGS500 Consortium, et al. Integrating mapping-, assembly-and haplotype-based approaches for calling variants in clinical sequencing applications. *Nature Genetics*, 46(8):912–918, 2014.

[231] Gordon Robertson, Jacqueline Schein, Readman Chiu, Richard Corbett, Matthew Field, Shaun D Jackman, Karen Mungall, Sam Lee, Hisanaga Mark Okada, Jenny Q Qian, et al. De novo assembly and analysis of RNA-seq data. *Nature Methods*, 7 (11):909–912, 2010.

[232] Michael G Ross, Carsten Russ, Maura Costello, Andrew Hollinger, Niall J Lennon, Ryan Hegarty, Chad Nusbaum, and David B Jaffe. Characterizing and measuring bias in sequence data. *Genome Biology*, 14(5):R51, 2013.

[233] Nicole Rusk. Torrents of sequence. *Nature Methods*, 8(1):44–44, 2010.

[234] F Sanger, GG Brownlee, and BG Barrell. A two-dimensional fractionation procedure for radioactive nucleotides. *Journal of Molecular Biology*, 13(2):373–398, 1965.

[235] F Sanger, Ar R Coulson, GF Hong, DF Hill, and GB d Petersen. Nucleotide sequence of bacteriophage λ DNA. *Journal of Molecular Biology*, 162(4):729–773, 1982.

[236] Frederick Sanger and Hans Tuppy. The amino-acid sequence in the phenylalanyl chain of insulin. 1. the identification of lower peptides from partial hydrolysates. *Biochemical Journal*, 49(4):463–481, 1951.

[237] Frederick Sanger, Gilian M Air, Bart G Barrell, Nigel L Brown, Alan R Coulson, John C Fiddes, Clyde A Hutchison III, Patrick M Slocombe, and Mo Smith. Nucleotide sequence of bacteriophage φx174 DNA. *Nature*, 265(5596):687–695, 1977.

[238] Frederick Sanger, Steven Nicklen, and Alan R Coulson. DNA sequencing with chain-terminating inhibitors. *Proceedings of the National Academy of Sciences*, 74 (12):5463–5467, 1977.

[239] Stephan Schiffels, Wolfgang Haak, Pirita Paajanen, Bastien Llamas, Elizabeth Popescu, Louise Loe, Rachel Clarke, Alice Lyons, Richard Mortimer, Duncan Sayer, et al. Iron age and Anglo-Saxon genomes from East England reveal British migration history. *Nature Communications*, 7:10408, 2016.

[240] Michael Schmid, Daniel Frei, Andrea Patrignani, Ralph Schlapbach, Juerg E Frey, Mitja NP Remus-Emsermann, and Christian H Ahrens. Pushing the limits of de novo genome assembly for complex prokaryotic genomes harboring very long, near identical repeats. *bioRxiv:300186*, 2018.

[241] Desmond Schmidt and Robert Colomb. A data structure for representing multi-version texts online. *International Journal of Human-Computer Studies*, 67(6): 497–514, 2009.

[242] Holger Schmitt, Ung-Jin Kim, Tatiana Slepak, Nikolaus Blin, Melvin I Simon, and Hiroaki Shizuya. Framework for a physical map of the human 22q13 region using bacterial artificial chromosomes (BACs). *Genomics*, 33(1):9–20, 1996.

[243] Korbinian Schneeberger, Jörg Hagmann, Stephan Ossowski, Norman Warthmann, Sandra Gesing, Oliver Kohlbacher, and Detlef Weigel. Simultaneous alignment of short reads against multiple genomes. *Genome Biology*, 10(9):R98, 2009.

[244] Valerie A Schneider, Tina Graves-Lindsay, Kerstin Howe, Nathan Bouk, Hsiu-Chuan Chen, Paul A Kitts, Terence D Murphy, Kim D Pruitt, Françoise Thibaud-Nissen, Derek Albracht, et al. Evaluation of GRCh38 and de novo haploid genome assemblies demonstrates the enduring quality of the reference assembly. *Genome Research*, 27 (5):849–864, 2017.

[245] Marcel H Schulz, Daniel R Zerbino, Martin Vingron, and Ewan Birney. Oases: robust de novo RNA-seq assembly across the dynamic range of expression levels. *Bioinformatics*, 28(8):1086–1092, 2012.

[246] Jay Shendure and Hanlee Ji. Next-generation DNA sequencing. *Nature Biotechnology*, 26(10):1135–1145, 2008.

[247] Jay Shendure, Gregory J Porreca, Nikos B Reppas, Xiaoxia Lin, John P McCutcheon, Abraham M Rosenbaum, Michael D Wang, Kun Zhang, Robi D Mitra, and George M Church. Accurate multiplex polony sequencing of an evolved bacterial genome. *Science*, 309(5741):1728–1732, 2005.

