Hypothesis-free detection of genome-changing events in pedigree sequencing



Kiran V Garimella
Green Templeton College
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy

Hilary 2016

To my sister, Sesi. My place as favorite child is now eternally secure. Booya!

Acknowledgements

In 2004, I enrolled in the physics graduate program at Clemson University. I expected to earn my PhD in six or seven years. Instead, I opted to leave with a master's degree after two. My first doctoral supervisor was puzzled (and perhaps a bit frustrated) at my decision to leave, but asked me a very simple question before my departure: "What do you want?"

The path from there to here has been sinuous, each point along the way an opportunity to recalibrate my aim towards answering that question. The most fruitful stop by far was my time at the Broad Institute. My initial interview with Jared Maguire, David Jaffe, and others convinced me that the job would be incredibly fun, even if I didn't quite understand all this "genomics" stuff yet. Later, Mark DePristo gave me some software to write, then some more, then a project to work on, autonomy to own it, a promotion, Chris Hartl to supervise, then a group, another promotion, and finally a friendly nudge to leave the nest and pursue my own doctorate. I am incredibly thankful that he taught me and then pushed me to do better work every day. And to that old team, Eric Banks, Matt Hanna, Ryan Poplin, Chris Hartl, and many more: I learned so much from your examples. You forced me to step up my game to keep up with your brilliance, but I had so much fun in your company that it never felt like work. I wouldn't be here without you.

I am grateful to Gil McVean for acting as my guide these past four years. Being given such freedom to ponder, experiment with ideas, and find my own way with only the gentlest nudges back within bounds is daunting, but also an extraordinary gesture of trust. I admire his patience, perseverence, and attention to detail. At times, I found my confidence in my work and myself waning. Thank you for helping me rediscover both by giving me the time to search and the faith (by proxy) that both were there to be found.

Zam, Jerome, Isaac, and Joe were partners in innumerable discussions that helped me shape ideas along the way, excellent role models, and good fun to chat with when I needed a break. Anna gave a scientific and non-genomics

sounding board, patiently listening to me speak in acronyms in return for me patiently listening to marine biology words she clearly made up. Jonathan Flint was always happy to take time out of his busy schedule to talk about science, people, frustrations therein, and encouraged me to get my head out of the coalescent book and try out some ideas of my own.

I was fortunate that many of my fellow students in the Genomic Medicine and Statistics PhD program became my housemates and closest friends. Thank you to Carme, Robbie, Maria, Winni, and Na for always being willing to talk shop, embark on creative distractions, keeping me well fed, preventing me from accidentally burning down the house whenever Thanksgiving came around, and for putting up with my... *unique* memory with regards to laundry and dishes. I might owe you all an uncountably infinite number of drinks in gratitude.

I am thankful that the first person I met upon arrival in Oxford, Dorothy, is *not* a student in the GMS program, as our friendship has pushed me to explore a world beyond my computer. Thank you for the day trips, the dog visits, the dancing, and the distractions. Thank you for training me to run that first half-marathon. Thanks to Ruth for running the second one with me, to Morwenna for the bike ride to London, and to Linda for not making me do any of that.

Ten years later, on the verge of finally completing a PhD (albeit in a completely different field), I'm reconsidering my first supervisor's question, "What do you want?". After a decade of trying a number of different things, I can say this: "To be happy. And working on interesting problems with interesting people makes me happy." Thank you, Dieter, for asking the question.

Abstract

In high-diversity populations, a complete accounting of *de novo* mutations can be difficult to obtain. Most analyses involve identifying such mutations by sequencing pedigrees on second-generation sequencing platforms and aligning the short reads to a reference assembly, the genomic sequence of a canonical member (or members) of a species. Often, large regions of the genomes under study may be greatly diverged from the reference sequence, or not represented at all (e.g. the HLA, antigenic genes, or other regions under balancing selective pressure). If the haplotypic background upon which a mutation occurs is absent, events can easily be missed (as reads have nowhere to align) and false-positives may abound (as the software forces the reads to align elsewhere).

This thesis presents a novel method for *de novo* mutation discovery and allele identification. Rather than relying on alignment, our method is based on the *de novo* assembly of short-read sequence data using a multi-color de Bruijn graph. In this data structure, each sample is assigned a unique index (or "color"), reads from each sample are decomposed into smaller subsequences of length k (or "kmers"), and color-specific adjacency information between kmers is recorded. Mutations can be discovered in the graph itself by searching for characteristic motifs (e.g. a "bubble motifs", indicative of a SNP or indel, and "linear motifs" indicative of allelic and non-allelic recombination). *De novo* mutations differ from inherited mutations in that the kmers spanning the variant allele are absent in the parents; in a sense, they facilitate their own discovery by generating "novel" sequence. We exploit this fact to limit processing of the graph to only those regions containing these novel kmers.

We verified our approach using simulations, validation, and visualization. On the simulations, we developed genome and read generation software driven by empirical distributions computed from real data to emit genomes with realistic features: recombinations, *de novo* variants, read fragment sizes, sequencing errors, and coverage profiles. In 20 artifical samples, we determined our sensitivity and specificity for novel kmer recovery to be approximately 98% and

100% at worst, respectively. Not every novel stretch can be reconstituted as a variant, owing to errors and homology in the graph. In simulations, our false discovery rate was 10% for "bubble" events and 12% for "linear" events. On validation, we obtained a high-quality draft assembly for a single *P. falciparum* child using a third-generation sequencing platform. We discovered three *de novo* events in the draft assembly, all three of which are recapitulated in our calls on the second-generation sequencing data for the same sample; no false-positives are present. On visualization, we developed an interactive web application capable of rendering a multi-color subgraph that assists in visually distinguishing between true variation and sequencing artifacts.

We applied our caller to real datasets: 115 progeny across four previously analyzed experimental crosses of *Plasmodium falciparum*. We demonstrate our ability to access subtelomeric compartments of the genome, regions harboring antigenic genes under tremendous selective pressure, thus highly divergent between geographically distinct isolates and routinely masked and ignored in reference-based analyses. We also show our caller's ability to recover an important form of structural *de novo* variation: non-allelic homologous recombination (NAHR) events, an important mechanism for the pathogen to diversify its own antigenic repertoire. We demonstrate our ability to recover the few events in these samples known to exist, and overturn some previous findings indicating exchanges between "core" (non-subtelomeric) genes. We compute the SNP mutation rate to be approximately 2.91 per sample, insertion and deletion mutation rates to be 0.55 and 1.04 per sample, respectively, multi-nucleotide polymorphisms to be 0.72 per sample, and NAHR events to be 0.33 per sample. These findings are consistent across crosses.

Finally, we investigated our method's scaling capabilities by processing a quintet of previously analyzed *Pan troglodytes verus* (western chimpanzee) samples. The genome of the chimpanzee is two orders of magnitude larger than the malaria parasite's ($3,300$ Mbp versus 23 Mbp), diploid rather than haploid, poorly assembled, and the read dataset is lower coverage ($20x$ versus $120x$). Comparing to Sequenom validation data as well as visual validation, our sensitivity is expectedly low. However, this can be attributed to overaggressiveness in data cleaning applied by the *de novo* assembler atop which our software is built. We discuss the precise changes that would likely need to be made in future work to adapt our method to low-coverage samples.

Contents

1 Motivation **1**

1.1 Introduction . . . 1

1.2 *De novo* mutations (DNMs) in *Plasmodium falciparum* . . . 2

1.2.1 A reference-based approach for DNM discovery and allele identification . . . 3

1.2.2 A reference-free approach for assessing DNM sensitivity . . . 4

1.3 DNM sensitivity of the reference-based approach . . . 6

1.3.1 Data processing . . . 6

1.3.2 Results . . . 8

1.4 Discussion . . . 10

1.4.1 Failure of the mapping approach . . . 10

1.4.2 *De novo* assembly as an alternative approach . . . 14

1.5 Overview of this work . . . 17

2 Background **19**

2.1 Introduction . . . 19

2.2 Lifecycle of the *P. falciparum* parasite . . . 19

2.2.1 Liver stage . . . 19

2.2.2 Blood stage . . . 20

2.2.3 Reproduction . . . 21

2.2.4 The *P. falciparum* crosses . . . 22

2.3 Mutational types . . . 23

2.3.1 Point mutations . . . 23

2.3.2 Insertions, deletions, inversions . . . 24

2.3.3 Cross-overs . . . 25

2.3.4 Gene conversion . . . 26

2.3.5 Non-allelic homologous recombination (NAHR) . . . 26

2.4 Known rates . . . 27

2.5 Biases . . . 28
2.6 Summary . . . 29

3 Simulation **30**
3.1 Introduction . . . 30
3.2 Simulating genomes . . . 31
3.2.1 Parents . . . 32
3.3 Children . . . 35
3.3.1 Homologous recombination . . . 35
3.3.1.1 Allelic homologous recombination . . . 35
3.3.1.2 Gene conversion . . . 37
3.3.1.3 Non-allelic homologous recombination . . . 37
3.3.2 SNPs, insertions, and deletions . . . 39
3.3.2.1 Expansion and contraction of short tandem repeats (STRs) 39
3.3.2.2 Tandem duplications . . . 41
3.4 Simulating reads . . . 41
3.4.1 Constructing the coverage profile . . . 42
3.4.1.1 Computing read and fragment starts, and error rates . . . 42
3.4.1.2 Lifting read profile over from reference to simulated genome 43
3.4.2 Constructing the read profile . . . 45
3.4.2.1 Scalar properties . . . 45
3.4.2.2 Empirical distributions . . . 46
3.4.2.3 Covariate table . . . 46
3.4.3 The read simulator . . . 48
3.5 The simulated dataset . . . 48
3.6 Summary . . . 48

4 Detection and classification **52**
4.1 Introduction . . . 52
4.2 *De novo* assembly . . . 52
4.2.1 Definitions . . . 54
4.2.2 Overview of the Cortex graph format . . . 57
4.2.3 The Java-based Cortex graph API . . . 58
4.2.3.1 CortexKmer . . . 58
4.2.3.2 CortexRecord . . . 58
4.2.4 CortexGraph . . . 59
4.2.5 CortexUtils . . . 59

4.3 Variant motifs . . . 61
4.3.1 Simple variant motifs . . . 61
4.3.2 Complex variant motifs . . . 64
4.3.3 Handling errors in sequencing . . . 68
4.4 Calling and classifing *de novo* variants . . . 69
4.4.1 Identify confident and trusted novel kmers . . . 69
4.4.1.1 Remove low coverage kmers . . . 70
4.4.1.2 Remove possible contaminants . . . 70
4.4.2 Construct multi-color de Bruijn "trio" graphs . . . 71
4.4.3 Graph cleaning . . . 73
4.4.4 Load subgraph local to a novel kmer . . . 73
4.4.4.1 Depth-first search . . . 74
4.4.4.2 Stopping conditions for child . . . 75
4.4.4.3 Stopping conditions for parents . . . 77
4.4.5 Identify and classify variants in the subgraph . . . 77
4.4.5.1 Dijkstra's shortest path algorithm . . . 78
4.4.5.2 Classify event . . . 81
4.4.5.3 Choosing haplotypic background . . . 81
4.4.5.4 Mark traversed novel kmers as used . . . 84
4.4.6 Evaluate performance . . . 84
4.4.6.1 Pre-compute novel kmer to variant map . . . 85
4.4.6.2 Load variant containing a novel kmer and comparing . . . 85
4.5 Results on simulated data . . . 85
4.6 Summary . . . 92

5 Validation **93**
5.1 Introduction . . . 93
5.1.1 Data processing for validation isolates . . . 93
5.1.1.1 Establishing assembly quality . . . 94
5.1.1.2 Preparing the validation isolate reference . . . 98
5.2 Comparison to validation isolate . . . 100
5.2.1 Verification of novel kmers . . . 100
5.2.2 Comparison to reference-based analysis . . . 105
5.3 Summary . . . 111

6 *De novo* mutations in *Plasmodium falciparum* 115
6.1 Introduction . . . 115
6.2 Data processing . . . 116
6.2.1 Initial data . . . 116
6.2.2 Data processing for progeny . . . 116
6.2.2.1 Graph construction . . . 117
6.2.3 Data processing for parents . . . 117
6.2.3.1 Graph construction . . . 117
6.2.3.2 Transfer of gene models . . . 119
6.2.3.3 Generation of visualization resources . . . 119
6.2.4 Quality control . . . 121
6.3 *De novo* mutations in a single 3D7xHB3 sample . . . 122
6.3.1 List of mutations . . . 122
6.3.2 Manual examination of all *de novo* mutations . . . 126
6.3.2.1 Event 1 (a-h): NAHR . . . 126
6.3.2.2 Event 2: unknown . . . 127
6.3.2.3 Events 3 and 4: two SNPs . . . 128
6.3.3 Rejected events . . . 128
6.3.4 Novel kmer accounting . . . 129
6.4 *De novo* mutations in all crosses and samples . . . 129
6.4.1 Novel kmer use . . . 129
6.4.2 Variants per sample and cross . . . 130
6.4.3 Manual examination of every event in the 3D7xHB3 cross . . . 131
6.4.4 Recovery of published *var* gene recombinations . . . 132
6.5 Summary . . . 143

7 *De novo* mutations in *Pan troglodytes verus* 144
7.1 Introduction . . . 144
7.2 Data processing . . . 145
7.2.1 Initial data . . . 145
7.2.2 Sample processing . . . 146
7.2.3 Novel kmers . . . 146
7.2.4 Additional filters for low-coverage data . . . 146
7.3 *De novo* mutations in the *P. troglodytes* dataset . . . 148
7.4 Summary . . . 149

8 Discussion **152**
8.1 Introduction . . . 152
8.2 Reference versus graph . . . 152
8.2.1 Indirect comparison: reference-based analysis . . . 152
8.2.2 Direct comparision: graph-based analysis . . . 153
8.3 Implementation notes . . . 154
8.3.1 Merits of the graphical approach . . . 154
8.3.2 Challenges of the graphical approach . . . 155
8.3.3 Addressing the implementation challenges . . . 157
8.3.4 Novel kmers and their effect on design considerations . . . 159
8.4 Further improvements . . . 160
8.4.1 Towards a probabilistic treatment of assembly graphs . . . 160
8.4.2 Removing graphical tangles . . . 161
8.4.3 Detecting inversions . . . 161
8.4.4 Other miscellaneous improvements . . . 162
8.5 Conclusion . . . 162

References **162**

Appendices **179**

A Manual examination of all DNMs in a single *P. falciparum* sample **180**
A.1 Introduction . . . 180
A.2 Screenshots . . . 180

List of Figures

1.1 a. Parental and child sequences at the site of a *de novo* mutation, and the kmers generated at $k = 3$. b. The resulting Venn diagram of kmers found exclusively in the parents, the child, or common to both. c. Novel kmers found around the genome indicate the presence of a *de novo* mutation. . . . 5

1.2 Kmer coverage distribution for a single 3D7xHB3 progeny, PG0063-C. Red line indicates non-parametric LOESS fit upon which the lower and upper coverage thresholds are chosen. 7

1.3 a. Novel kmers observed in the reference-based analysis vs novel kmers expected from the reference-free analysis. b. Reads that map uniquely to the reference genome ($MQ > 0$) versus mapping multiple times or not mapping at all ($MQ = 0$), conditioned on the read containing a "confident" novel kmer. 9

1.4 Venn diagram of kmers present in the 3D7 and HB3 reference genomes at $k = 47$. Both forward and reverse-compliment kmers are considered the same. 10

1.5 Presence and absence of unique kmers in three 3D7 *var* genes, laid out along the length of each gene. Each row represents the coding sequence (CDS) of three *var* genes in the 3D7 repertoire. Each vertical line represents a kmer from that gene. Colored kmers represent those unique 3D7 kmers that are recovered in the sample. Grey indicates no recovery - the kmer was absent from the sample. White indicates the kmer at that position was not unique in the 3D7 genome. Protein domain annotations from the VarDom server are overlaid onto the CDS and represented in different shapes.[1] Additional information about each gene (location in the genome, 5' telomere/3' telomere/core placement in the genome, upstream promoter class, and percentage of the gene's unique kmers recovered) are provided below the gene name. 12

1.6 The process of generating a de Bruijn graph representation of sequence data. a. The underlying genome. b. Reads sequenced from the genome (including one read with a sequencing error). Reverse-complement reads are not shown for clarity. c. The $k = 3$ de Bruijn graph reconstruction, including kmer coverage annotations. 14
1.7 Pipeline for making graph-based *de novo* mutation calls in a trio. 16

2.1 Lifecycle of parasites from the *Plasmodium* genus. Reprinted from the CDC's public domain Public Health Image Library (PHIL), image #3405. . 20

3.1 Workflow for generating the HB3 parental genome. a. Reads from HB3 sample, PG0052-C, are mapped to the 3D7 reference genome, and variants (SNPs and indels) are called and stored as a VCF file. b. We remove the 3D7 *var* gene repertoire, replacing each with a reasonable HB3 *var* counterpart, and encode the changes to the 3D7 reference genome as a VCF file. c. We alter the reference genome using Algorithm 2, thus producing the simulated HB3 genome. 33
3.2 Workflow for generating children's genomes. a. Generate chromosomes from the parental genomes (compatible *var* genes shown in green and orange). b. Recombine homologous chromosomes. c. Add gene conversion events (by incorporating variants from the alternative haplotypic background over a limited genomic window). d. Replace one of the *var* genes with a chimera of compatible genes. e. Add *de novo* mutations. 37
3.3 Empirical recombination frequencies per chromosome 38
3.4 Simulated haplotype mosaics for chromosome 12. Genomic position is shown at the top of the figure, while variant number is shown at the bottom. Each variant is depicted as a vertical grey line attached to the mosaic plot at the appropriate location. In the mosaic, every variant is color-coded by parent of origin. 39
3.5 Non-allelic recombinations for two compatible *var* genes. 40
3.6 Simulated variant types. a. Original, parental haplotypic background upon which variants will be placed. b. A single nucleotide polymorphism. c. An insertion of two nucleotides. d. A deletion of three nucleotides. e. An inversion of four nucleotides. f. Expansion of a 3 bp short tandem repeat (STR) by one unit. g. A contraction of an STR by one unit. h. A tandem duplication of 11 nucleotides. 40

3.7 Construction of the coverage profile for the reference genome and liftover to the altered genome. Each read start (and fragment start, not shown) is stored along with a count of the number of reads at that location that contain errors. This information is then lifted over to the simulated sequence, and gaps in the table are filled in with neighboring values. 42
3.8 Top: empirical fragment size distribution for PG0051-C. Bottom: empirical deletion (negative values) and insertion (positive values) length distribution for the same sample. 47
3.9 Read datasets for real (top panel), simulated perfect (middle panel), and simulated realistic (bottom panel) data. 49

4.1 Graph for a variant between a parent (blue edges) and a child (red edges), with novel kmer annotations in red. 55
4.2 An example directed graph with six vertices, six edges, and coverage information written on each edge. 55
4.3 Three example Cortex record entries, each showing a kmer, coverage, and edge information across three samples. 57
4.4 A simple variant motif for a C to G SNP. a. Haploid sequences from a mother (green), father (blue), and child (red), the last differing from the first two by a single base substitution. b. The resulting multi-color de Bruijn graph for $k = 3$. Red vertices denote kmers that are deemed "novel", i.e. present in the child and absent in the parents. Edge colors reflect the samples in which the connected pairs of kmers are found. Edges that are part of the bubble (variant call) are displayed with thicker lines. 62
4.5 A multi-color de Bruijn graph at $k = 47$ for a haploid pedigree spanning a simulated *de novo* SNP, produced by our VisualCortex software. Vertex labels have been supressed for clarity. Spatial layout is arbitrary and for display purposes only. 63
4.6 A 5 bp insertion in the child (red), with the two parents (blue and green) sharing a homologous and distinct allele, evidenced by the shared path taken by the parental samples and unshared path taken by the child. 63
4.7 A 5 bp deletion in the child, with the two parents (blue and green) sharing a homologous and distinct allele, evidenced by the shared path taken by the parental samples and unshared path taken by the child. 64
4.8 A *de novo* tandem duplication in the child, with parental and child sequences shown in inset above. 65

4.9 A variant wherein the child's path does not simply diverge from that of the parents, but rather navigates both. 66
4.10 A gene conversion event in the child, switching between mother (green), father (blue), and back to mother. Parental and child sequences in the inset. For illustrative purposes, kmer size is 7. 67
4.11 Indel with sequencing errors in graph. a. Selected extraneous branches expanded for illustrative purposes b. All extraneous branches collapsed. . 68
4.12 Kmer coverage histogram for a real sample, with LOESS fit 71
4.13 Removal of contaminating kmers; *P. falciparum* kmers are shown in green; the putative contaminants are shown in orange. 72
4.14 Decision tree for *de novo* variant classification. 82
4.15 Confusion matrix for observed events (below) versus expected events (right), in simulated perfect data. 90
4.16 Confusion matrix for observed events (below) versus expected events (right), in simulated realistic data. 91

5.1 Coverage for Illumina data (orange) and PacBio data (red) of the same sample: PG0051-C (the reference isolate, 3D7). a. Coverage over the chromosome 4 centromere. b. Coverage over the 5′ subtelomere of chromosome 1. Genes from the *var*, *rifin*, and *stevor* antigenic gene families are highlighted as red, orange, and blue rectangles, respectively. 95
5.2 Alignment of contigs from PG0051-C to 3D7 reference assembly, with sequences from reference and draft assembly on the x and y-axis, respectively. Each aligned section of a contig from the draft assembly is represented by a line segment terminated at either end with circles. Red denotes forward alignment of contigs to the reference, blue denotes reverse alignment. . . . 96
5.3 Alignment of contigs from PG0446-C to 3D7 reference assembly, with sequences from reference and draft assembly on the x and y-axis, respectively. Each aligned section of a contig from the draft assembly is represented by a line segment terminated at either end with circles. Red denotes forward alignment of contigs to the reference, blue denotes reverse alignment. . . . 98

5.4 Placement and orientation of exons from the reference assembly mapped to the PG0446-C assembly. Chromosomes 1, 5, and 9 shown. Each point represents an exon aligned to both the reference and PG0446-C assembly. Green indicates matching orientation of the exon's alignment in both assemblies. Red indicates a forward alignment in one, reverse-complement alignment in the other. Crosses indicate exons from a different chromosome than the one indicated in the plot title. 100
5.5 Haplotype mosaic for chromosome 14 in the 803xGB4 cross dataset. Parental samples are identified with asterisks. For remaining samples, alleles are color-coded according to the parent from which they are apparently inherited - orange for "mother" (PG0443-C, the 803 isolate), blue for "father" (PG0050-CX2, the GB4 isolate). Missing data (sites where no genotype could be ascertained) are shown in white. Bottom axis represents variant number. Top axis represents variant position along the length of the chromosome. Arrow on right-hand side indicate samples that are apparently clonal copies of one another. 101
5.6 a. Kmer coverage in a 3D7xHB3 sample, with LOESS curve and calculated lower threshold at curve's first local minimum shown in red. b. Kmer coverage in the 803xGB4 sample, PG0446-C, with LOESS curve and calculated lower threshold at curve's first local minimum shown in red. c. Kmer coverage distributions for all novel kmers in PG0446-C, conditional on the kmer being present or absent in the PacBio assembly. Calculated lower threshold from panel b shown as red dotted line. 102
5.7 Local subgraphs in the Illumina data for PG0446-C near novel kmers that are absent in the corresponding PacBio assembly. Red circles denote "trusted" novel kmers (those that passed our initial coverage and contamination checks). Red crosses indicate "untrusted" novel kmers (those that passed our initial coverage check but failed the contamination check). a. An "orphan" branch: a series of novel kmers not connected to the parental graphs. b. A "bushy-tailed" branch: a series of novel kmers that appear to connect to everything, including the parental graphs, by chance. 103
5.8 The results of filtering kmers private to the PG0446-C child 104

5.9 IGV screenshot of an A to G SNP, showing reads and assembly information for the trio, aligned to the PacBio PG0446-C assembly. Top panel: 803. Middle two: GB4 reads and PacBio assembly (sliced to facilitate alignment). Lower two: PG0446-C reads and contig containing the variant in question from the a. panel. 106

5.10 A C to T SNP that has likely not reached fixation in the sequenced population of ostensibly clonal parasites. a. Local subgraph at the site, wherein the child's graph contains paths that traverse the series of novel kmers as well as the parental kmers, indicating polymorphic status. b. IGV screenshot of the site showing reads and assembly information for the trio, aligned to the 3D7 reference genome. Top panel: positions of the called variants in the reference-basde analysis. Second panel: PG0446-C (803). Third: PG0050-CX2 (GB4). Fourth: PG0446-C (child). Fifth: Uncorrected PacBio reads from PG0446-C. Bottom: gene model track from PlasmoDB 9.0. Stacked barplots above tracks indicate the proportion of reads supporting each allele. 107

5.11 A strange event involving a single novel kmer. a. Local subgraph at the site; the novel kmer in question is depicted as a filled red vertex in the right side of the image. b. IGV screenshot of the site showing reads and assembly information for the trio, aligned to the PacBio PG0446-C assembly. 108

5.12 A validated *de novo* SNP, recovered in the child amidst a great deal of sequencing error. Top panel: PG0443-C (803). Middle panel: PG0050-CX2 (GB4). Lower panel: PG0446-C (child). 112

5.13 False-positive *de novo* variants in the 3′ subtelomeric region of chromosome 2. Top panel: positions of the called variants in the reference-based analysis. Second panel: PG0443-C (803). Third: PG0050-CX2 (GB4). Fourth: PG0446-C (child). Fifth: Uncorrected PacBio reads from PG0446-C. Bottom: gene model track from PlasmoDB 9.0. Stacked barplots above tracks indicate the proportion of reads supporting each allele. Reads with ambiguous alignments are displayed in a lighter shade. 113

6.1 Relationships for samples in four *P. falciparum* crosses, with the number of QC-passing Illumina datasets for progeny shown. Green and orange shading indicates the availability of finished and draft assemblies, respectively. Orange stripes denote the availability of a draft assembly genome for a single progeny, used as validation for *de novo* events. Gender assignments for the parents are arbitrary, intended only to simplify discussion. 115

6.2 Number of kmers and assembly length in the 803xGB4 samples. Vertical dashed lines indicate the mean value of the metric in other crosses. 122
6.3 Circos visualization of all *de novo* mutations in PG0063-C, positioned on the reference sequence (outer ring), aligned maternal assembly (middle), and aligned paternal assembly (inner). Unplaced parental contigs are concatenated and placed in the "U" pseudochromosome. Each parental ring is annotated with a sequence compressibility score for every 2,500 bp shown above their ideograms, while the reference ring is annotated with mean read coverage per 2,500 bp. Within the reference ideogram, all reference-based variant calls made in this sample from the MalariaGen project are shown, color-coded to reflect their parentage. Regions that were masked out of MalariaGen's variant calling process are shown as grey bars. Variants are shown as glyphs below the parental ideograms, shape-coded for type (small circle: unknown, large circle: SNP, triangle: indel, square: MNP, diamond: NAHR event). The color and positioning of the glyph reflects the haplotypic background upon which the event occurred. Interchromosomal structural variation and gene-conversion events are additionally denoted as arcs linking the relevant parts of the genome. The closest gene to the event is reported in black if the mutation falls within the gene and as grey if it falls outside the gene. 134
6.4 Graph motifs for discarded events. a. Disconnected events: novel kmers that fail to link back to the genome. b. "Dirty" events: the dirty kmers (shown in grey with red edges) make up more than 10% of the novel kmers in the region. 135
6.5 Cumulative accounting of novel kmers and the mutation types to which they correspond. Events passing filters are depicted in green. Those failing filters are in red. 135
6.6 Novel kmers per sample and cross. Left: number of novel kmers in each sample, after filtering. Right: distribution of novel kmers per sample in each cross. 136
6.7 Novel kmer usage by variants per sample and cross. Left: novel kmers used in variants (green), in unknown events (orange), and events failing filters (red). Right: fraction of novel kmers used by variants (green), unknown events (orange), and events failing filters (red). 137

6.8 All monomorphic, typed *de novo* variants in the 3D7xHB3 cross (untyped events suppressed). Outer ring: chromosome ideogram from the 3D7 reference genome. Middle ring: mapping of chromosomes from the 3D7 parental assembly (finished genome; identical to the 3D7 reference) to the reference. Inner ring: mapping of HB3 parental assembly (draft genome from PacBio sequencing) to the reference. SNPs, MNPs, and indels are depicted by circles, squares, and triangles, respectively. NAHR events shown as arcs linking the involved chromosomes; thickness of each arc is proportional to the contig length. 138

6.9 All monomorphic, typed *de novo* variants in the HB3xDD2 cross (untyped events suppressed). Outer ring: chromosome ideogram from the 3D7 reference genome. Middle ring: mapping of chromosomes from the HB3 parental assembly (draft genome from PacBio sequencing) to the reference. Inner ring: mapping of DD2 parental assembly (draft genome from PacBio sequencing) to the reference. SNPs, MNPs, and indels are depicted by circles, squares, and triangles, respectively. NAHR events shown as arcs linking the involved chromosomes; thickness of each arc is proportional to the contig length. 139

6.10 All monomorphic, typed *de novo* variants in the 7G8xGB4 cross (untyped events suppressed). Outer ring: chromosome ideogram from the 3D7 reference genome. Middle ring: mapping of chromosomes from the 7G8 parental assembly (draft genome from PacBio sequencing) to the reference. Inner ring: mapping of GB4 parental assembly (draft genome from PacBio sequencing) to the reference. SNPs, MNPs, and indels are depicted by circles, squares, and triangles, respectively. NAHR events shown as arcs linking the involved chromosomes; thickness of each arc is proportional to the contig length. 140

6.11 All monomorphic, typed *de novo* variants in the 803xGB4 cross (untyped events suppressed). Outer ring: chromosome ideogram from the 3D7 reference genome. Middle ring: mapping of chromosomes from the 803 parental assembly (low-quality draft genome from Illumina sequencing) to the reference. Inner ring: mapping of GB4 parental assembly (draft genome from PacBio sequencing) to the reference. SNPs, MNPs, and indels are depicted by circles, squares, and triangles, respectively. NAHR events shown as arcs linking the involved chromosomes; thickness of each arc is proportional to the contig length. 141

6.12 Recovery of diagnostic kmers in 3D7 and HB3 *var* genes. White regions indicate kmers that were not diagnostic. Vertical grey indicates a diagnostic kmer that absent in the progeny, while vertical colored lines indicate the kmer's presence. a. Two genes in PG0020-C purported to have an NAHR event in Claessens *et al.*, but exhibiting no kmer loss indicative of an NAHR event. b. Two genes in PG0034-C with missing diagnostic kmers, possibly indicative of an NAHR event. c. Two genes (DD2var13 and DD2var18) which exhibit very slight recovery or loss of diagnostic kmers towards their terminal ends. 142

7.1 The three-generation chimpanzee pedigree. The highlighted quintet is discussed in this chapter. Average read coverage shown in parentheses. 144
7.2 A putative and suspicious *de novo* SNP in child Ruud with very few novel kmers (filled red circles), and many non-novel kmers (filled grey circles) along the alternate branch. An example kmer is highlighted; its coverage (c), incoming edges (i), and outgoing edges (o) are shown in the inset table. 147
7.3 Positions of *de novo* mutations from the reference-based analysis (left of ideogram) and graph-based analysis (graph-based analysis). 148

A.1 IGV screenshot of PG0063-C event 1 (a-c), SNP and two MNPs: three apparent *de novo* mutations within a much larger NAHR event. Top panel: 3D7 parent. Middle panel: HB3 parent. Lower panel: PG0063-C child. Variant positions are shown as red blocks at the top of the three panels below the genome position track. Stacked barplots above variant positions indicate the proportion of bases supporting an alternate allele to the reference. . . . 181
A.2 Graph screenshot of event 1 (a-c), SNP and two MNPs: three apparent *de novo* mutations within a much larger NAHR event shown as two bubbles with novel kmer involvement in the upper and middle-left sections of the plot. 182
A.3 IGV and graph screenshots of event 1 (d-h), a NAHR event. a. Graphical view of the exchange; chromosome of origin specified for various regions. b. IGV screenshot for 3D7 (top), HB3 (middle), and PG0063-C (child) on the chromosome 1 region of the event. c. IGV screenshot of the chromosome 2 region of the event. 183
A.4 Event 1 h: a *de novo* SNP in a *var* on the 3D7 haplotypic background. Top panel: 3D7. Middle: HB3. Lower: PG0063-C child. 184

A.5 Event 2, a graph "tangle": a region of the graph where the in/out degree of vertices within a small radius sharply increases, resulting from attempting to traverse a low-complexity region of the genome. 185
A.6 IGV screenshot of event 2, a low-complexity region of the genome containing many reads with many apparent errors. 186
A.7 Events 3 and 4, two SNPs, non-obvious from the alignments, but apparent in graph. a. IGV screenshot of two *de novo* SNPs in a *var* gene. b. Graphical view of the two SNPs. 187
A.8 The true home of the events 3 and 4, two SNPs misplaced during alignment. 188

List of Tables

1.1 Phenotypes of *P. falciparum* isolates used for genetic crosses (+ to indicate exhibiting phenotype, - to indicate phenotype absence, blank to indicate untested and irrelevant to studied crosses). 3
1.2 Theoretical percentage of a 23 Mbp genome observed at a target depth of coverage. 6

3.1 Assembly statistics on publicly available finished and draft *P. falciparum* references, ordered by scaffold N50 length. Parents are shown in bold. . . . 34
3.2 Variants found the HB3 (PG0052-C) sample from the 3D7xHB3 dataset. . . 34
3.3 *Var* gene replacements from 3D7 to HB3 repertoire. 36
3.4 Example read and fragment scalar properties for sample PG0063-C. 46
3.5 *De novo* variant counts for each of the 20 simulated children. 50

4.1 Comparison of assembly statistics of the finished 3D7 reference genome and a reconstruction of the same sample from 76-bp Illumina data. 53
4.2 ROC metrics on simulated perfect data 88
4.3 ROC metrics on simulated realistic data 89

5.1 Assembly statistics for PacBio RS II data on PG0051-C and PG0446-C isolates 94
5.2 Apparent errors per chromosome in the PG0051-C assembly 97
5.3 Callset metrics in the PG0446-C sample of the trio 109

6.1 Summary of sequencing data for four *Plasmodium falciparum* crosses 116
6.2 Additional data availability for all *P. falciparum* cross parents 117
6.3 Statistics on all parental assemblies 120
6.4 *De novo* variants in 3D7xHB3 progeny, PG0063-C 124
6.5 Monomorphic variants and rates of occurrence per cross and per sample . 131
6.6 NAHR events in Claessens *et al.* and this study 133

7.1 Summary of *P. troglodytes* quintet data 146

7.2 Variants found in Ruud in the reference- and graph-based analyses. 151

List of Algorithms

1 Given a small set of variants, generate all possible subsets of variants . . . 8
2 Generate an alternative reference sequence based on a VCF file. 32
3 Emit all fragment starts, read starts, and error rates per position. 43
4 Lift a table from reference to child coordinates. 44
5 Get next kmers for a color, c, from CortexRecord, rec 59
6 Get previous kmers for a color, c, from CortexRecord, rec 59
7 Get next kmers for a color, c, matched to expected orientation 60
8 Get next kmers from the clean graph, failing back to the dirty graph 61
9 A basic, iterative depth-first search 74
10 The recursive depth-first search with arbitrary stopping conditions 76
11 Child's traversal success determination method 77
12 Child's traversal failure determination method 77
13 Parents' traversal success determination method 78
14 Parents' traversal failure determination method 78
15 Annotating possible variant starts and ends of the subgraph 79
16 Finding the shortest path in a graph 80
17 Trim back haplotypes to reveal alleles 81
18 Stretch has switches 83
19 Stretch has chimeras 83
20 Score variant 84
21 Evaluate variant 86

1 Motivation

1.1 Introduction

Novel, inherited variation arises through spontaneous mutations, rare genomic variants occurring within germ cells or the fertilized egg itself. These events are termed *de novo* mutations, or DNMs. DNMs are a class of mutations particularly important in the study of pathogens. For example, a single serine to asparagine amino acid substitution (S31N) in influenza A alters the properties of the M_2 ion channel,[2] interferes with the action of the adamantane class of antiviral drugs, and has reached fixation in the population.[3] In *Mycobacterium tuberculosis*, a four-year *in vitro* drug challenge experiment on nine drug-susceptible isolates from a single patient demonstrated the strain could acquire resistance to nearly all first-line and most second-line drugs through just 12 single nucleotide variants (SNVs).[4] *Staphylococcus aureus* is the most common cause of post-operative infection worldwide and is increasingly unresponsive to β-lactam treatment and drugs in the penicillin group (methicillin, dicloxacillin, nafcillin, oxacillin, etc.). Sequencing of methicillin-susceptible (MSSA) in the penicillin group and resistant (MRSA)[1] strains reveals the acquisition of resistance genes via mobile genetic elements and horizontally transferred genomic islands (e.g. the SCC*mec* cassette chromosome upon which the methicillin and other β-lactam antibiotic resistance gene, *mecA*, resides).[5]

That DNMs are present in a child's genome and absent from their parents' genomes should make them straightforward to detect, at least in principle. Modern experiments on second-generation sequencing platforms can now produce billions of short 75 to 150-bp reads during each instrument run. These experiments are relatively inexpensive, making it feasible to generate moderate to high coverage data for members of a pedigree in mere hours or days. Aligned to a reference, a canonical representation of a genome for a single member of the species, consistent differences at a single genomic position within reads that span the locus reveal genomic variants. Sufficiently high coverage helps statistical

[1] The term "methicillin resistant" is used in the literature, though it is commonly taken to mean all drugs, including more stable variants of methicillin used in modern clinical practice.

software to distinguish sequencing error from true variation, and searches for alleles present in a child, but absent from its parents, can yield a candidate set of DNMs.

This protocol, widely adopted and applied in a variety of human and non-human datasets,[6–12] makes a tacit assumption: the reference genome and a new sample's genome should be nearly identical in sequence. This hypothesis is necessary and is typically reasonable: when working with samples having a low heterozygosity, most sites in the genomes of two or more samples should be identical, and occasional differences are small and easily handled by read-to-reference alignment methods that are capable of tolerating a small amount of heterology.

However, there are plenty of loci within genomes that do not meet this assumption. Different strains of the uropathogenic bacterium *Escherichia coli* are reported to have strain-specific genomic islands containing virulence factors that are entirely absent from the laboratory strain *MG*1655 used as the basis for the reference sequence.[13] In assembled samples from the diploid coccolithophore *Emiliania huxleyi*, 8 to 40 Mbp of the approximately 142 Mbp genome was shown to be isolate-specific; 25% of genes in the reference strain, CCMP1516, were not found in the other 13 examined isolates.[14] The human major histocompatibility complex (MHC), a 163 Mbp region of chromosome 6 that encodes genes critical to innate and adaptive immunity,[15,16] exhibits tremendous population diversity[17] and can generally not be recovered by simple alignment to the reference, but *can* be reconstructed with high accuracy by aligning data to an augmented version of the reference with alternative loci included.[18]

When a haplotype represented in a canonical reference sequence differs substantially from a sample under study, sequenced reads spanning gross differences may behave in two different ways: either they fail to align anywhere in the reference, or they align incorrectly. In the former case, interesting inter-sample variation in those regions is lost. In the latter case, the incorrect alignments may generate false-positive variant calls. This constitutes a reference bias: samples similar to the reference align properly and their genomic content can be analyzed easily. Those sufficiently dissimilar cannot.

1.2 *De novo mutations (DNMs) in Plasmodium falciparum*

To see how reference bias may affect DNM discovery, let us consider a second-generation sequencing dataset on experimental crosses of *Plasmodium falciparum* parasites, the etiological agent for the most deadly form of malaria.[19] These crosses have enabled the discovery of a number of virulence factors. The drug resistance and infectious phenotypes for various strains of *P. falciparum* are listed in Table 1.1. Crossing of the pyrimethamine-resistant 3D7 and sensitive HB3 strains[20] revealed a non-synonymous point mutation in

Table 1.1: Phenotypes of *P. falciparum* isolates used for genetic crosses (+ to indicate exhibiting phenotype, - to indicate phenotype absence, blank to indicate untested and irrelevant to studied crosses).

	3D7	HB3	DD2	7G8	GB4	803
pyrimethamine sensitivity	-	+				
chloroquine sensitivity		+	-			
infects mosquitoes easily		+	-			
infects *Aotus nancymaae*				-	+	-
artemisinin sensitivity					+	-

the *dhfr-ts*[2] gene, inhibiting binding of (and thus conferring resistance to) the drug.[21] Analysis of the HB3xDD2 cross,[22] the latter of which is resistant to chloroquine, localized the determinant to a previously undetected gene on chromosome 7, labelled *pfcrt*[3], a member of a new family of transporters. Additional investigation into differences in mosquito infection efficacy revealed down-regulation of the *pfmdv-1*[4] gene,[23,24] disruption of which results in marked reduction of mature and functional male gametocytes.