[248] Stephen T Sherry, M-H Ward, M Kholodov, J Baker, Lon Phan, Elizabeth M Smigielski, and Karl Sirotkin. dbSNP: the NCBI database of genetic variation. *Nucleic Acids Research*, 29(1):308–311, 2001.

[249] Jonas Andreas Sibbesen, Lasse Maretty, The Danish Pan-Genome Consortium, and Anders Krogh. Accurate genotyping across variant classes and lengths using variant graphs. *Nature Genetics*, 50:1054–1059, 2018.

[250] F Sigaux. Cancer genome or the development of molecular portraits of tumors. *Bulletin de L'Academie Nationale de Medecine*, 184(7):1441–7, 2000.

[251] Birgit Sikkema-Raddatz, Lennart F Johansson, Eddy N de Boer, Rowida Almomani, Ludolf G Boven, Maarten P van den Berg, Karin Y van Spaendonck-Zwarts, J Peter van Tintelen, Rolf H Sijmons, Jan DH Jongbloed, et al. Targeted next-generation sequencing can replace Sanger sequencing in clinical diagnostics. *Human Mutation*, 34(7):1035–1042, 2013.

[252] Jared T Simpson. Exploring genome characteristics and sequence quality without a reference. *Bioinformatics*, 30(9):1228–1235, 2014.

[253] Jared T Simpson and Richard Durbin. Efficient construction of an assembly string graph using the FM-index. *Bioinformatics*, 26(12):i367–373, 2010.

[254] Jared T Simpson and Richard Durbin. Efficient de novo assembly of large genomes using compressed data structures. *Genome Research*, 22(3):549–556, 2012.

[255] Jared T Simpson, Kim Wong, Shaun D Jackman, Jacqueline E Schein, Steven JM Jones, and Inanç Birol. ABySS: a parallel assembler for short read sequence data. *Genome Research*, 19(6):1117–1123, 2009.

[256] Jouni Sirén. Indexing variation graphs. In *2017 Proceedings of the ninteenth workshop on algorithm engineering and experiments (ALENEX)*, pages 13–27. SIAM, 2017.

[257] Jouni Sirén, Niko Välimäki, and Veli Mäkinen. Indexing finite language representation of population genotypes. In *International Workshop on Algorithms in Bioinformatics*, pages 270–281. Springer, 2011.

[258] Jouni Sirén, Niko Välimäki, and Veli Mäkinen. Indexing graphs for path queries with applications in genome research. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 11(2):375–388, 2014.

[259] Jouni Sirén, Erik Garrison, Adam M Novak, Benedict Paten, and Richard Durbin. Haplotype-aware graph indexes. *arXiv:1805.03834*, 2018.

[260] Temple F Smith. Functional genomics—bioinformatics is ready for the challenge. *Trends in Genetics*, 14(7):291–293, 1998.

[261] Temple F Smith and Michael S Waterman. Comparison of biosequences. *Advances in Applied Mathematics*, 2(4):482–489, 1981.

[262] Rodger Staden. A strategy of DNA sequencing employing computer programs. *Nucleic Acids Research*, 6(7):2601–2610, 1979.

[263] Kraig R Stevenson, Joseph D Coolon, and Patricia J Wittkopp. Sources of bias in measures of allele-specific expression derived from RNA-seq data aligned to a single reference genome. *BMC Genomics*, 14(1):536, 2013.

[264] Erich C Strauss, Joan A Kobori, Gerald Siu, and Leroy E Hood. Specific-primer-directed DNA sequencing. *Analytical Biochemistry*, 154(1):353–360, 1986.

[265] P. H. Sudmant, J. O. Kitzman, F. Antonacci, C. Alkan, M. Malig, A. Tsalenko, N. Sampas, L. Bruhn, J. Shendure, 1000 Genomes Project, and E. E. Eichler. Diversity of human copy number variation and multicopy genes. *Science*, 330(6004): 641–646, 2010.

[266] Peter H Sudmant, Tobias Rausch, Eugene J Gardner, Robert E Handsaker, Alexej Abyzov, John Huddleston, Yan Zhang, Kai Ye, Goo Jun, Markus Hsi-Yang Fritz, et al. An integrated map of structural variation in 2,504 human genomes. *Nature*, 526(7571):75–81, 2015.

[267] Granger G Sutton, Owen White, Mark D Adams, and Anthony R Kerlavage. TIGR Assembler: A new tool for assembling large shotgun sequencing projects. *Genome Science and Technology*, 1(1):9–19, 1995.

[268] Hajime Suzuki and Masahiro Kasahara. Acceleration of nucleotide semi-global alignment with adaptive banded dynamic programming. *bioRxiv:130633*, 2017.