Cytoadherence and antigenic properties of parasites facilitate evasion from host immune attack, and can differ substantially from the properties of their progenitors. In 2000, Freitas-Junior *et al.* showed that some 3D7xHB3, HB3xDD2, and HB3xHB3 progeny harbored non-parental forms of subtelomeric *var* genes, key members of an antigenic gene family.[25] Notably, these altered forms do not segregate with the parents; they were likely generated during mitosis by non-allelic homologous recombination[5] (NAHR) of telomeres from two different chromosomes.[26] The resulting genes are novel, functional, and never before observed by the host immune system.

These mutations are critical in malaria parasite's exploration of the fitness landscape. NAHR can further diversify a pathogen's antigenic repertoire, enabling continued evasion of immunological actors. As parasites are exposed to new drugs, spontaneous duplications of a transporter gene may enable faster drug clearance in a parasite, thus conferring resistance. *De novo* point mutations may alter the binding site of a drug to a receptor, thus conferring immunity.

1.2.1 A reference-based approach for DNM discovery and allele identification

Many groups have released sophisticated software packages to facilitate the discovery of *de novo* mutations, along with detailed instructions on their use.[27–29] The fundamen-

[2] dihydrofolate reductase-thymidylate synthase
[3] *P. falciparum* chloroquine resistance transporter
[4] *P. falciparum* male gametocyte development gene 1
[5] sometimes referred to as "ectopic" (aberrant) recombination in the literature

tal premise of all of these methods is the same: reads from a pedigree are mapped to a high-quality reference genome[6]. Alignment artifacts (e.g. PCR duplication, misalignment in the presence of indels) are mitigated (via de-duplication and local realignment, respectively). Variants are discovered by examining reads spanning a locus for consistent discrepancies from the reference. The entire pedigree is examined simultaneously and read evidence is compiled to compute genotype likelihoods for each sample. Depending on the software package, these likelihoods are either computed independently or with knowledge of the pedigree, the latter of which may reweight the data by considering how alleles are transmitted from parents to offspring. Finally, any variant confidently present in the child, absent in the parents (and, depending on study design, present in later generations, implying occurrence in the germline and transmission to subsequent offspring) are retained as the set of candidate DNMs.

Variant calling software has been successfully applied to many sequencing datasets for the discovery of DNMs. These variant callsets have provided insight into the development of drug resistance,[31] the genetic architecture of some common diseases,[8] and mutational rates in humans and chimpanzees with a strong paternal age effect.[6,9,11,12]

1.2.2 *A reference-free approach for assessing DNM sensitivity*

Specificity and sensitivity are crucial metrics to consider for any variant callset. There are many approaches to establishing the specificity of a DNM callset. For select candidate variants (typically on the order of ~100 variants from a callset), Sanger sequencing, Sequenom assays, and even targeted third-generation sequencing (e.g. PacBio sequencing, which can currently generate reads up to ~$50,000$ bp) have been used successfully to validate mutations. Establishing sensitivity is more difficult, and there are a number of strategies to choose from. One could produce reference genomes for both parents and each child in which the mutation calls are made. This is prohibitively expensive for large (gigabase) genomes, but feasible for small (megabase and smaller) genomes. Another strategy is to produce simulations of realistic data. However, producing realistic data is quite a challenge in and of itself (one we address in Chapter 4), and may not provide a reliable sensitivity measure for real data.

Alternatively, it may be possible to establish the sensitivity of the reference-based protocol by considering how DNMs alter the genome of a sample with respect to its progenitors. Consider a site where a DNM - say, a single SNV - has occurred, as depicted in Figure 1.1. Although a single base of the genome has been altered, when the genome is

[6]Sometimes this is augmented with "decoy sequences" to reduce artifacts from common contaminants like Epstein-Barr virus, commonly used in the immortalization of human lymphocytes used to establish human cell lines.[30]

divided into fixed-length words of length k, or "kmers", we find k kmers that are present in the child but absent in the parents. In this manner, DNMs can be considered generators of "novel" kmers - kmers present in the child but absent in the parents. These kmers can be used as a signal to indicate the presence of a *de novo* variant. By choosing k to be reasonably large so as to avoid analyzing short sequences that pervade the genome (in our work, $k = 47$ bp; this parameter is discussed further in Chapter 4), we can simply count continuous stretches of novel kmers as a proxy for the number of DNMs.

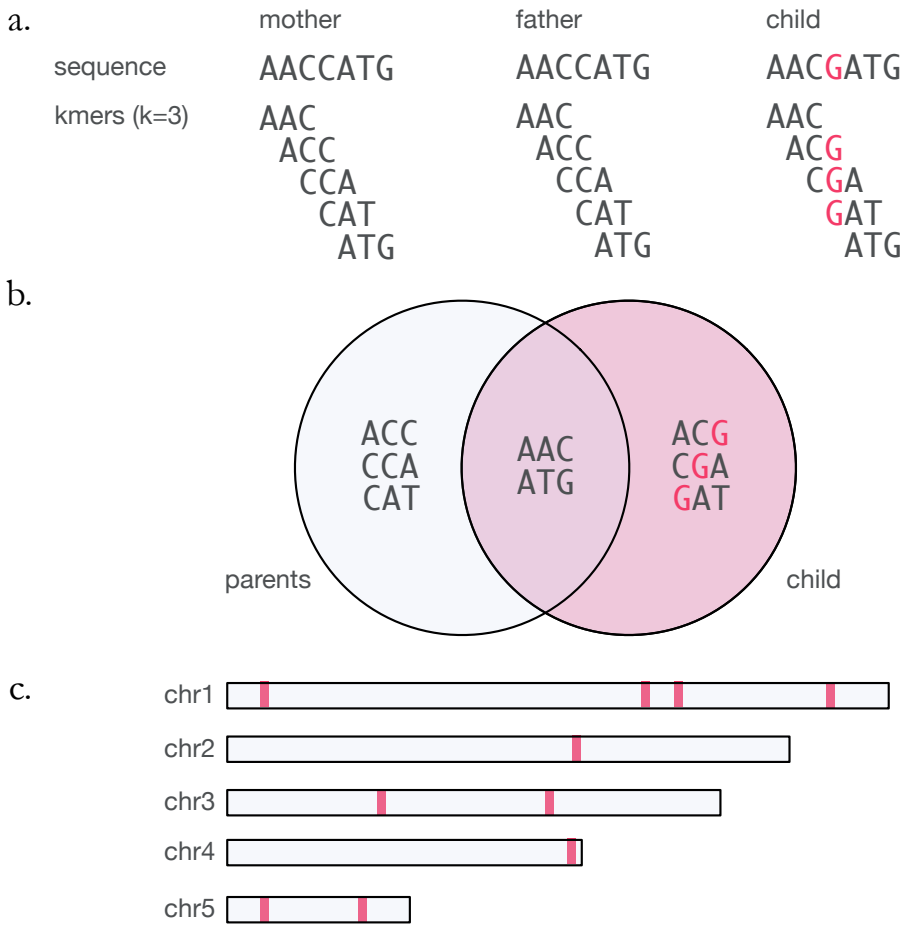


Figure 1.1: a. Parental and child sequences at the site of a *de novo* mutation, and the kmers generated at $k = 3$. b. The resulting Venn diagram of kmers found exclusively in the parents, the child, or common to both. c. Novel kmers found around the genome indicate the presence of a *de novo* mutation.

This approach gives us a powerful, reference-free mechanism to verify the results of the reference-based analysis. Sequencing of the whole genome is independent of any reference sequence that may already exist for the sample, and given sufficient coverage (Table 1.2 shows the requirements, assuming 76 bp reads and a 23 megabase genome,

Table 1.2: Theoretical percentage of a 23 Mbp genome observed at a target depth of coverage.

coverage	reads	% genome (ideal)	% genome (realistic)
1	302,631	63.21	57.17
2	605,263	86.47	76.77
3	907,894	98.17	84.92
4	1,210,526	98.17	89.10
5	1,513,157	99.33	91.55
6	1,815,789	99.75	93.13
7	2,118,421	99.91	94.28
8	2,421,052	99.97	95.15
9	2,723,684	99.99	95.82
10	3,026,315	100.00	96.34
11	3,328,947	100.00	96.76
12	3,631,578	100.00	97.11
13	3,934,210	100.00	97.39
14	4,236,842	100.00	97.63
15	4,539,473	100.00	97.84
16	4,842,105	100.00	97.99
17	5,144,736	100.00	98.14
18	5,447,368	100.00	98.28
19	5,750,000	100.00	98.40
20	6,052,631	100.00	98.50

computed for theoretical data and realistically simulated data, see Chapter 3), the raw data from a sequencing experiment should contain the full set of DNM-generated novel kmers, regardless of any mapping issues.[32] Taking the reads that map, calling *de novo* variants, and extracting the novel kmers from the immediate vicinity should theoretically reproduce that set. Comparing the expected (reference-free) set to the observed (reference-based) kmer set should thus provide the sought sensitivity measure.

1.3 DNM sensitivity of the reference-based approach

We now test these ideas on part of a real dataset that will be used throughout this dissertation: experimental crosses between malaria parasites (*Plasmodium falciparum*). For simplicity, we shall focus on 17 samples from the 3D7xHB3 experimental cross, sequenced to an average fold-coverage of 105 ± 34, using 76 bp reads.

1.3.1 Data processing

We first generated the list of expected novel kmers by processing the raw sequencing data for each sample with the *de novo* assembly software, Cortex[33] at $k = 47$, standard

settings for other parameters, and no error cleaning. We produced the initial list of putative novel kmers by selecting kmers present in a child but absent in both parents. We further filtered this list in a number of ways. First, we imposed lower and upper kmer coverage thresholds, automatically determined by a custom algorithm designed to find lower and upper coverage thresholds on the LOESS regression of the coverage distribution, as shown in Figure 1.2 (more details in Chapter 4). Additionally, we removed kmers originating from possible contaminants, as determined by performing a BLAST search on the confident list, removing all non-*Plasmodium* hits, and any putatively novel kmers connected to them in the graph. More details on these steps are presented in Chapter 5. These steps ensure that the list of novel kmers is very restrictive.

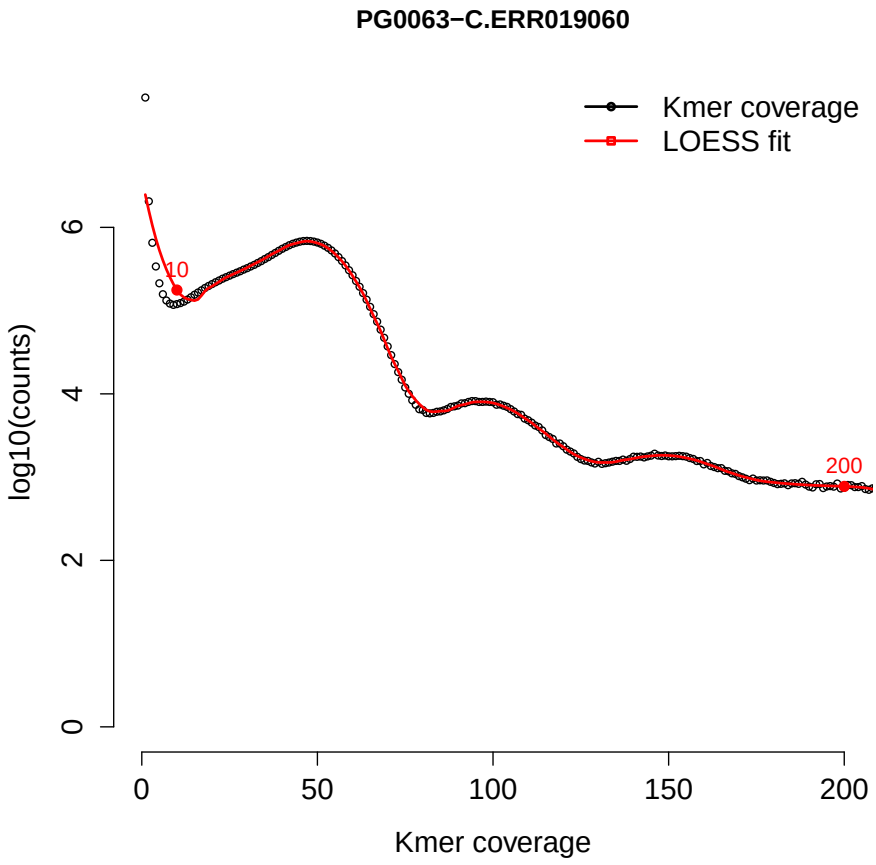


Figure 1.2: Kmer coverage distribution for a single 3D7xHB3 progeny, PG0063-C. Red line indicates non-parametric LOESS fit upon which the lower and upper coverage thresholds are chosen.

We aligned each sample's reads to the PlasmoDB 9.0 release of the *Plasmodium falciparum* genome[34,35] using BWA-MEM.[36] We followed data processing guidelines as specified in the Genome Analysis Toolkit (GATK) best-practices documentation,[27,37] flagging

duplicate reads so that they are ignored downstream, and recalibrating base quality scores using the MalariaGen data release on the crosses as a truth set.[19] As recent guidance by the authors indicates the GATK's variant calling software has internalized the local indel realignment functionality, we opted to forego running this step separately. We called variants across all 18 samples (parents and progeny) simultaneously using the GATK's HaplotypeCaller with standard settings and a ploidy of 1.

Algorithm 1 Given a small set of variants, generate all possible subsets of variants

```
1:  function generateAllPossibleSubsets(variants)
2:      los ← []
3:      for i ← 0 to length(variants) do
4:          lo ← []
5:          for j ← 0 to length(variants) do
6:              if i ≠ j then
7:                  lo.add(variants[j])
8:          if lo.size() ≥ 0 then
9:              los.add(lo)
10:         if lo.size() ≥ 1 then
11:             generateAllPossibleSubsets(lo)
12:     return los
```

A complete and perfect variant callset details the alterations that must be performed on the reference sequence in order to generate the genome of the sequenced sample. Unfortunately, it is typically not possible to obtain a perfectly sensitive and specific callset. False positives and false negatives proximal to true positives may interfere with our ability to generate the true underlying haplotype sequence. To bypass this problem, we did not filter the variant callset. Instead, we combinatorically generated all possible subsets of variants found within 100 bp of each other using a recursive strategy to leave single elements out of a given set of kmers, presented in Algorithm 1. This will certainly generate vastly more kmers than truly exist in the sample's genome. However, as we are only interested in verifying that the variant-induced kmers are present in our novel kmer set, this approach has the benefit of providing maximum sensitivity.

1.3.2 Results

Figure 1.3a shows the results of our reference-based vs reference-free analysis. Per sample, our restrictive reference-free analysis has generated hundreds of novel kmers to explain. However, the reference-based analysis recapitulates only a fraction of these - about 36% on average.

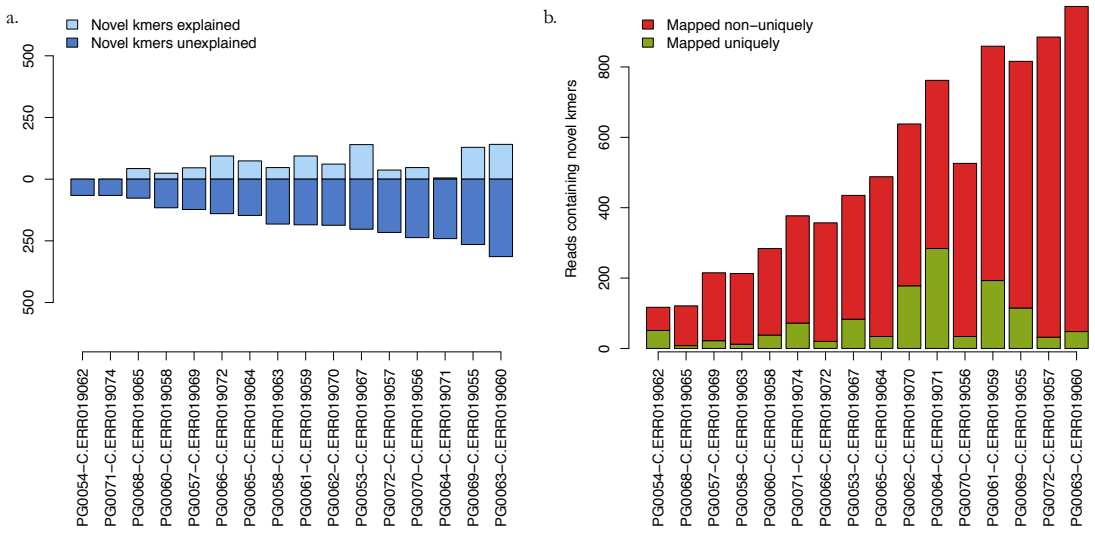


Figure 1.3: a. Novel kmers observed in the reference-based analysis vs novel kmers expected from the reference-free analysis. b. Reads that map uniquely to the reference genome ($MQ > 0$) versus mapping multiple times or not mapping at all ($MQ = 0$), conditioned on the read containing a "confident" novel kmer.

Attempting to explain where these missing kmers have gone, we searched the reads for every novel kmer. On average, more than 80% of reads containing these kmers were found to map to multiple locations in the genome (summarized per sample in Figure 1.3b).

In both measures, we found considerable variation in the novel kmers explained or unexplained, and in reads mapped uniquely or non-uniquely. This variance is likely caused by multiple sources. For example, in PG0063-C, structural variants known to exist between antigenic genes (discussed in greater detail below) cannot be captured by the variant caller we used (and are exceedingly difficult to detect in short read data). This uncaptured structural variation will contribute to some of the unexplained variance. Furthermore, despite being as restrictive as possible in order to avoid some obvious confounds (contamination, sequencing error), our filters may not be functioning as well as we'd hope. Some data contamination may have escaped our filters, which is possible if the contaminant is not in the BLAST database. This would thus contribute some apparently novel and unexplaind kmers to our analysis. Additionally, our attempts to remove sequencing errors may have also been confounded by *recurring* errors, as these will likely not share the property of being low-coverage kmers. Such errors typically have extreme coverage (e.g. $5,000x$), as they tend to originate from long homopolymer runs that are frequently misread by Illumina sequencers. Of course, it's possible that such a recurring

error could manifest at more modest coverage (e.g. $50x$).

1.4 Discussion

1.4.1 Failure of the mapping approach

The preponderance of reads containing novel kmers fail to map uniquely to the reference genome, explaining why we should find such a massive discrepancy between the novel kmers we find versus explain - reference-based calling approaches cannot call variants on poorly placed reads. That there should be so many reads that fail to map is perhaps not surprising, given the divergent nature of the reference genome to other samples. Figure 1.4 shows the overlap between kmer sets between the 3D7 and HB3 reference genomes. More than 20% of the total set of kmers between these two samples is unique to each sample.

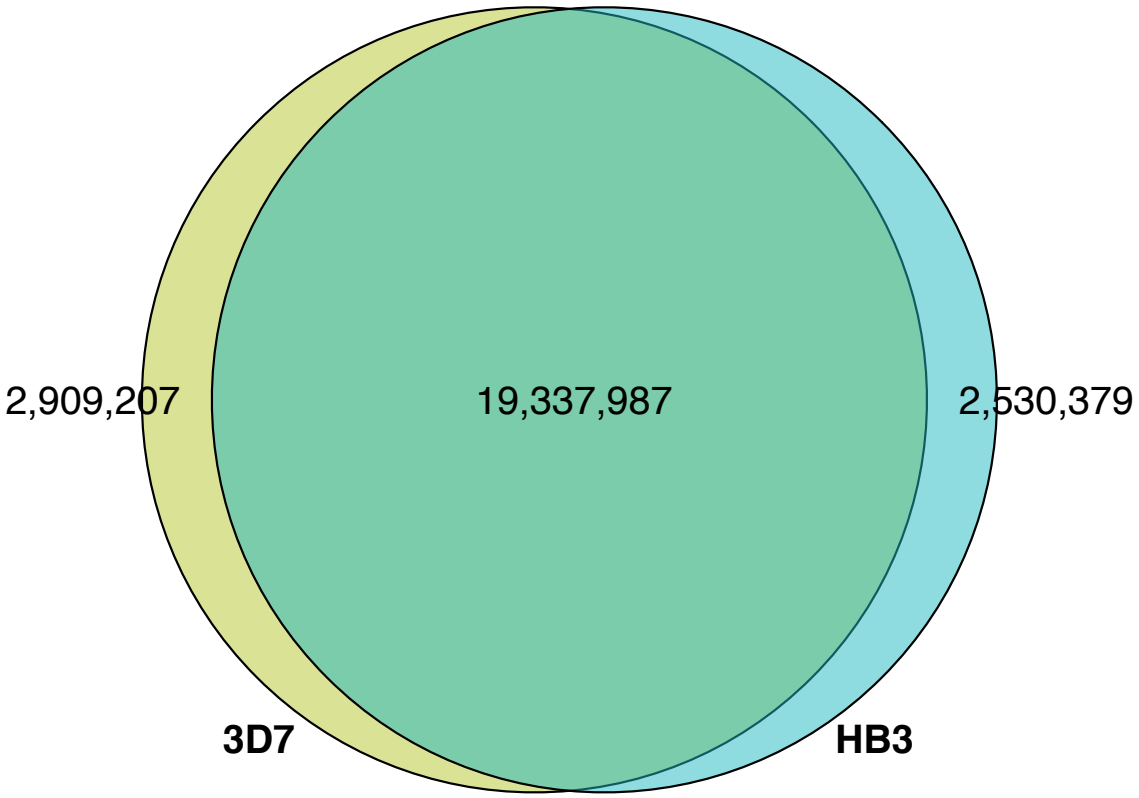


Figure 1.4: Venn diagram of kmers present in the 3D7 and HB3 reference genomes at $k = 47$. Both forward and reverse-compliment kmers are considered the same.

These unique kmers are not simply repetitive, intergenic, or otherwise uninteresting. They often reflect interesting biology. Figure 1.5 shows one example: three *var* genes from 3D7's 60-member antigenic repertoire in a parent-parent-child trio of *P. falciparum*

samples: PG0051-C (the 3D7 parent), PG0052-C (the HB3 parent), and PG0063-C (the progeny of these two parents). Kmers unique to each of the three *var* genes are color-coded for clarity (kmers unique to *var* gene *PF3D7_0100100* in red, *PF3D7_0223500* in orange, and *PF3D7_0712600* in green). Non-unique kmers are colored white, indicating that they do not uniquely tag the gene in question. The three genes are located on different chromosomes (1, 2, and 7, respectively). All three *var* genes from 3D7 are apparently absent in the HB3 repertoire, as none of the constituent kmers are present in the HB3 sample. These kmers are colored grey to indicate that they were not recovered in this sample. Examining the 3D7xHB3 progeny, we see that the kmer recovery pattern exhibited in *PF3D7_0712600* matches that of the 3D7 parent, and thus it has likely inherited this 3D7 *var* gene in full. Curiously, the sample exhibits piecemeal recovery of kmers from *PF3D7_0100100* and *PF3D7_0223500*. This is a previously reported NAHR event between the telomeres of chromosomes 1 and 2, likely to have occurred during mitosis, preserving the domain architecture and yielding a functional product.[10]

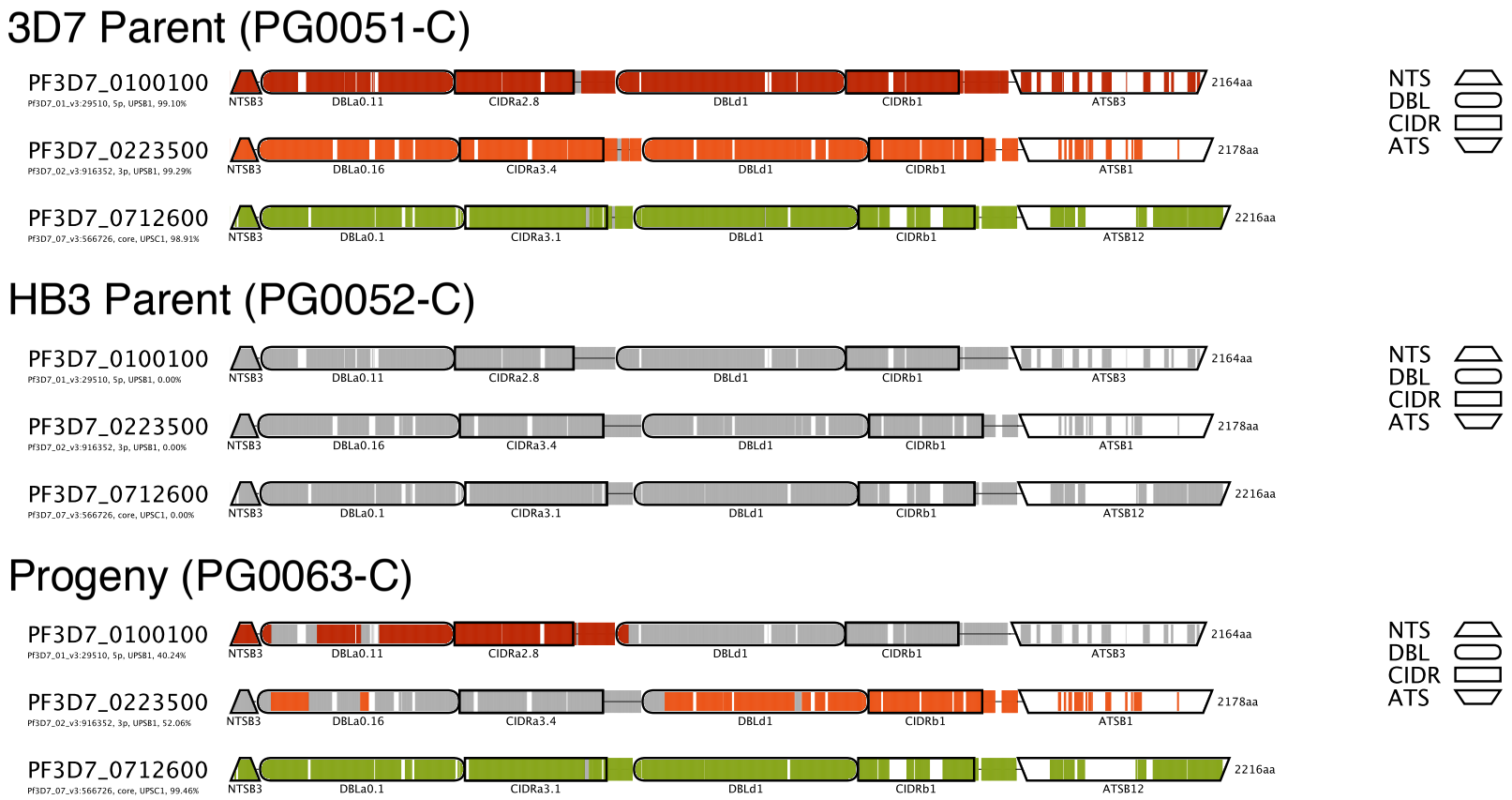


Figure 1.5: Presence and absence of unique kmers in three 3D7 *var* genes, laid out along the length of each gene. Each row represents the coding sequence (CDS) of three *var* genes in the 3D7 repertoire. Each vertical line represents a kmer from that gene. Colored kmers represent those unique 3D7 kmers that are recovered in the sample. Grey indicates no recovery - the kmer was absent from the sample. White indicates the kmer at that position was not unique in the 3D7 genome. Protein domain annotations from the VarDom server are overlaid onto the CDS and represented in different shapes.[1] Additional information about each gene (location in the genome, 5′ telomere/3′ telomere/core placement in the genome, upstream promoter class, and percentage of the gene's unique kmers recovered) are provided below the gene name.

Alignment of short reads to a single reference and subsequent application of variant callers is a poor strategy for *de novo* variant discovery in *P. falciparum* (and likely higher-order species as well) for a number of reasons:

1. **Absent or divergent loci in genome**
   The underlying assumption that two genomes from the same species should be very similar is inappropriate for highly diverse populations (e.g. pathogens) or hypervariable regions (e.g. immune loci in mammals). If a haplotype present in the sample is too dissimilar to the reference, or perhaps even completely absent, read aligners may return incorrect results. Reads may align to the wrong location, the resultant mismatches mistaken for real mutations. Alternatively, they may fail to align at all, thus obscuring evidence of real variation.

2. **Incomplete or errorful reference sequences**
   Reference sequences are often incomplete and/or contain errors due to technical artifacts. Chaisson *et al.* provide an excellent review on the various errors that may arise.[38] Regions of the genome may fail to amplify during library preparation, leading to coverage dropout and subsequent gaps in the assembly. Insufficiently long reads used in the reference genome construction may lead to the misestimation of lengths of repetitive regions, causing repeats to have a collapsed representation relative to the true genome. Segmental duplications, gene families, or other loci with high sequence identity may generate ambiguous read overlaps that cannot be resolved without very long reads.

3. **Difficult to include prior information about variation in a species**
   Read aligners must make a decision as to how many apparent mismatches to permit with respect to the reference sequence. However, these software packages typically operate per-read, without information on prior population variation throughout the genome.

4. **Inability to include improved or project-specific data**
   Improvements in sequencing technology, or new platforms altogether, can yield supplementary datasets that add missing information to a genome or repair a misrepresented locus. There is no natural framework for incorporating these additional datasets to the alignment framework.

a. Genome to be sequenced

```
ACCATAGCTTTAATTA
```

b. Sequenced reads and resulting kmers (k=3)

```
ACCATAG (ACC, CCA, CAT, ATA, TAG)
TAGCTTT (TAG, AGC, GCT, CTT, TTT)
TTAATTA (TTA, TAA, AAT, ATT, TTA)
ATAGCTT (ATA, TAG, AGC, GCT, CTT)
CATAGCT (CAT, ATA, TAG, AGC, GCT)
AGCGTTA (AGC, GCG, CGT, GTT, TTA)
GCTTTAA (GCT, CTT, TTT, TTA, TAA)
```

c. de Bruijn graph reconstruction (k=3)

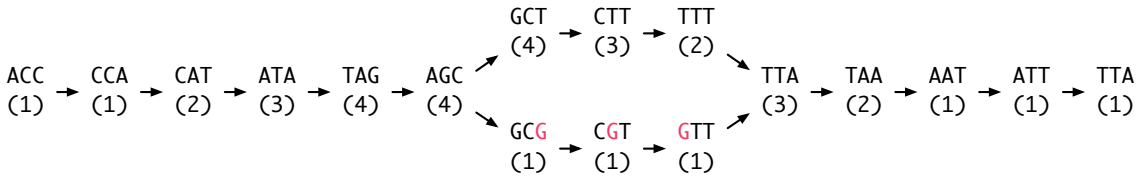


Figure 1.6: The process of generating a de Bruijn graph representation of sequence data. a. The underlying genome. b. Reads sequenced from the genome (including one read with a sequencing error). Reverse-complement reads are not shown for clarity. c. The $k = 3$ de Bruijn graph reconstruction, including kmer coverage annotations.

1.4.2 De novo assembly as an alternative approach

Consider Table 1.2 again, which demonstrates that we can expect to recover nearly all kmers present in the genome with as little as 10-fold coverage. For small genomes (*P. falciparum* is approximately 23 megabases in length), modern sequencing experiments can routinely return excess of 100-fold coverage. It is therefore clear that deeply-sequenced samples will have reads representing the entirety of the genome despite our inability to align all of them to the genome. Rather than relying on mapping to an imperfect and incomplete reference, we can attempt to assemble the genome *de novo* - from the sequenced data itself, ignoring the availability of a reference sequence.

The problem of performing *de novo* assembly essentially reduces to computing read-to-read alignments, rather than read-to-reference alignments. As each read represents recovery of some small region of the genome, the supersequence of overlapping reads (the aligned nucleotides flanked by the non-overlapping sequences from each read) represent some larger linear stretch of the genome. Brute-force computation of all possible pairwise alignments is $\mathcal{O}(N^2)$ in the number of reads, which is impractical for second-generation sequencing datasets with tens or hundreds of millions of reads. Fortunately, there are many ways to compute and represent these overlaps efficiently. We shall focus on one

$\mathcal{O}(N)$ method in this work: assembly via construction of a de Bruijn graph.

Formal definitions can be found in Chapter 4. Informally, a graph is simply a data structure representing a collection of objects (termed "vertices" or "nodes") and connections ("edges" or "links"). Applied to sequencing, a graph encodes linear stretches of sequence, while each edge represents an overlap with the connected vertices. Commonly, each read is decomposed into fixed-length words of an arbitrary length k, or "kmers". As each kmer is sequentially extracted from a read and added to the graph as vertices, edges between adjacent kmers in the read are stored as well. Overlapping reads will share kmers, and since each kmer can only appear once in the graph, adding kmers and edges from the overlapping read effectively records the alignment without needing to literally compute all possibilities. This formulation, wherein fixed-length subsequences and their adjacencies are recorded, is termed a "de Bruijn" graph (as distinct from other formulations, e.g. a string graph, wherein vertices represent variable-length sequences, and edges represent overlaps computed via alignment). Figure 1.6 depicts a simple 16 bp genome, sequenced with 7 bp reads, and the resulting de Bruijn graph constructed at a kmer size of 3.

Construction of this data structure is challenging. First, errors in second-generation sequencing data are very common, and therefore the graph produced from raw sequencing data will contain many branches that are not in the actual genome (examine Figure 1.6 again, observing the highlighted base - a sequencing error - and the resultant perturbation to the otherwise linear graph). These can be mitigated (but perhaps not completely solved) by error-cleaning algorithms that remove low coverage kmers, as presumably in high coverage data, random errors are rare and can be detected and discarded. Second, long repetitive stretches of the genome that feature the same kmers multiple times will be collapsed into a single copy, as de Bruijn graphs only store each kmer once. Finally, homology in the genome can cause two separate regions of the genome to appear adjacent to one another in the graph. This may result in a vertices with multiple outgoing edges, causing unresolvable ambiguity when traversing the graph.

Nevertheless, such an approach should resolve many of the deficiencies of an alignment-based approach. Absent or divergent loci should be recovered. The uncleaned graph should be complete (barring any regions of the genome that suffer from an amplification bias that prevents them from being sequenced). Prior information about variation in the species (or in this case, just the parents) can be included by assembling the parents and comparing to them directly, rather than indirectly via the reference. Finally, additional information can be added at graph construction time.

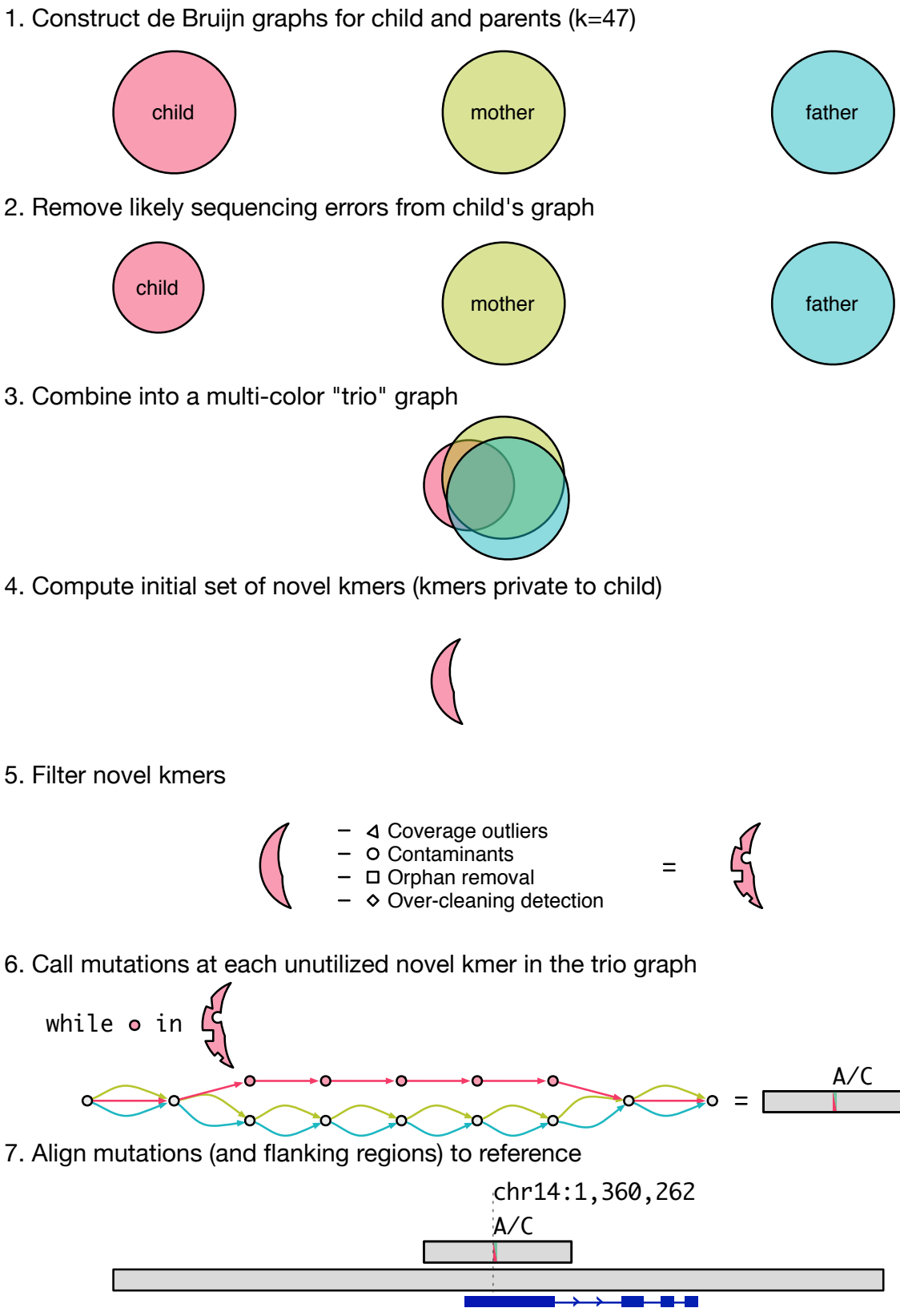


Figure 1.7: Pipeline for making graph-based *de novo* mutation calls in a trio.

1.5 Overview of this work

In this dissertation, we present a novel multi-color graph-based approach to *de novo* mutation detection and allele identification. Figure 1.7 shows the various steps in our pipeline, each of which is discussed in depth in this work. We show how knowledge of the pedigree enables us to identify novel kmers - kmers present in children and absent from parents (step 1-4) - that serve as an exceedingly strong signal as to the presence of *de novo* variation. We demonstrate that strong filtering is required to reduce the number of sequencing and contamination artifacts that arise in the initial novel kmer set (step 5). We use these filtered novel kmers to analyze subgraphs in the genome likely to represent DNMs and navigate color-specific paths and trails to determine the precise allele (step 6). We also demonstrate how the novel kmers act as "sign posts" during graph traversal, indicating that a traversal is following a fruitful path. This simultaneously constrains the runtime of the algorithm and allows us to bypass sequencing error that could not otherwise be overcome. We show that, when given a reference sequence, regions in the graph flanking the variant can be extracted and aligned in order to supply the user with canonical coordinates useful for downstream analysis (step 7). We demonstrate that in high-coverage, haploid datasets, this approaches yields superior sensitivity and specificity to DNMs than reference-based methods, and can access events that occur on the haplotypic background of the non-reference parent. Our sensitivity in low-coverage datasets is substantially lower, and we discuss possible algorithmic changes that could overcome an assembler's aggressive error cleaning and other related issues.

In Chapter 2, we present an overview of the *P. falciparum* lifecycle, review mutational mechanisms that generate *de novo* mutations, discuss their rates, factors that influence their generation, and known events in various species.

Chapters 3 and 4 detail the software packages we have written for this work, the former including descriptions of the realistic variant read simulations, the latter detailing the graph genotyping algorithm.

Chapter 5 presents long-read data for a *P. falciparum* isolate which, when assembled, provides validation data for DNMs in a single sample. It also reveals the need for modifications to our novel kmer filtering strategy when presented with real (rather than simulated) data.

Chapters 6 and 7 present results from applying the algorithm to real datasets. The former chapter focuses on 152 samples from four experimental *P. falciparum* crosses, providing insight into point mutation, indel, and NAHR events. The latter addresses 3 samples from a 9-member *Pan troglodytes verus* (western chimpanzee) pedigree. The chimpanzee

genome is two orders of magnitude larger than the malaria parasite. The dataset is substantially lower coverage than the *P. falciparum* data and lacks a high-quality reference genome. Both are challenging datasets to process for different reasons.

Chapter 8 concludes the work by discussing it in a larger context and details various improvements that can be made in future development and analyses.

2 Background

2.1 Introduction

Many mechanisms contribute to the generation of *de novo* mutations in a genome. In *Plasmodium falciparum*, these mutations can occur at various stages in the parasite's lifecycle. We begin this chapter with a brief review of the parasite's lifecycle so that we might better understand when various mutational mechanisms apply. We follow this with an overview of the mutational mechanisms themselves, and how their products may manifest in genomic sequence data.

2.2 Lifecycle of the P. falciparum parasite

Plasmodium falciparum is a protozoan pathogen with a complex life cycle spanning several stages in (limited) vertebrate and mosquito hosts. We shall review these stages and their constituent components in turn.