[269] Artem Tarasov, Albert J Vilella, Edwin Cuppen, Isaac J Nijman, and Pjotr Prins. Sambamba: fast processing of NGS alignment formats. *Bioinformatics*, 31(12): 2032–2034, 2015.

[270] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[271] Aaron E Tenney, Jia Qian Wu, Laura Langton, Paul Klueh, Ralph Quatrano, and Michael R Brent. A tale of two templates: Automatically resolving double traces has many applications, including efficient pcr-based elucidation of alternative splices. *Genome Research*, 17(2):212–218, 2007.

[272] Hervé Tettelin, Vega Masignani, Michael J Cieslewicz, Claudio Donati, Duccio Medini, Naomi L Ward, Samuel V Angiuoli, Jonathan Crabtree, Amanda L Jones, A Scott Durkin, et al. Genome analysis of multiple pathogenic isolates of streptococcus agalactiae: implications for the microbial "pan-genome". *Proceedings of the National Academy of Sciences*, 102(39):13950–13955, 2005.

[273] Julie D Thompson, Desmond G Higgins, and Toby J Gibson. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22(22):4673–4680, 1994.

[274] Cole Trapnell, Adam Roberts, Loyal Goff, Geo Pertea, Daehwan Kim, David R Kelley, Harold Pimentel, Steven L Salzberg, John L Rinn, and Lior Pachter. Differential gene and transcript expression analysis of RNA-seq experiments with tophat and cufflinks. *Nature Protocols*, 7(3):562–578, 2012.

[275] Susannah Green Tringe, Christian Von Mering, Arthur Kobayashi, Asaf A Salamov, Kevin Chen, Hwai W Chang, Mircea Podar, Jay M Short, Eric J Mathur, John C Detter, et al. Comparative metagenomics of microbial communities. *Science*, 308 (5721):554–557, 2005.

[276] Peter J Turnbaugh, Ruth E Ley, Micah Hamady, Claire M Fraser-Liggett, Rob Knight, and Jeffrey I Gordon. The human microbiome project. *Nature*, 449(7164): 804–810, 2007.

[277] Isaac Turner, Kiran V Garimella, Zamin Iqbal, and Gil McVean. Integrating long-range connectivity information into de Bruijn graphs. *Bioinformatics*, 34(15): 2556–2565, 2018.

[278] Esko Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

[279] Daniel Valenzuela and Veli Mäkinen. CHIC: a short read aligner for pan-genomic references. *bioRxiv:178129*, 2017.

[280] Robert Vaser, Ivan Sović, Niranjan Nagarajan, and Mile Šikić. Fast and accurate de novo genome assembly from long uncorrected reads. *Genome Research*, 27(5): 737–746, 2017.

[281] J Craig Venter, Mark D Adams, Eugene W Myers, Peter W Li, Richard J Mural, Granger G Sutton, Hamilton O Smith, Mark Yandell, Cheryl A Evans, Robert A Holt, et al. The sequence of the human genome. *Science*, 291(5507):1304–1351, 2001.

[282] George Vernikos, Duccio Medini, David R Riley, and Herve Tettelin. Ten years of pan-genome analyses. *Current Opinion in Microbiology*, 23:148–154, 2015.

[283] Y. Wang, J. Lu, J. Yu, R. A. Gibbs, and F. Yu. An integrative variant analysis pipeline for accurate genotype/haplotype inference in population NGS data. *Genome Research*, 23(5):833–842, 2013.

[284] Robert H Waterston, Eric S Lander, and John E Sulston. On the sequencing of the human genome. *Proceedings of the National Academy of Sciences*, 99(6):3712–3716, 2002.

[285] James D Watson, Francis HC Crick, et al. Molecular structure of nucleic acids. *Nature*, 171(4356):737–738, 1953.

[286] Detlef Weigel and Richard Mott. The 1001 genomes project for Arabidopsis thaliana. *Genome Biology*, 10(5):107, 2009.

[287] Peter Weiner. Linear pattern matching algorithms. In *Switching and Automata Theory, 1973. SWAT'08. IEEE Conference Record of 14th Annual Symposium on*, pages 1–11. IEEE, 1973.

[288] David A Wheeler, Maithreyan Srinivasan, Michael Egholm, Yufeng Shen, Lei Chen, Amy McGuire, Wen He, Yi-Ju Chen, Vinod Makhijani, G Thomas Roth, et al. The complete genome of an individual by massively parallel DNA sequencing. *Nature*, 452(7189):872–876, 2008.

[289] Ryan R Wick, Mark B Schultz, Justin Zobel, and Kathryn E Holt. Bandage: interactive visualization of de novo genome assemblies. *Bioinformatics*, 31(20): 3350–3352, 2015.