2.2.1 Liver stage

Vertebrate infection by *P. falciparum* begins when a pregnant, infected *Anopheles* spp. mosquito bites a host to consume a blood meal.[39] Dozens to hundreds of **sporozoites** (the initial motile invasion stage of the *falciparum* parasite) are transmitted to the host bloodstream via the saliva of the mosquito.[40] The sporozoites travel the bloodstream until they reach the liver, infecting hepatocytes, specifically **Kupffer cells**: macrophages which express low levels of MHC I molecules.[41]

Inside the **parasitophorous vacuole** (a cytoplasm-filled protective envelope, "PV") produced by the sporozoite upon invasion of the hepatocyte, the parasite is protected from the phagolysosomes of the Kupffer cell and can begin asexual reproduction.[42] The sporozoites undergo schizogonic development wherein the sporozoite produces many copies of its nucleus in preparation for multiple fission. Upon cell segmentation, the parasite cells differentiate into **merozoites**, the next invasion stage of the *falciparum* parasite.

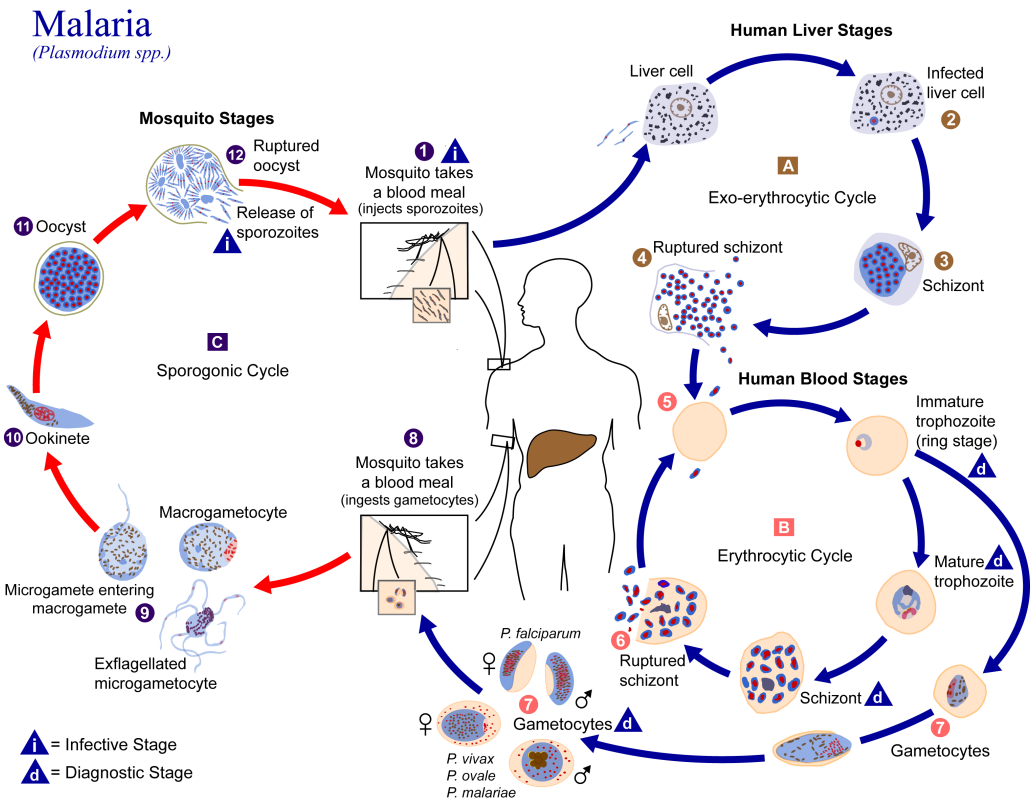


Figure 2.1: Lifecycle of parasites from the *Plasmodium* genus. Reprinted from the CDC's public domain Public Health Image Library (PHIL), image #3405.

The liver stage takes between 6 and 8 days[43] to complete, producing thousands of haploid merozoites per parasetimized hepatocyte.[44] This stage ends when the merozoites exit the hepatocytes and return to the bloodstream via physical rupture of the host cell.

2.2.2 *Blood stage*

Newly released merozoites are non-motile; they come into contact with erythrocytes during transit in the bloodstream. As erythrocytes lack MHC I and II molecules, they make for an effective hiding place from the adaptive immune system.[45] Merozoites invade the RBCs quickly, within 60 to 120 seconds of their egress from hepatocytes[46,47] (presumably this speed is to minimize its exposure to immunological attack). A merozoite once again forms a PV after invasion in order to create a more hospitable development environment for itself. This is followed by a repeating cycle of asexual reproduction. The intra-erythrocytic development cycle progresses through three stages: **ring** (immature trophozoite, named for its morphology under microscopy), **trophozoite** (the merozoite after shedding its surface coat and invasion organelles in its apical complex), and **schizont** (a multi-nucleated cell resulting from multiple fission and just prior to cell seg-

mentation).[48] Eventually the schizont ruptures, releasing 16 to 32 merozoites into the bloodstream which can then quickly infect more erythrocytes. This process takes 24 to 48 hours.

To avoid destruction in the spleen, the parasites express and transport to the surface of the parasetimized RBC (pRBC) a cytoadherence factor termed **PfEMP1** (*Plasmodium falciparum* erythrocyte membrane protein 1).[49] The pRBCs typically adhere to the endothelial cells of post-capillary vessels, thus withdrawing from peripheral circulation.[50] Such parasites are said to be **sequestered** from the immune system, which affords the parasite time to replicate within the parasetimized cell. Should the cell still come into contact with B-cells, each parasite's genome encodes approximately 60 variants of this protein in a family of genes termed *var*. Little to no overlap exists in the repertoires of geographically distant parasites.[51] Expression of a single member of the *var* gene famly is mutually exclusive with all other forms being silenced. This differential expression is epigenetically controlled through the placement of methyllation marks on upstream promoter elements.[52]

All clinical symptoms manifest during red blood cell rupture. Waste products and toxic factors that accumulate in the cell are released into the blood stream when the cell is lysed. This stimulates immune responses. In uncomplicated malaria, symptoms include fever, chills, sweats, rigors, and other flu-like symptoms. As many iterations of erythrocyte invasion, parasite replication, and erythrocyte rupture will occur, these symptoms can appear to periodically relapse. Other symptoms such as anaemia and hypovolemia may occur due to the destruction of RBCs. Severe malaria may occur if the parasite sequestration occurs in the brain (**cerebral malaria, CM**), or in the placenta (**pregnancy associated malaria, PAM**), potentially compromising blood flow to critical tissues. These severe pathologies are often fatal.[53]

2.2.3 *Reproduction*

During the intra-erythrocytic development stage, approximately 10% of parasites are randomly committed to developing into the sexual form: the **gametocyte**.[54] This is currently thought to occur in the erythrocytic schizonts. Committed parasites reinvade new erythrocytes and mature into gametocytes in 10 to 12 days. Mature gametocytes do not exit pRBCs within the human host. Instead, they lie in wait for another mosquito to bite, drawing the gametocyte-carrying erythrocytes as part of the blood meal. Gametocytes mature into male and female gametes upon sensing the different environmental conditions in the mosquito midgut (change in temperature, change in pH, and exposure to

xanthurenate). Mature haploid gametes fuse to form a diploid zygote, which soon mature into an **ookinete**, a motile form of the zygote that can penetrate the lumen of the mosquito stomach and in which meiotic recombination occurs. The ookinete migrates to the mosquito mid-gut epithelial cell wall and forms an **oocyst**, a non-motile thick-walled spore on the exterior surface of the midgut. The newly formed sporozoite replicates via mitosis within the oocyst. The eventual oocyst rupture releases haploid sporozoites that migrate to the salivary gland of the mosquito. Once the mosquito bites another host and transmits the new sporozoites, the cycle repeats.

2.2.4 The P. falciparum crosses

Observing *de novo* mutations in *P. falciparum* parasites requires a system wherein we can obtain DNA for the new sporozoites and their progenitors. Currently, no method for performing such crosses *in vitro* exists, and using a human host for the vertebrate component of the parasite's life cycle would be ethically infeasible. Rodent models are also inviable; *Plasmodium falciparum* is very particular about its vertebrate hosts. Only humans and our closest extant relatives, chimpanzees (*Pan troglodytes*) are known to become infected. In the latter case, it appears infection by *P. falciparum* is asymptomatic, even in splenectomized chimpanzees with high levels of parasitemia.[55]

All four crosses (3D7xHB3, HB3xDD2, 7G8xGB4, and 803xGB4) have involved infecting chimpanzees with sporozoites and later collection of pRBCs for later culturing and analysis.[56] Briefly, this protocol proceeds as follows. First, gametocytes (3D7, HB3, DD2, 7G8, GB4, and 803) are cultured *in vitro*. Next, mosquitos are fed parasetimized blood through an artifical blood feeder;[57] hopefully the mosquitos ingest gametocytes from both parents. These mosquitoes are then permitted to bite chimpanzees, passing the recombinant parasites (and non-recombinant forms as well, given that there is no mechanism to prevent self-crossings) to the animals' bloodstream. Once blood-stage infection is detected, the parasites are harvested, cultured *in vitro* (limiting dilutions ensure that each culture reflects a clonal expansion of a single sporozoite), and screened for recombinant progeny. This process is repeated until enough recombinant progeny have been detected and successfully cultured.

Until single cell sequencing methods can recapitulate a cell's genome with extremely high fidelity, the culturing step is necessary to ensure adequate material for sequencing. Most second-generation sequencing protocols require at least 1 nanogram of DNA to proceed (though most recommendations call for much more), which is still tens of thousands of parasites required at minimum to ensure sufficient genomic yield during library construction. PCR amplification is not recommended; high fidelty enzymes exhibit an error

rate of 10^{-6} mutations/bp, which would lead to 230 PCR-induced mutations in the 23 megabase *P. falciparum* genome. Taken together, the need for culturing is clear.

However, culturing is a third environment in which the parasite must survive. Culturing of *Plasmodium* spp. parasites is notoriously difficult and laborious,[58] and most parasites do not thrive in culture conditions. Those that do almost certainly incur *de novo* mutations that underlie culture adapation. For example, the F12 line of parasites (derived from 3D7) is no longer capable of producing gametocytes.[59–61] This may be a response to existing exclusively in culture where the gametocytogenesis pathway cannot be completed. Commitment to sexual differentiation would preclude further asexual replication in the committed parasites. No selective pressure remains to ensure the genes involved in gametocytogenesis remain functional. Parasites that do not produce gametocytes will quickly out-compete those that do. Their new alleles will quickly reach fixation in the culture population, and many more might reach a nominal allele frequency in the pool.

The implications for what we can expect to see in second- and third-generation sequencing data are clear: we must expect mutations to appear with different allele frequencies, despite occurring in a seemingly haploid genome. Any mutation sustained prior to mitosis of newly recombinant sporozoites in the ookinete will appear to be fixed in the sequenced sample. As cultures are initiated with a single parasite (one hopes), any mutation sustained between recombination and harvesting from chimpanzee peripheral blood will also appear fixed. Finally, mutations occurring *in vitro* may or may not achieve fixation. Barring reverse mutations, only *de novo* mutations occurring during culture can ever be non-fixed in this experimental system. Special care must be taken to not permit these mutations to confound rate calculations for mutational events in the genome, though future analyses regarding culture adaptation may do well to study them more carefully.

2.3 Mutational types

We have discussed the parasite life cycle which gives us a sense of when mutations may occur. We now discuss the types of mutations we should expect to find, and the mechanisms that generate them.

2.3.1 Point mutations

Point mutations (single nucleotide variants, or SNVs) are the simplest (and likely most common) alteration that can occur in the genome. These may be induced by exposure to mutagenic agents (ultraviolet radiation, oxidative damage, base analogs, alkylating agents, etc.). More commonly, point mutations arise as the result of errors during DNA

replication. **Mispairings** between base pairs may occur when a nucleotide, capable of existing in various **tautomeric** forms (repositionings of a hydrogen atom which alters the hydrogen bonding pattern of the nuclotide) cause the altered form to bind to incorrect nucleotides. Such a tautomeric shift may occur prior to or during replication without being corrected. As the polymerase generates the compliment for each strand, the mispaired base opposite the tautometric base will have its true compliment incorporated, thus making the error permanent for all descendent cells.[62]

Similar replication errors may arise after **spontaneous base degradation**. Deamination of cytosine (C) to uracil (U) - a non-standard nucleotide in the genome - can result in the mispairing between the latter and adenine (A). Deamination of methylcytosine can produce a thymine. Should these degraded nucleotides escape repair processes (and in the latter case, there is no error recognition mechanism to detect and remove the aberrant thymine, as it is a normal base in DNA), the changes will effectively be ratified by replication.

2.3.2 *Insertions, deletions, inversions*

The most common cause of **insertion** (the addition of one or more nucleotides to a locus in the genome) and **deletion** (the removal of one or more nucleotides) mutations is errors during the replication of repetitive elements. This can occur when a repeat unit (or several) are looped out during replication, resulting in a "slip" in the template strand versus the nascent strand (an apparent insertion) or vice versa (an apparent deletion).[63]

Indels also commonly arise as the result of DNA repair efforts on **double-strand breaks (DSBs)** by the cell, via a number of different repair pathways. **Homology-directed repair (HDR)** proceeds by first resecting a short region around the DSB in the $5'$ to $3'$ directions on both strands. A genome-wide homology search returns a (presumably) homologous template sequence with which to guide the repair. **Strand invasion** brings the broken strands and complementary complete strands into proximity so that the missing sequence can be resynthesized, forming a DNA **heteroduplex**. A **displacement loop (D-loop)** is formed between the invading $3'$ overhanging region and the homologous counterpart. The action of the polymerase extends the invading $3'$ strand by synthesizing the missing sequence according to the template, and in doing so, the D-loop is changed to a cross-shaped structure known as a **Holliday junction**. The crossing strands are cut, yielding the repaired chromosomes. Mutations can arise if the homology search returns sequence that is similiar but not perfectly homologous to the lost sequence. While in a diploid genome, the homology search would yield sequence on the sister chromatid, such a search outside of the cell's synthesis phase necessarily yields a different part of a

haploid genome. Many types of mutation can be induced by this mechanism, including insertions, deletions, even inversions.

In contrast, **non-homologous end joining (NHEJ)** does not require a homologous template. In most cases, a clean DSB (one that simply separates the strands at the same position in both strands) is easily repaired by the NHEJ pathway by simple ligation of the loose ends. However, should the break be messier, a short region (1 to 4 bp) around the DSB may first be resected in the 5′ to 3′ directions on both strands. Once the ends are blunted and therefore compatible for ligation, the NHEJ pathway fuses the two loose ends, resulting in a deletion of the resected sequence.[64] In rare cases, the repair process can incorporate **non-homologous exogenous DNA** (free DNA that diffuses into the active site and gets caught in the repair process).[65]

The **microhomology-mediated end joining (MMEJ)** pathway also performs a lossy repair on double-strand breaks.[66] Both strands are extensively resected until 5 to 25 bp homologies are located in each strand. These homologous regions are annealed, leading to a deletion of everything between the DSB and the located homologies. These deletions are substantially larger than those produced by NHEJ, typically from 30 to 200 bp in length.[67]

Single-strand annealing (SSA) can be employed to repair DSBs occurring between two repetitive sequences on the same strand with identical orientation.[68] Single strand regions are created adjacent to the breaks via resection, extending to the end of the complimentary repeat sequence until the strands can be brought together by complementarity. The 3′ overhangs are then digested as necessary, and gaps are filled in the sequence in order to restore linearity. The original sequence between the repeat copies is destroyed.

While in many genomes, **transposable elements** (DNA sequences that can move from one region of the genome to another) can contribute to indels, in the finished assembly of the *Plasmodium falciparum* 3D7 reference genome, no transposable elements were found.[34,69,70]

2.3.3 Cross-overs

Cross-overs can take place via **homologous recombination (HR)** during meiosis in the ookinete. The process follows the steps outlined above for the HDR (although in the case of HR, the DSB is not the result of DNA damage, but rather is developmentally programmed to initiate the crossover). The homology search now returns the homologous chromosome from the counterpart gamete. Strand invasion and Holliday junction generation proceed as usual so that the counterpart strand may be synthesized. In contrast to HDR, the second 3′ overhang (which was not part of the initial strand invasion) forms

a second Holliday junction with the homologous chromosome. Rather than cutting the Holliday junction on the crossing strands, two sets of cuts are made (one cut per junction): first on the crossing strand, the second on the non-crossing strands. This results in the crossover product.[71]

While crossovers can produce new chromosomes not present in the gametes, the sequence immediately surrounding the breakpoint may not look altered. Homologous recombination necessarily occurs in regions of homology. In rare cases, the breakpoint may occur between two variants positions between the gametes and produce a crossover product that has the maternal allele on one end of the breakpoint and the paternal allele on the other. Assembly-based approaches to variation discovery may miss this event unless the two alleles now appearing on the same haplotype also appear within a kmer length of one another.

2.3.4 Gene conversion

Gene conversions occur in the same manner as cross-overs, with one exception: should a sequence mismatch be present between the donor and acceptor strands in the heteroduplex, the mismatch must be repaired before the Holliday junctions can be resolved. The mismatch repair process usually acts on the broken strand, using the intact strand as a template, thus effectively converting the sequnce of the broken strand to that of the intact strand in order to restore perfect complementarity. The Holliday junctions are then resolved as they are for cross-overs, yielding a short stretch of sequence that appears to have been donated by one parent amidst the haplotypic background of another.[72]

Once again, these events are difficult to resolve in an assembly framework unless the mismatches are sufficiently close to one another.

2.3.5 Non-allelic homologous recombination (NAHR)

Occassionally, the genome-wide homology search during mitosis or meiosis may yield a template that is not an allele of the sequence being repaired. In this scenario, the recombination machinery described above for cross-over and gene conversion events may yield a non-allelic repair. This joins two disparate regions of the genome, often regions from non-homologous chromosomes, to produce a sequence absent from either parent. Such **non-allelic homologous recombination (NAHR)** events are typically mediated by homologous repetitive elements: **segmental duplications, (SDs)**, typically 10 to 300 kb in length with 95% to 97% sequence identity among them.[73–76] The SDs from different regions of the genome are brought together in the cross-over pathway to generate the new sequence. As it is unlikely that these regions are perfect homologs, the additional action

of mismatch repair may generate some additional sequence modifications before the final sequence is synthesized.

Identification of NAHR events in assembly data is non-trivial. In a perfect assembly, one should be able to observe repeated template switching of the child's haplotype to the maternal and paternal haplotypes, and additionally detect that the templates do not belong to the same locus. However, as NAHR events are mediated by homology, the assembly must be able to overcome the resultant loops in the graph without ambiguity in order to complete the traversal. This is unlikely to be achieved every time (if ever). Thus, in second-generation sequencing data, one expects an NAHR event to manifest as multiple pieces.

The *P. falciparum* genome is 80% AT, highly repetitive, and contains a number of highly conserved SDs in subtelomeric regions of several chromosomes.[77] Notably, the subtelomeric regions of nearly all of *P. falciparum's* 14 chromosomes (excluding the mitochondria and apicoplast) house several members of the antigenic gene families, *var*, *rifin*, and *stevor*. As mitotic diversification of the *var* genes has been observed many times, NAHR is thought to be a critical capability of a pathogen that must continuously evade immune pressure to survive.

2.4 *Known rates*

Little work has been done on *de novo* mutation rates in *P. falciparum*. Two studies are known to exist. The first uses clone trees, an experimental design where mitotic clones of a founding isolate are cultured individually, of selected parasites.

Bopp *et al.* investigated parasite genome changes over time by generating new cell lines from the 3D7 parasite and DD2 parasites. The authors reported detection of 167 SNVs in 25 clones total (6.68 SNVs per sample on average) and 59 structural variants (insertions and deletions). Bopp *et al.* estimate the SNV rate to be $(1.7 - 3.2) \times 10^{-9}$ mutations per base per generation[1] (absent clones exposed to anti-malarial drugs), and 9.5×10^{-6} structural variations per base per generation.[78] SNVs were relatively confined to the so-called **core** of the genome (excluding subtelomeric, pericentromeric, and interspersed hypervariable regions), while structural variants were found predominantly in subtelomeric regions housing antigenic genes.

Claessens *et al.* find similar numbers in their clone tree work. Having produced and whole-genome sequenced 37, 56, and 81 subclones from the 3D7, DD2, and HB3 (respectively), the authors find SNV mutation rates of 4.07, 3.63, and 3.78×10^{-10} mutations

[1] A generation is defined as the number of intraerythrocytic life cycles, computed by dividing the time in culture by the time per intraerythrocytic life cycle.

per base per generation.[10] While this may seem considerably different than the Bopp *et al.* estimate, the Bopp estimates adjusted their raw data to account for non-synonymous deleterious mutations that may have been eliminated by purifying selection prior to identification. Adjusting the Claessens *et al.* estimates using the Bopp methodology, the rates become 4.28, 3.82, and 3.98×10^{-9}, in line with the Bopp estimates. Claessens *et al.* note that they observed "back-to-back" mutations (what we refer to in this dissertation as *multinucleotide polymorphisms, MNPs*). In their rate calculations, they considered these as two adjacent SNPs, but note that this assumption may be unwarranted.

In examining NAHR, Claessens *et al.* isolated their analysis to exon 1 of *var* genes in 3D7, DD2, and HB3. This is the longer exon of the two that comprise *var* genes, and encodes the region of the PfEMP1 protein that is immunologically exposed. The authors report a ratio of exon 1 recombinations per SNP to be 0.27, 0.35, and 0 for 3D7, DD2, and HB3, respectively. HB3 is notable in that it appears to exhibit no NAHR events whatsoever.

2.5 *Biases*

The location where DNMs may occur is not purely random. Sequence context can contribute to DNM generation. As discussed above, spontaneous deamination of cytosine to uracil can cause mispairings and be improperly corrected by the cell to generate a DNM. Thus, we will expect to see an elevated mutation rate in CpG islands - long runs of CG dinuclotides that provide many opportunities for mutation. Repetitive sequences also provide ample opportunities for mutation, as copies of the repeated sequence may slip and be looped out during replication, producing deletion mutations.

Parental age can also contribute to DNM production. In humans and chimpanzees, it has been shown that the *de novo* mutation rate in children is largely driven by the age of the father at conception.[7,9] From five months prior to birth to birth itself, female primordial germ cells undergo a mere 22 mitotic divisions, and 2 meiotic divisions during sexual maturity[79] to produce mature female gametes (ova). In contrast, male sperm stem cells undergo a division every 16 days in addition to subsequent mitotic divisions of the gonial cells (4) and meiotic divisions of the meiotic cells (2). The sperm cells thus have vastly more opportunities to accumulate errors than ova. A number of disorders (e.g. Apert syndrome, achondroplastia, thanatophoric dysplasia, Costello syndrome, etc.) have been associated to *de novo* mutation.[80–82]

In *P. falciparum*, no information regarding a parental age effect is available. The experiment would be exceedingly difficult to perform in principle as we'd have to ensure that one parasite only contributed male gametocytes and the other only contributed female

gametocytes. In our crosses, the "gender" of the parents may change from one progeny to the next; no data is available for the gender of the gametocytes that have contributed their genetic material to the resulting progeny.

2.6 Summary

De novo mutations may arise from a multitude of sources. Although SNPs are the most common mutational class observed, *de novo* mutations range from simple point mutations, to insertions and deletions of various lengths, even to interchromosomal exchanges. They tend to be rare, occurring just a handful of times within a genome per sample per generation. Our approach will need to be able to distinguish error from true variants, and encompass a wide range of mutational classes.

3 Simulation

3.1 Introduction

De novo MUTATIONS WILL UNDOUBTEDLY TAKE MYRIAD FORMS (SNPs, insertions and deletions of varying length, expansions and contractions of short tandem repeats, tandem duplications, non-allelic homologous recombinations, and possibly even inversions). Detecting all types of variants is a considerable challenge, and the software to do so will be introduced in the next chapter. In order to measure that software's expected sensitivity and specificity to such variation, we require truth datasets to which we can compare our calls. A sufficiently small genome (on the order of tens of megabases), could be run on third-generation sequencers, the long reads used to assemble full-length genomes. Then, the variants called from short-read second-generation sequencing data could be compared to the "truth" dataset established by the newer platform. This is still very expensive (thousands of dollars for each sample), which limits the number of samples that could be feasibly obtained. Furthermore, if variants of the classes we are attempting to identify are absent in the handful of genomes we can afford to sequence, we would not be able to accurately ascertain our power. Finally, it is extraordinarily time-consuming; long-read sequencing experiments require tens of micrograms of genomic, high molecular weight DNA during library construction, which translates to several months of culturing. For *P. falciparum* parasites which are notorious for being difficult to adapt to culture conditions, this limits the number of samples one can reasonably expect to sequence.

We chose to pursue both a simulation strategy and a validation strategy in order to measure our DNM calling performance. The results from validation will be presented in Chapter 5. For the simulation work, we developed two components: first we must generate the genomes of the parents and several children, including each type of DNM we hope to discover. From these genomes, we must then simulate reads that realistically model errors inherent in our data (matching read lengths, fragment size distributions, single base mismatch errors, indel errors, read pair chimeras, coverage profile, etc.). We discuss both of these components below.

3.2 *Simulating genomes*

Simulating a genome merely involves generating an artificial reference sequence in FASTA format. Our framework is a simple forward simulation of samples. We first generate the genomes of the parents. To generate a child's initial genome, we perform recombination *in silico*. We then add *de novo* mutations on this substrate, thus producing the child's final genome.

We make use of the Variant Call Format (VCF),[83] a text file that encodes one variant per line, specifying the genomic locus, reference and alternate alleles, and metadata for the variant, to describe differences between the two parents and the DNMs to incorporate into the child's genome. While the `FastaAlternateReferenceMaker` module in the GATK does purport to generate a new reference sequence based on variants in a VCF, we note that at the time of this writing, it silently fails to incorporate multinucleotide polymorphisms (MNPs) (simultaneous deletion and insertion). We generate many of these events to remove a reference allele and add an alternate allele in its place (e.g. inversions or gene repertoire replacements). To include this critical functionality, we developed our own tool to permute an existing reference sequence based on a single-sample VCF file.

Our algorithm, `IncorporateVariantsIntoGenome`, is described in Algorithm 2. Briefly, the sequence of each chromosome is loaded into an array, one nucleotide per array element. We then iterate over each record in the VCF file. For each SNP or insertion, we replace the reference nucleotide at that position with the entire alternate allele (for insertions, more than a single nucleotide). For deletions, we replace each corresponding array elements with empty strings. To generate the new genome, we iterate through each element of the array, emitting the string found in each position.

Note that we chose not to process each variant iteratively as insertions (deletions) would increase (decrease) the size of the array, altering the mapping between the VCF positions and the array positions. Keeping track of the mapping in spite of the changes is cumbersome. Instead, out scheme of placing all of the variants on the chromosome array first and then emitting the resulting sequence preserves the mapping. We will revisit this strategy later on in this chapter when we introduce an algorithm to lift data over from the reference genome coordinates to a simulated genome's coordinates.

Algorithm 2 could fail to produce a correct FASTA file in the pathological case that there are multiple overlapping variants called at a single locus. For example, if we were to simulate a 50 bp deletion at position 3 in the chromosome and a SNP at position 5, our procedure would wipe out information between positions 3 and 53 before we could modify the data at position 5, owing to the sequential nature of the processing. We are

Algorithm 2 Generate an alternative reference sequence based on a VCF file.

```
1: function IncorporateVariantsIntoGenome(ref, vcf)
2:     for all chr in ref do
3:         vcs ← vcf.getVariants(chr)
4:         for all vc in vcs do
5:             if vc.getType() == DEL || vc.getType() == MNP then
6:                 for pos in vc.getPosition() to (vc.getPosition() +
   vc.getReferenceAllele().length()) do
7:                     chr[pos] = ""
8:             chr[vc.getPosition()] = vc.getAlternateAllele()
9:         write(chr)
```

careful to avoid that scenario; we set our simulated variants to be placed no closer than 1,000 bp from one another. We do not simulate variants larger than this, as we're unlikely to detect such large events in short read sequencing anyway. Overall, this means we do not test ourselves on our ability to discover and properly type variants that appear in rapid succession in the genome.

Many algorithms presented in this chapter rely on empirical distributions to model cross-over rates, read fragment size, indel lengths, and positional errors in reads. In all cases, we use the inverse transform sampling method for pseudo-random number generation from an arbitrary probability distribution given its cumulative distribution function (CDF).[84] Simply put, we compute the CDF for an empirical probability distribution, generate a random uniform deviate between 0 and 1 for the x value, and interpolate the y value from the CDF.

3.2.1 *Parents*

We began by simulating the genomes of two parents using a workflow depicted in Figure 3.1. For convenience, we simulated samples from the 3D7xHB3 cross. Choosing the existing reference sequence for the 3D7 sample obviates the need to generate any data for the first parent. The second parent is tricker; while an existing draft reference sequences does exist for HB3, it is of low quality, assembled into thousands of pieces rather than the simple 14 autosomes we expect (Table 3.1 shows metrics on every *P. falciparum* sample publicly available). Using the supercontigs from draft reference sequences is hugely cumbersome for simulating recombination as it is not straightforward to decide which chromosomes in the 3D7 genome and which supercontigs should be processed together.

Instead, we chose to produce a new HB3 reference genome sequence by taking the 3D7 reference and inserting the appropriate modifications using Algorithm 2. These modifi-

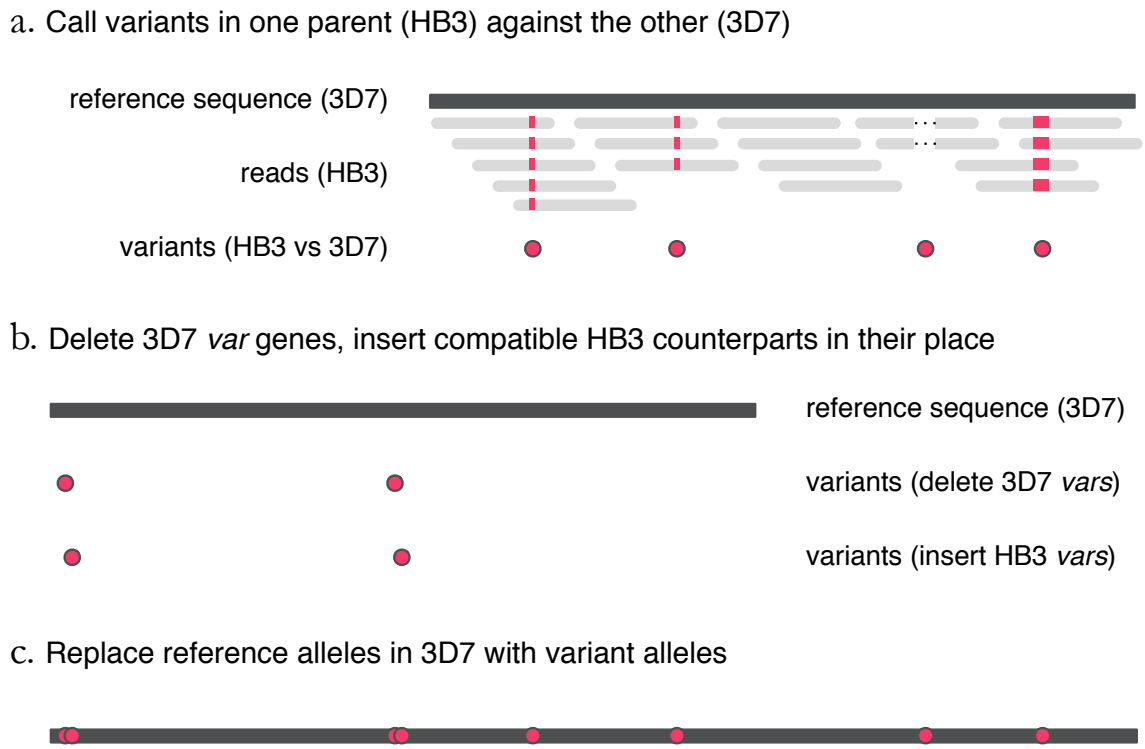


Figure 3.1: Workflow for generating the HB3 parental genome. a. Reads from HB3 sample, PG0052-C, are mapped to the 3D7 reference genome, and variants (SNPs and indels) are called and stored as a VCF file. b. We remove the 3D7 *var* gene repertoire, replacing each with a reasonable HB3 *var* counterpart, and encode the changes to the 3D7 reference genome as a VCF file. c. We alter the reference genome using Algorithm 2, thus producing the simulated HB3 genome.

Table 3.1: Assembly statistics on publicly available finished and draft *P. falciparum* references, ordered by scaffold N50 length. Parents are shown in bold.

Isolate	Origin	Length (Mb)	Scaffolds	Scaffolds N50 (Kb)	%Q40
3D7	**Unknown**	**23.30**	**16**	**1,690.00**	**-**
HB3	**Honduras**	**24.26**	**1,189**	**96.47**	**93.17**
IGH-CR14	India	21.74	849	37.02	95.49
DD2	**Indochina/Laos**	**20.88**	**2,837**	**19.11**	**85.66**
RAJ116	India	14.11	1,199	13.00	89.68
VS/1	Vietnam	18.89	5,856	4.42	74.79
7G8	**Brazil**	**14.28**	**4,843**	**3.87**	**71.00**
Senegal_V34.04	Senegal	13.24	4,329	3.76	76.22
D10	PNG	13.38	4,471	3.71	71.80
RO-33	Ghana	13.71	4,991	3.47	69.91
K1	Thailand	13.29	4,772	3.42	73.30
FCC-2/Hainan	China	12.96	4,956	3.30	69.39
D6	Sierra Leone	13.22	5,011	3.23	71.62
SL	El Salvador	13.19	5,193	3.08	69.41
PFCLIN	Ghana	42.19	18,711	2.99	-

cations are comprised of two parts: introducing the appropriate variants, and replacing the *var* gene repertoire.

We first obtained a VCF of variants found in the HB3 sample, PG0052-C, from the 3D7xHB3 cross dataset.[19] Variant counts are described in the Table 3.2, and include SNPs, insertions, deletions, and complex (simultaneous insertions and deletions) events. These calls were made by combining the results of the reference-based `UnifiedGenotyper` module in the GATK[27] and the reference-free bubble-calling algorithm in the Cortex[33] software. All calls were restricted to the core genome; subtelomeric regions were masked out due to poor mapping properties (owing to the tremendous diversity in these regions of other *P. falciparum* parasites with respect to the 3D7 reference).

Next, we produced a VCF describing *var* gene replacements. The sequences for HB3 *var* genes and upstream promoter metadata were obtained from the VarDom server.[85] No positional information from this data source is available, thus the exact placement of these *var* genes in a chromosomal context is unclear. However, previous work has established a strong association between conserved sequences of upstream promoters (phylogenetically

Table 3.2: Variants found the HB3 (PG0052-C) sample from the 3D7xHB3 dataset.

variants	SNPs	insertions	deletions	complex
42,054	15,376	11,807	14,643	228

grouped into five classes: A through E) and placement in the genome.[86] We therefore replaced 3D7 *var* genes with HB3 *var* gene counterparts, taking care to replace genes with similar UPS classes whenever possible, and grouping genes with suspiciously incomplete metadata otherwise. The replacements were described in the resulting VCF as simultaneous deletions of the 3D7 allele and insertions of the HB3 allele. The precise replacements are summarized in Table 3.3.

These two VCFs were combined to produce a complete set of changes required to transform the 3D7 genome into a pseudo HB3 genome. The transformation was applied using Algorithm 2.

3.3 Children

Generating the genomes of the children is slightly more involved, as there is much more biology to simulate, and many more considerations to be made when placing variants. We generated VCF descriptions of the children using a multi-stage workflow shown in Figure 3.2. In order, we simulate:

1. homologous recombination between 3D7 and HB3 genomes
2. gene conversion events
3. NAHR events between compatible *var* genes
4. *de novo* SNPs, insertions, deletions, and inversions

3.3.1 Homologous recombination

3.3.1.1 Allelic homologous recombination

For each bivalent chromosome, the cross-over rate will be dependent on the length of the chromosome. The empirical distributions for bivalent formation and crossover were generated from the 75 samples in the MalariaGen crosses data. For each sample, only half of the chromosomes are expected to exhibit cross-over events. The cross-over rates are plotted in Figure 3.3. We simulated recombination in a sample by first drawing a binary number indicating whether a chromosome should be recombined, and if so, drawing the number of cross-over events per chromosome from these empirical distributions. The recombination sites themselves were chosen by drawing a uniform random variate between 1 and the length of the chromosome. Although there are certainly hotspots and coldspots of recombination in the genome, we have ignored this complication. An example haplotype mosaic of chromosome 12 for five samples is shown in Figure 3.4.

Table 3.3: *Var* gene replacements from 3D7 to HB3 repertoire.

3D7 gene	3D7 ups class	HB3 gene	HB3 ups class
PF3D7_0421100	UPSB5	PFHG_02500	UNKNOWN
PF3D7_0600200	UPSB2	PFHG_02495	UNKNOWN
PF3D7_0632500	UPSB5	PFHG_05132	UNKNOWN
PF3D7_0800300	UPSB2	PFHG_04012	ND
PF3D7_1200400	UPSB5	PFHG_05200	ND
PF3D7_1240900	U	PFHG_05483	ND
PF3D7_0400400	UPSA1	PFHG_03840	UPSA1
PF3D7_0425800	UPSA1	PFHG_03671	UPSA1
PF3D7_1100200	UPSA1	PFHG_04861	UPSA1
PF3D7_1150400	UPSA1	PFHG_05052	UPSA1
PF3D7_1300300	UPSA1	PFHG_03234	UPSA1
PF3D7_0533100	UPSA2	PFHG_03521	UPSA2*
PF3D7_0100300	UPSA3	PFHG_02274	UPSA3
PF3D7_0100100	UPSB1	PFHG_04081	UPSB1
PF3D7_0115700	UPSB1	PFHG_04749	UPSB1
PF3D7_0200100	UPSB1	PFHG_03516	UPSB1
PF3D7_0223500	UPSB1	PFHG_04277	UPSB1
PF3D7_0300100	UPSB1	PFHG_03717	UPSB1
PF3D7_0324900	UPSB1	PFHG_04035	UPSB1
PF3D7_0400100	UPSB1	PFHG_04491	UPSB1
PF3D7_0426000	UPSB1	PFHG_04620	UPSB1
PF3D7_0500100	UPSB1	PFHG_04593	UPSB1
PF3D7_0632800	UPSB1	PFHG_04770	UPSB1
PF3D7_0700100	UPSB1	PFHG_04057	UPSB1
PF3D7_0712300	UPSB1	PFHG_03232	UPSB1
PF3D7_0733000	UPSB1	PFHG_04928	UPSB1
PF3D7_0800100	UPSB1	PFHG_03416	UPSB1
PF3D7_0413100	UPSB3	PFHG_03476	UPSB3*
PF3D7_0712400	UPSB3	PFHG_02421	UPSB3
PF3D7_1240300	UPSB4	PFHG_02272	UPSB4
PF3D7_0809100	UPSB6	PFHG_02276	UPSB6
PF3D7_0712800	UPSB7	PFHG_04014	UPSB7
PF3D7_0808700	UPSB7	PFHG_02425	UPSB7
PF3D7_1240400	UPSB7	PFHG_04769	UPSB7
PF3D7_0412400	UPSC1	PFHG_03480	UPSC1
PF3D7_0412700	UPSC1	PFHG_03478	UPSC1
PF3D7_0412900	UPSC1	PFHG_00592	UPSC1
PF3D7_0420700	UPSC1	PFHG_02419	UPSC1
PF3D7_0420900	UPSC1	PFHG_02429	UPSC1
PF3D7_0421300	UPSC1	PFHG_02277	UPSC1
PF3D7_0617400	UPSC1	PFHG_02273	UPSC1
PF3D7_0712900	UPSC2	PFHG_04015	UPSC2
PF3D7_1200600	UPSE	PFHG_05046	UPSE

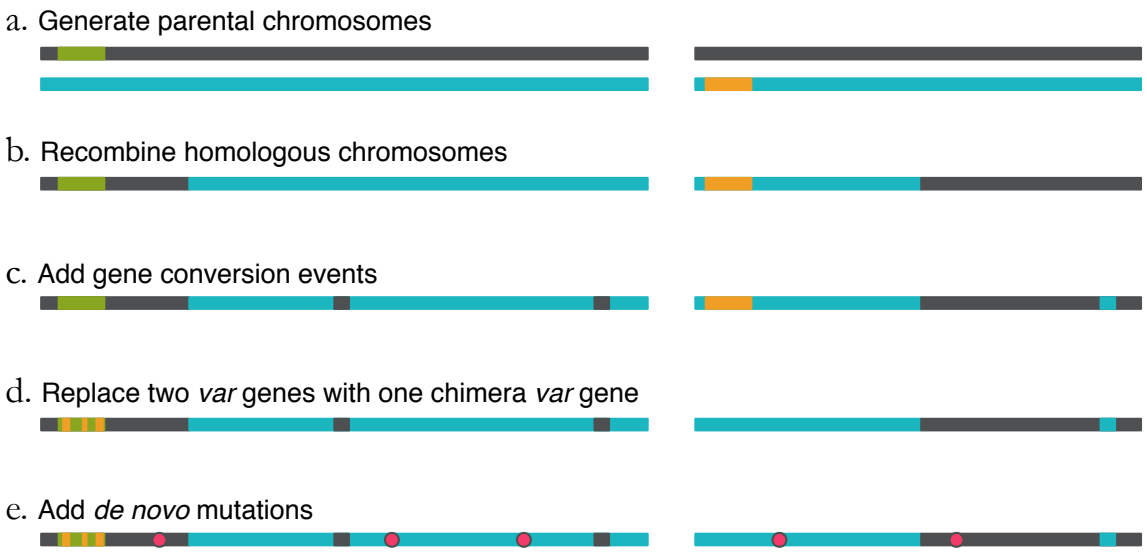


Figure 3.2: Workflow for generating children's genomes. a. Generate chromosomes from the parental genomes (compatible *var* genes shown in green and orange). b. Recombine homologous chromosomes. c. Add gene conversion events (by incorporating variants from the alternative haplotypic background over a limited genomic window). d. Replace one of the *var* genes with a chimera of compatible genes. e. Add *de novo* mutations.