[290] Ray Wu. Nucleotide sequence analysis of DNA. *Nature New Biology*, 236(68): 198–200, 1972.

[291] Sun Wu, Udi Manber, and Eugene Myers. A subquadratic algorithm for approximate regular expression matching. *Journal of Algorithms*, 19(3):346–360, 1995.

[292] Jia-Xing Yue, Jing Li, Louise Aigrain, Johan Hallin, Karl Persson, Karen Oliver, Anders Bergström, Paul Coupland, Jonas Warringer, Marco Cosentino Lagomarsino, et al. Contrasting evolutionary genome dynamics between domesticated and wild yeasts. *Nature Genetics*, 49(6):913–924, 2017.

[293] Georg Zeller, Richard M Clark, Korbinian Schneeberger, Anja Bohlen, Detlef Weigel, and Gunnar Rätsch. Detecting polymorphic regions in Arabidopsis thaliana with resequencing microarrays. *Genome Research*, 18(6):918–929, 2008.

[294] Daniel Zerbino and Ewan Birney. Velvet: algorithms for de novo short read assembly using de Bruijn graphs. *Genome Research*, 18(5):821–829, 2008.

[295] Mengyao Zhao, Wan-Ping Lee, Erik P Garrison, and Gabor T Marth. SSW library: An SIMD Smith-Waterman C/C++ library for use in genomic applications. *PLoS ONE*, 8(12):e82138, 2013.

[296] Grace XY Zheng, Billy T Lau, Michael Schnall-Levin, Mirna Jarosz, John M Bell, Christopher M Hindson, Sofia Kyriazopoulou-Panagiotopoulou, Donald A Masquelier, Landon Merrill, Jessica M Terry, et al. Haplotyping germline and cancer genomes with high-throughput linked-read sequencing. *Nature Biotechnology*, 34 (3):303–311, 2016.

[297] Boyan Zhou, Shaoqing Wen, Lingxiang Wang, Li Jin, Hui Li, and Hong Zhang. Antcaller: an accurate variant caller incorporating ancient DNA damage. *Molecular Genetics and Genomics*, 292(6):1419–1430, 2017.

[298] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.

[299] Justin M Zook, Brad Chapman, Jason Wang, David Mittelman, Oliver Hofmann, Winston Hide, and Marc Salit. Integrating human sequence data sets provides a resource of benchmark SNP and indel genotype calls. *Nature Biotechnology*, 32(3): 246–251, 2014.

[300] Justin M Zook, David Catoe, Jennifer McDaniel, Lindsay Vang, Noah Spies, Arend Sidow, Ziming Weng, Yuling Liu, Christopher E Mason, Noah Alexander, et al. Extensive sequencing of seven human genomes to characterize benchmark reference materials. *Scientific Data*, 3:160025, 2016.

Related publications

During my graduate studies I have been an author on a number of publications which are related to this work. These include the following:

- Garrison, Erik, Jouni Sirén, Adam M. Novak, Glenn Hickey, Jordan M. Eizenga, Eric T. Dawson, William Jones et al. "Variation graph toolkit improves read mapping by representing genetic variation in the reference." *Nature Biotechnology*, 36(9):875–879, (2018).
- Paten, Benedict, Jordan M. Eizenga, Yohei M. Rosen, Adam M. Novak, Erik Garrison, and Glenn Hickey. "Superbubbles, ultrabubbles, and cacti." *Journal of Computational Biology*, 25(7):649–663, (2018).
- Garg, Shilpa, Mikko Rautiainen, Adam M. Novak, Erik Garrison, Richard Durbin, and Tobias Marschall. "A graph-based approach to diploid genome assembly." *Bioinformatics*, 34(13):i105–i114, (2018).
- Sirén, Jouni, Erik Garrison, Adam M. Novak, Benedict Paten, and Richard Durbin. "Haplotype-aware graph indexes." *arXiv:1805.03834* (2018).
- Paten, Benedict, Adam M. Novak, Jordan M. Eizenga, and Erik Garrison. "Genome graphs and the evolution of genome inference." *Genome Research*, 27(5):665–676, (2017).
- Novak, Adam M., Glenn Hickey, Erik Garrison, Sean Blum, Abram Connelly, Alexander Dilthey, Jordan Eizenga et al. "Genome graphs." *bioRxiv:101378* (2017).
- Computational pan-genomics consortium. "Computational pan-genomics: status, promises and challenges." *Briefings in Bioinformatics*, 19(1):118–135, (2016).
- Novak, Adam M., Erik Garrison, and Benedict Paten. "A graph extension of the positional Burrows–Wheeler transform and its applications." *Algorithms for Molecular Biology*, 12:18, (2017).