3.3.1.2 Gene conversion

To simulate gene conversion events, we first chose a handful of random sites known to be variant in HB3, and determined the size of the event (number of adjacent HB3 variants involved in the gene conversion) by choosing a random uniform variate between 1 and 3 (these bounds keep gene conversion tracts short, justified by the fact that observed pathological gene-conversion events are typically around 200 bp in mammals, and rarely exceed 1 kbp[72]). If these sites were originally transmitted to the child, they were removed from the VCF. If they had not been transmitted, they were added. The homologous recombinations and gene conversion events are displayed below for each chromosome and sample.

3.3.1.3 Non-allelic homologous recombination

NAHR events were generated by first finding compatible recombination partners. This list was generated by determining which 3D7 *var* genes had been transmitted to the child, grouped by upstream promoter class and telomeric positioning (5′ or 3′). In each group, two random genes were chosen. If necessary, the gene sequences were reverse complemented to have matching orientation (note that the sequences may not have the same orientation in the simulated genomes themselves). As NAHR events between two *var*

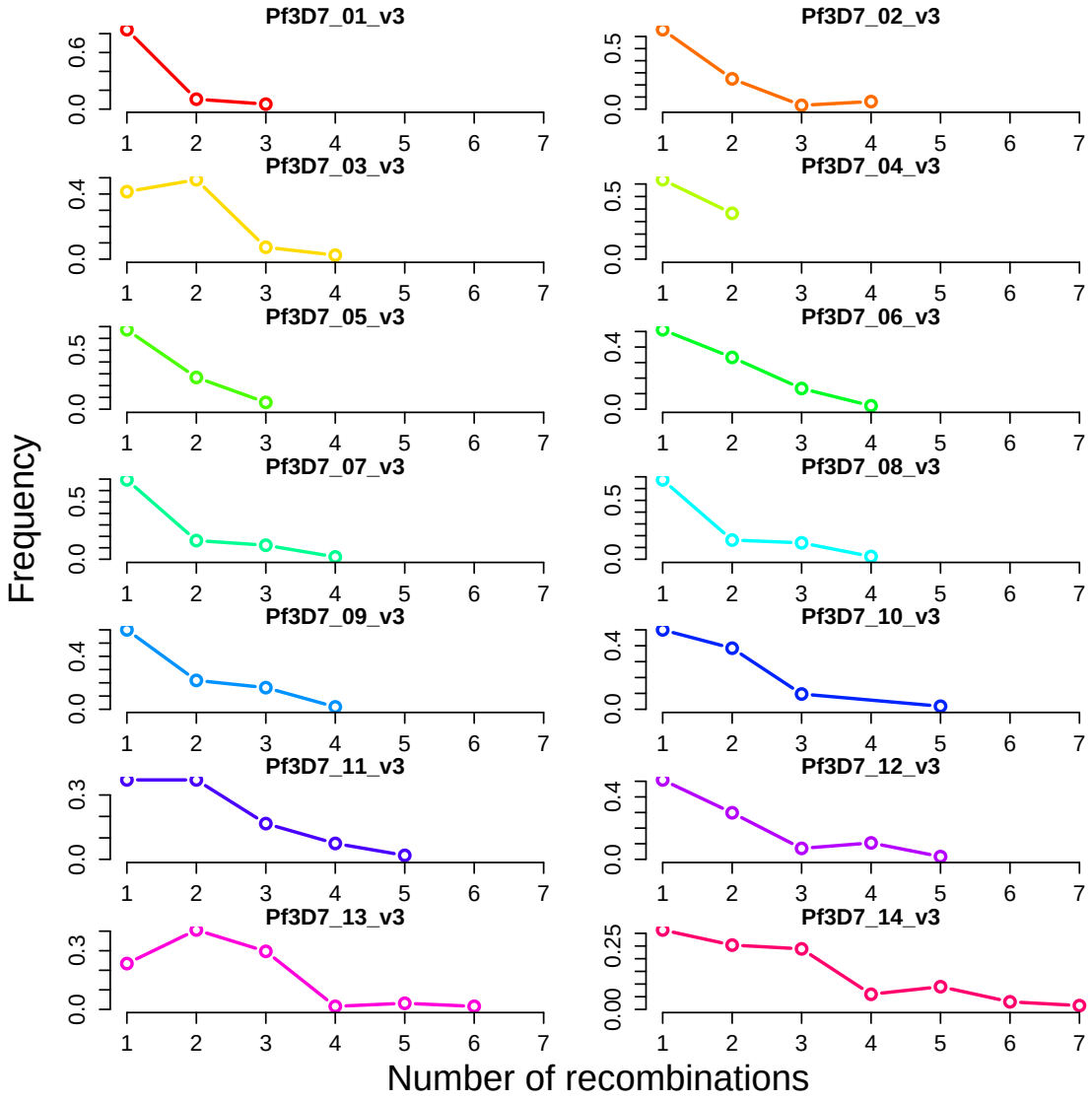


Figure 3.3: Empirical recombination frequencies per chromosome

genes appear to occur in regions of homology, we identified shared 9-mers, positioned between 20 bp and 100 bp between the two genes, to act as possible recombination sites. We randomly selected between 2 and 5 of these positions to act as recombination breakpoints, switching between the sequences and copied sequence data accordingly. Keeping in mind that our previous work has shown that the recombined *var* genes are lost in order to produce the chimera, we added VCF records to delete the previous *var* alleles from the child's genome and replace one of them with the recombined sequence.

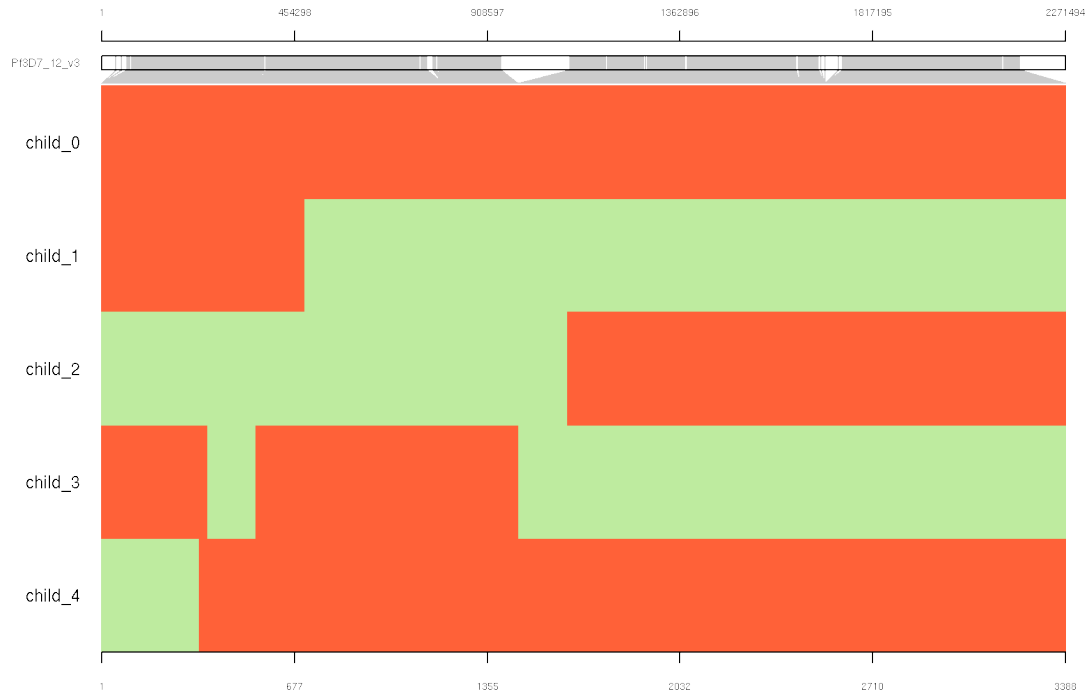


Figure 3.4: Simulated haplotype mosaics for chromosome 12. Genomic position is shown at the top of the figure, while variant number is shown at the bottom. Each variant is depicted as a vertical grey line attached to the mosaic plot at the appropriate location. In the mosaic, every variant is color-coded by parent of origin.

3.3.2 SNPs, insertions, and deletions

With the ground state genome now generated, we further generated simple events - SNPs, insertions, and deletions - on this foundation to complete the production of the child's genome. The precise number of events can be specified by the user at runtime, and for each simulated genome, different counts were specified.

To simulate *de novo* SNPs, we added sites with random (non-reference) alleles at random positions throughout the genome. We also simulated insertion, deletion, and inversion events at every length between 1 and 100 bp. For insertions, random alleles were generated and tested to ensure they did not match the allele already in the reference sequence. For deletions, we simply replaced the reference allele with a truncated allele of the prescribed length. For inversions, we replace the reference allele with its complement.

3.3.2.1 Expansion and contraction of short tandem repeats (STRs)

As a special case of indels, we sought specifically to simulate the expansion and contraction of short tandem repeats (STRs), depicted in Figure 3.6f-g. STRs are constrained to occur at loci where there are already existing repeats, and expansions (contractions)

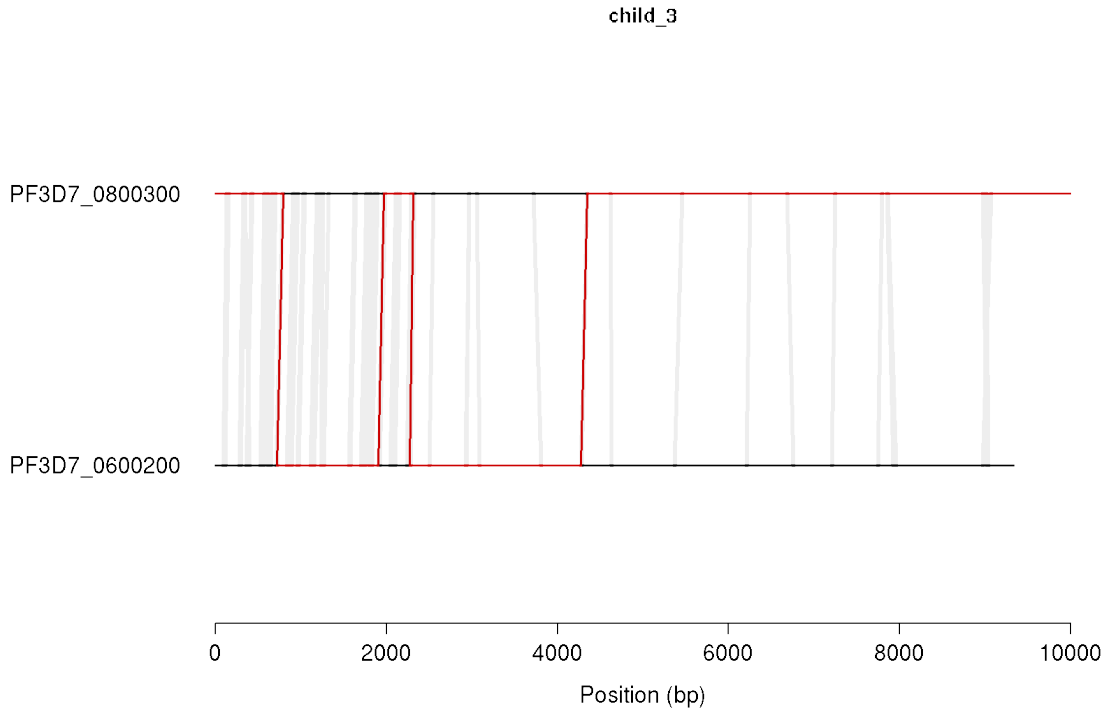


Figure 3.5: Non-allelic recombinations for two compatible *var* genes.

a. parental haplotype ATAAATATTACTCGTCGTCGTTGTGTATACTGCAGT

b. SNP ATAAATATTACTCGTCATCGTTGTGTATACTGCAGT

c. insertion ATAAATATTACTCGTCTCATCGTTGTGTATACTGCAGT

d. deletion ATAAATATTACTCGTCGTTGTGTATACTGCAGT

e. inversion ATAAATATTACTCGTCCGAGTTGTGTATACTGCAGT

f. STR expansion ATAAATATTACTCGTCGTCGTCGTTGTGTATACTGCAGT

g. STR contraction ATAAATATTACTCGTCGTTGTGTATACTGCAGT

h. tandem duplication ATAAATATTACTCGTCGTCGTTGCGTCGTCGTTGTGTATACTGCAGT

Figure 3.6: Simulated variant types. a. Original, parental haplotypic background upon which variants will be placed. b. A single nucleotide polymorphism. c. An insertion of two nucleotides. d. A deletion of three nucleotides. e. An inversion of four nucleotides. f. Expansion of a 3 bp short tandem repeat (STR) by one unit. g. A contraction of an STR by one unit. h. A tandem duplication of 11 nucleotides.

should manifest as the insertion (deletion) of whole units at a time. We first built a map of repeated 2-bp, 3-bp, 4-bp, and 5-bp sequences in the 3D7 genome. We filtered these lists, retaining only STRs where the repeated unit occurred at least three times. For each simulated event, we randomly chose a position from the appropriate list and select a

number of units to add or remove. The number of units is constrained to be less than the length of the number of repeat units of the existing STR. With these considerations, we simulated expansions and contractions at the aforementioned repeat unit sizes.

3.3.2.2 Tandem duplications

For tandem duplications (as depicted in Figure 3.6h) of length l, we chose positions in the genome at random, copied the next l bases, and inserted an identical copy at the same locus. Events at each length between 10 bp and 50 bp were produced.

3.4 Simulating reads

We now turn our attention to simulating second-generation sequencing reads given an underlying genome. There are existing tools that will simulate perfect reads and uniform coverage, which will be important for initial tests of our variant identification software. However, the crucial simulation is of imperfect reads with non-uniform coverage. Without this, any estimate of our sensitivity and specificity is unlikely to be predictive of performance in real data.

There are many tools that can simulate imperfect reads from a given sequence. Almost every solution involves user-specified parameters controlling the properties of the sequencing data. For instance, a tool may model fragment size as a normal distribution, requiring the user to specify the requisite shape parameters. It may also permit the user to specify a desired mismatch rate to simulate the presence of sequencing error. Some tools have presets that automatically set parameters to those consistent with average behavior for various sequencing platforms.

The problem with all of these tools is an over-reliance on assumed parameters of real data. Should a fragment size distribution for real data deviate from the typical normal distribution, this will not be captured in the simulated dataset. Mismatch and indel errors do not happen at any position in the read with equal probability, but rather are biased towards later cycles and certain sequence contexts. Read coverage is not simply Poisson-distributed, but varies across the length of the genome depending on GC bias, secondary structure, even the particular sequencing chemistry used. There are currently no tools that are capable of capturing such nuance.

Instead, we chose to learn empirical distributions of major sequencing properties using an exemplar dataset and sample from those distributions directly in order to generate reads. This frees us from having to make any assumptions about our dataset, easily generalizes to any dataset regardless of when, where, or how it was sequenced. It's also vastly more realistic than other approaches.

Our approach is detailed below. Briefly, it consists of three components:

1. A "coverage profile": a description of where every fragment and read in the genome should fall and which should contain errors.
2. A "read profile": a description of where to place errors within a read, including mismatches, insertions, and deletions.
3. The read simulator: samples from the profiles to generate reads in the new reference sequence.

3.4.1 Constructing the coverage profile

Construction of a coverage profile consists of two sub-problems: providing a complete description of where reads fall on the existing reference sequence, and transferring that information sensibly to the modified reference sequence. This is depicted in Figure 3.7. We address each of these needs with custom tools.

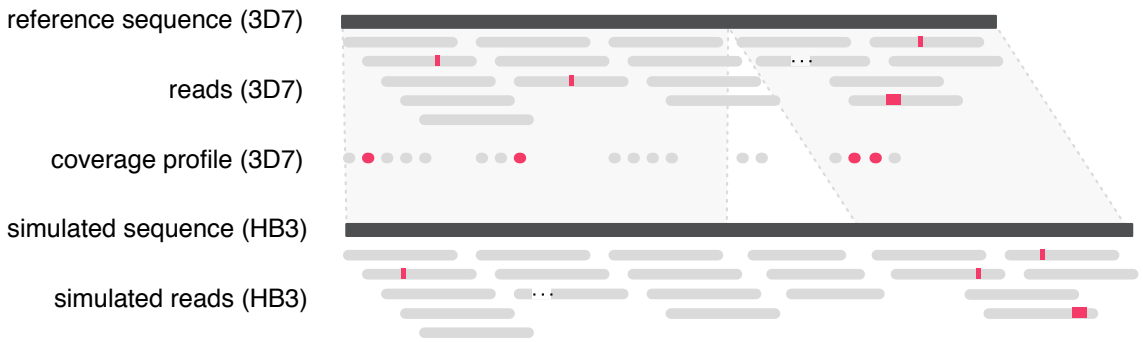


Figure 3.7: Construction of the coverage profile for the reference genome and liftover to the altered genome. Each read start (and fragment start, not shown) is stored along with a count of the number of reads at that location that contain errors. This information is then lifted over to the simulated sequence, and gaps in the table are filled in with neighboring values.

3.4.1.1 Computing read and fragment starts, and error rates

We developed a tool, `ComputeBaseAndFragmentErrorRates`, which provides a precise specification for the number of fragments and reads found starting at of every position in the canonical reference genome, as well as an accounting as to which reads and fragments contained any kind of error (mismatch or indels). Briefly, we iterate over every chromosome in the reference genome, advancing through every aligned read stored in an

exemplar sample's coordinate-sorted BAM file. We instantiate a chromosome-length array indicating the number of reads that start at a given position and the number of reads that contain apparent errors (any discrepancy from the reference sequence).

We must also store information about read fragment starts and error rates, which requires us to keep track of read pairs as we traverse the BAM file. To do so, we hash the read name to the first instance of the read we see along the length of a chromosome. As paired-end reads have the same name, the second time we see the same read name, we will have found the second end of the pair. We then increment the count at the chromosome array position that corresponds to the 5′-most end of the pair, and increment the fragment errors array based on errors in either end of the pair.

This algorithm is described in Algorithm 3.

Algorithm 3 Emit all fragment starts, read starts, and error rates per position.

```
1:  function ComputeBaseAndFragmentErrorRates(BAM)
2:      for all chr in chrs do
3:          nReadsErrors ← []
4:          nReads ← []
5:          nFragmentsErrors ← []
6:          nFragments ← []
7:          seenReads ← []
8:          reads ← BAM.getAllReads(chr)
9:          for all read in reads do
10:             readErrorPositions ← getPositionsErrors(read)
11:             refErrorPositions ← convertReadPositionsToReferencePositions(readErrorPositions)
12:             for all refErrorPosition in refErrorPositions do
13:                 nReadsErrors[refErrorPosition] + +
14:             for readPosition in 0 : read.length() do
15:                 nReads[convertReadPositionsToReferencePositions(readPosition)] + +
16:             if !seenReads.contains(read.getName()) then
17:                 seenReads ← read.getName()
18:             else
19:                 mateErrorPositions ← getPositionsErrors(seenReads[read.getName()])
20:                 if  thenreadErrorPositions.size() + mateErrorPositions.size() ≥ 1
21:                     nFragmentsErrors[seenReads[read.getName()].getAlignmentStart() + +
22:                 nFragments[seenReads[read.getName()] + +
23:         for pos in 0 : nReads.length() do
24:             print chr, pos, nReadsErrors[pos], nReads[pos], nFragmentsErrors[pos], nFragments[pos]
```

3.4.1.2 *Lifting read profile over from reference to simulated genome*

The genomic coordinates present in the table produced by Algorithm 3 must be transformed from the reference sequence to the simulated genome before it can be used. To do

so, we developed a tool that could liftover the coordinates appropriately when given the table, reference sequence, and a VCF file describing the alterations made to the reference to transform it into the simulated genome. This is accomplished with an algorithm similar to Algorithm 2. We iterate through each chromosome as we did before, storing the entire sequence as an array of strings, placing alternate alleles in the place of reference alleles, or in the case of deletions, replacing reference alleles with blank strings. Once the array is populated, we advance through the coverage table one position at a time. At each position, we emit the contents of the previously constructed table with the number of reads, fragments, and errors for each. At some positions, insertions will have changed the length of the sequence in the bin, and the extra positions will not have explicit read and fragment information to emit. Rather than simply reusing the read, fragment, and error counts for each position (which would have the unrealistic effect of keeping these measures constant over the length of the variant), we sample these measures from the local distributions. For a variant length l, we examine the reads, fragments, and error counts for a window of size l immediately preceding the location where the variant is to be placed. Over those intervals, we assume these metrics to be normally distributed. We compute the mean and standard deviation for each of these measures in the window and draw a random integer from each appropriately parameterized normal distribution.

To account for this, we keep a tally of the number of statistics on previously seen read and fragment statistics from which we can compute running means and standard deviation. We generate new values for the number of reads, fragments, and error rates as necessary as the mean plus standard deviation of the relevant metric, ensuring the values are never less than 0, and that we round up or down to the nearest integer (error rates are rounded up and read/fragment counts are rounded down to increase our self-penalty).

This procedure is described in Algorithm 4.

Algorithm 4 Lift a table from reference to child coordinates.

```
1: function LiftoverFromRefToChild(ref, vcf, table)
2:     for all chr in ref do
3:         newchr = IncorporateVariantsIntoGenome(ref, vcf)
4:         for i in 0 : newchr.length() do
5:             emit(table[i])
6:             if newchr[i].length() > 1 then
7:                 for j in 1 : newchr[i].length() do
8:                     emit(extrapolate(table[i]))
```

3.4.2 *Constructing the read profile*

Next, we generate the read profile, describing the various properties of fragments and read. As in previous examples, we iterate over each read in the exemplar sample's BAM file, storing relevant information in tabular form.

3.4.2.1 *Scalar properties*

We first store the following scalar information (as the data we are modelling is assumed to be Illumina data generated from a single library, some properties which could otherwise be stored as distributions are instead treated as a single number that will be simple constants in our simulation):

1. read length
2. number of reads
3. number of reads with errors
4. number of read pairs
5. number of chimeric pairs (each end aligned to different chromosomes)
6. number of mismatches
7. number of insertions
8. number of deletions

Example values for each of these metrics can be found in Table 3.4. Note that some of the mismatches, insertions, and deletions may represent true variation between the sample and the reference sequence. We do not mask these sites out. While this increases the apparent error rate, the variant rates are typically low compared to the number of bases in the reference sequence itself, making the difference negligible. Furthermore, we will choose the reference sample as the exemplar dataset for the read simulations. Since this sample should theoretically be identical to the reference sequence, this choice obviates the need for any masking of variants in the reference sample against the reference sequence.

Table 3.4: Example read and fragment scalar properties for sample PG0063-C.

	PG0063-C
readLength	76
numReads	38,846,418
numReadsWithErrors	3,696,708
numPairs	19,473,634
numChimericPairs	348,294
numMismatches	6,525,504
numInsertions	139,941
numDeletions	317,076

3.4.2.2 Empirical distributions

Other properties take on a range of values with some probability. Most simulation software attempts to prescribe distributions from which these properties should be drawn, or require a user to specify them as input parameters. Rather than trying to fit the underlying distributions explicitly (which can often fail as datasets contain more nuance than what sequencing platforms should theoretically produce), we instead store empirical distributions and generate random values from them accordingly. We store the following empirical distributions:

1. number of errors (of any type - mismatch, insertion, or deletion) in a read
2. fragment size
3. insertion size
4. deletion size

Note that these distributions are not conditioned on position in the read. Example distributions derived from an exemplar sample are shown in Figure 3.8.

3.4.2.3 Covariate table

Finally, we construct the covariate table: a set of empirical distributions for various error events conditioned on the following:

1. type (SNP, insertion, deletion)
2. end of pair (first end or second end)
3. strand (positive or negative)

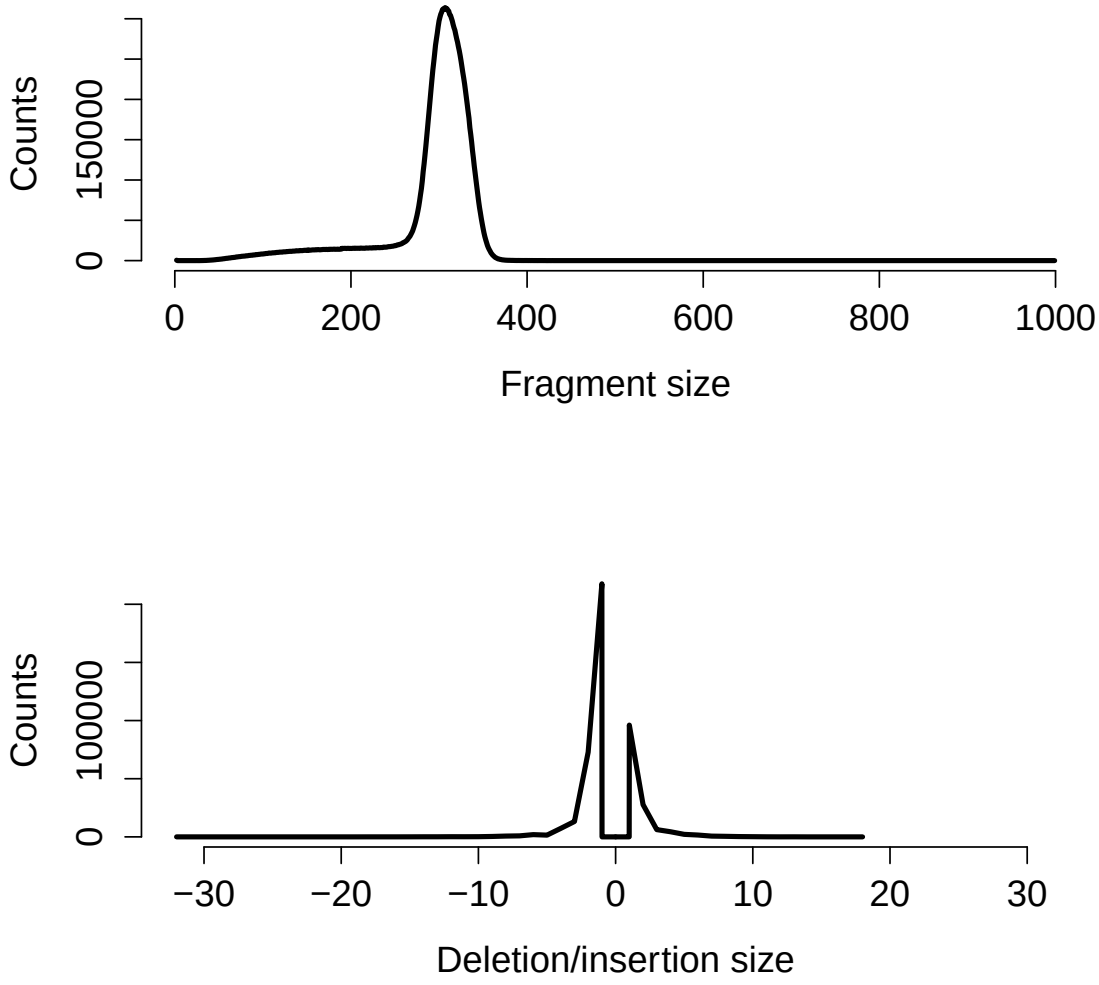


Figure 3.8: Top: empirical fragment size distribution for PG0051-C. Bottom: empirical deletion (negative values) and insertion (positive values) length distribution for the same sample.

4. 5′ dinucleotide context
5. position in read
6. first base of error sequence (empty for deletions)

For each read, we increment an element in a table that corresponds to these six covariates. The code to do so is contained in our `GenerateReadSimProfile` module.

3.4.3 *The read simulator*

With the coverage and read profiles in hand, we are finally ready to simulate reads. We advance through each base of the simulated genome, reading the coverage profile as we traverse. At each site, we use the coverage profile to dictate the number of fragments that must be generated and how many of those fragments should contain errors, thus modelling regions of the genome with higher or lower error rates. We then generate fragments, sampling fragment lengths from the empirical fragment length distribution. If a read is to contain an error, as specified by the coverage profile, we construct a read-specific error profile for mismatches, insertions, and deletions. These profiles take into account the preceeding dinucleotide context for each position in the read[1], the end of the pair being simulated, whether the fragment came from the positive or negative strand, and the first nucleotide of the error to be incorporated (not applicable for deletions). Figure 3.9 shows reads from a real sample and simulated data from an artificial sample over a 4 kbp region.

3.5 *The simulated dataset*

With our genome and read simulation framework now complete, we simulated the genomes of 20 progeny from a pseudo 3D7xHB3 cross. The precise counts of variant types per sample are shown in Table 3.5. For each sample, we generated two datasets: a so-called "perfect" dataset (uniform coverage, perfect reads), and a so-called "realistic" dataset (non-uniform coverage, imperfect reads). Both datasets contain approximately $150x$ coverage.

3.6 *Summary*

Unlike many other genome and read simulation software packages, our framework is able to generate realistic genomes and reads that exhibit allelic recombinations, non-allelic recombinations, a wide (and tuneable) spectrum of *de novo* variants, and reads with realistic sequencing properties. Rather than requiring a user to specify recombination rates, coverage levels, insert size distributions, and sequencing error rates (all of which may deviate wildly from real data), we have adopted the point of view that the end goal is to simulate data similar to a dataset one already possesses. By drawing from empirical distributions derived from real data, we should find that the performance of our software on simulated data is informative of its capabilities on real data.

[1] To handle the first and second positions in the read, which do not have a dinucleotide context, we simply extract a read by starting two nucleotides into the simulated fragment.

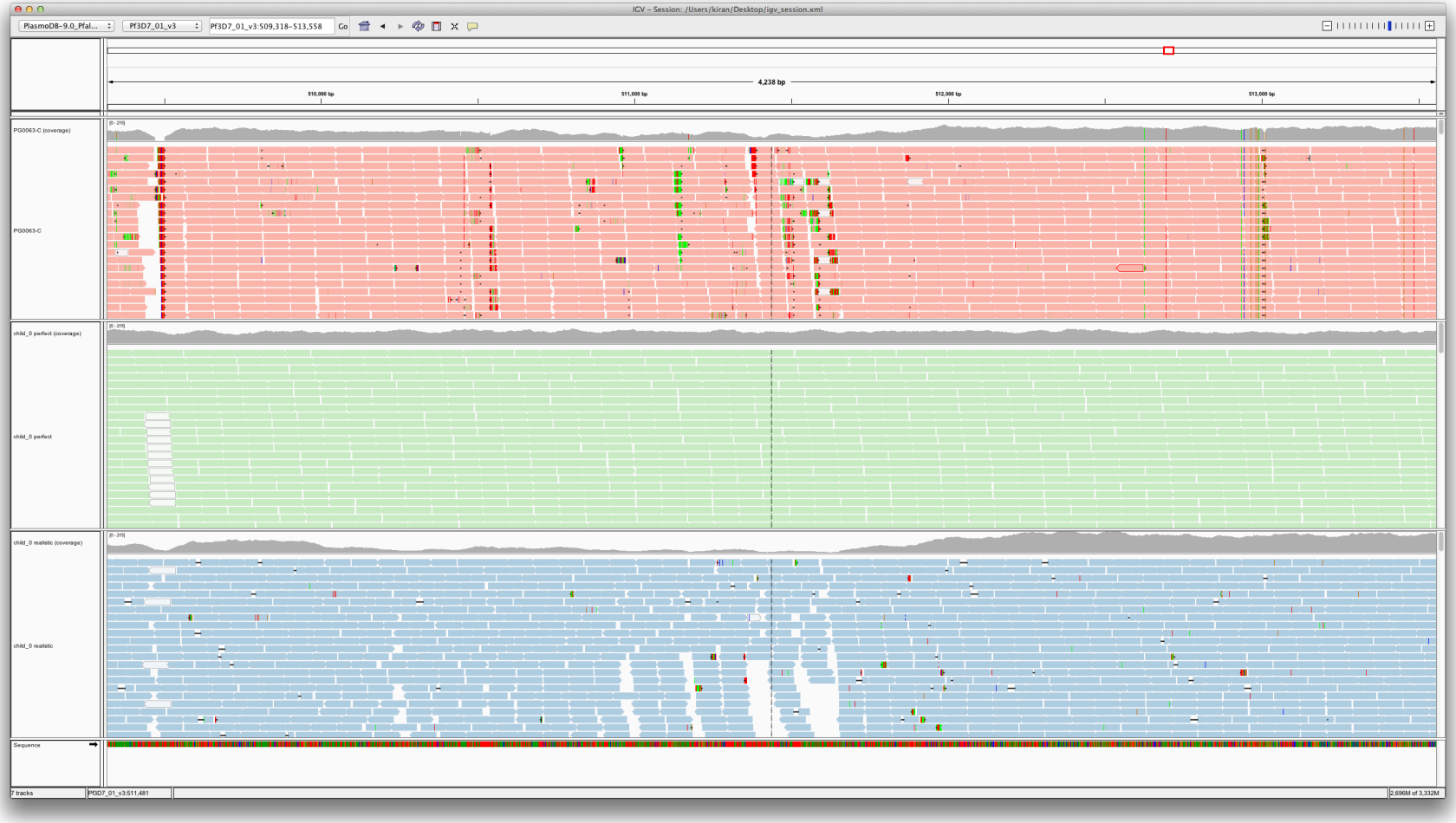


Figure 3.9: Read datasets for real (top panel), simulated perfect (middle panel), and simulated realistic (bottom panel) data.

Table 3.5: *De novo* variant counts for each of the 20 simulated children.

	DEL	GC	INS	INV	NAHR	RECOMB	SNP	STR_CON	STR_EXP	TD
0	22	25	22	22	2	10	5	3	3	26
1	21	21	21	21	2	10	23	11	11	26
2	11	22	11	11	2	8	13	22	22	8
3	25	17	25	25	2	32	23	17	17	21
4	4	23	4	4	2	8	14	16	16	12
5	26	23	26	26	2	37	28	5	5	8
6	13	17	13	13	2	15	27	23	23	4
7	1	22	1	1	2	30	28	5	5	12
8	26	16	26	26	2	16	16	13	13	8
9	4	23	4	4	2	25	23	23	23	21
10	12	20	12	12	2	16	23	19	19	27
11	13	19	13	13	2	7	20	18	18	17
12	19	20	19	19	2	8	14	5	5	28
13	10	20	10	10	2	25	13	28	28	11
14	28	18	28	28	2	8	3	29	29	17
15	27	22	27	27	2	30	5	19	19	2
16	23	22	23	23	2	12	29	24	24	17
17	17	22	17	17	1	10	28	13	13	11
18	23	18	23	23	2	8	1	21	21	18
19	22	22	22	22	2	17	19	3	3	2
total	347	412	347	347	39	332	355	317	317	296

We do note some minor errors in our simulation. When producing empirical distributions for the lengths of indel errors, we treated insertions and deletions as separate events needing separate distributions. However, inspecting these distributions more closely, we notice a bias: more deletion errors are present than insertion errors. This is likely to be an artifact of read mapping, rather than of biology or even of sequencing. Reads with insertions are typically more troublesome to place on a reference genome correctly than reads with deletions. Still, it would have been more appropriate to sample insertion and deletion error lengths from the same distribution. This error is not expected to compromise the results from the simulation.

Furthermore, no effort was made to match the orientation of the replacement *var* gene with the replaced *var* gene. This has the effect of inadvertently turning upstream promoters into downstream promoters, and vice versa. This is not expected to have any impact in our ability to detect NAHR events between *var* genes.

One interesting limitation of our simulation software is the inability to recapitulate some seemingly correlated errors in the real data. This is evident in the top panel Figure 3.9, where many mismatches seem to cluster together. This is quite likely the result of recurrent sequencing errors that we have not adequately modeled. More work can be done on this to capture this aspect of the data. We anticipate that this would make our variant calling results on simulations slightly over-optimistic when compared to performance on real data.

Many entries in Table 3.5 are identical. This is because when choosing how many indels to simulate, we reused the random number chosen to apply for other variants under that umbrella (insertions, deletions, inversions). A similiar consideration was made for the short tandem repeat expansions and contractions.

Finally, we note a deliberate omission: sample contamination. Sample contamination by non-*P. falciparum* data could generate false positive calls if, for instance, a child's sample is contaminated but the parents are not. This point will be discussed in greater detail when examining validation data in Chapter 5.

4 Detection and classification

4.1 Introduction

We now present a novel graphical method for detecting and classifiying *de novo* variants. We shall present some brief foundational background on *de novo* assembly and graphical models, concepts upon which the variant caller is built. The simulation framework and dataset we described in Chapter 3 will be used to establish the method's accuracy.

Reference-based analyses benefit from being reasonably intuitive, as the underlying genome is linear and there are many visualization tools that can facilitate the user's understanding of the underlying data.[87–91] In contrast, graph data structures are far less intuitive and lack dedicated visualization tools in the genomics space. In this chapter, we will also present images from our purpose-built graph visualization software for the purposes of guiding the reader's intuition regarding graphical variant motifs and the operation of the calling algorithms.

4.2 De novo assembly

De novo assembly refers to the process wherein a sampled genome is reconstructed from sequencing data without guidance from an existing genome. Sequenced deeply enough, a set of reads from a single sample will contain overlaps that can be used to infer the sequence of contiguous segments of the genome into so-called **contigs**. If the reads are long enough to span repetitive regions in the genome, these contigs could be as long as the chromosome itself. Otherwise, separate contigs can be joined together into **scaffolds** (contigs plus gaps) using paired-end reads, long reads, information from linkage analysis, Hi-C data,[92] or other methods.

The quality of the reconstruction can vary considerably depending on how the assembly is performed. Typically, constructing a complete and high-quality reference sequence (one with few errors and few gaps in the linear sequence of each assembled chromosome)

Table 4.1: Comparison of assembly statistics of the finished 3D7 reference genome and a reconstruction of the same sample from 76-bp Illumina data.

	contigs	min length	max length	mean length	N50	total sequence
3D7	16	5967	3291936	1458302	1687656	23332831
PG0051-C	51425	47	27520	478	1280	24602599

is a big first step to studying the genome of an organism, done at great expense and labor over a period of several years. In 2002, after several years of work, Gardner *et al.* published the sequence of the 3D7 *P. falciparum* parasite clone. By using a whole chromosome shotgun strategy with accurate and long (several hundred bp) first-generation sequencing reads (Sanger), each chromosome was fully resolved with very few gaps in the sequence and no unplaced contigs.[34]

Today, the same sample can be assembled in a matter of days rather than years using high-throughput second-generation sequencing. Short reads ($76 - 100$ bp) from second-generation sequencers (Illumina) can be produced cheaply and in abundant quantities, and similar read overlap considerations can be used to generate assemblies. However, these reads tend to be much more error prone, and the shorter read length fails to span many repetitive regions of the genome. The result is a much cheaper, but far more fragmented assembly than the reference assembly. Table 4.1 compares the assembly statistics of the reference genome versus a reconstruction of the same sample from Illumina data taken from our cross dataset (PG0051-C) using the McCortex assembly software.[93] While the parasite genome is known to have 16 chromosomes (which the reference recapitulates), the Illumina assembly broken into over $50,000$ pieces with an N50[1] of a mere $1,280$ bp.

It is perhaps surprising then to discover the central focus of this dissertation is on the value of *de novo* assembly of short reads for *de novo* mutation discovery. After all, a poor-quality Illumina assembly - as evidenced by the numerous short contigs - would seem to be very limiting, even if it could theoretically reconstruct pieces of the genome that cannot be recovered from a reference-based analysis. However, for the work that we describe in this chapter, the value is not in the contig reconstructions, but in the underlying data structure from which the contigs are produced: the "de Bruijn graph".

The de Bruijn graph is a data structure that encodes read overlaps such that a walk through the vertices with unambiguous connections yields contiguous sequence in the genome. In real data, those walks may end prematurely due to errors or homology,

[1] **N50**: the length for which the set of all contigs that length or greater represents half the total lengths of all contigs.

yielding junctions that cannot be confidently navigated due to the risk that the traversal would never end. Typically, assembly algorithms will be conservative in the contigs they emit from these data, choosing to terminate the walk at the junction rather than risk going forward and either emitting sequence that does not actually appear in the genome, or getting stuck in an infinite loop because there are too many branches to explore. Searching all available branches is an incredibly expensive operation. However, with an additional set of metadata and a reasonable assumption described below, we can reduce the amount of searching we are required to perform in the graph. Consider that in trio data, it is easy to detect a so-called novel kmer: a kmer occuring in the child but absent in the parents. By annotating the graph with this information, we can see stretches of novel kmers. Figure 4.1 shows a portion of such an annotated graph around a small variant between two samples, depicted with blue edges (a parent) and red edges (child). We refer to this particular graphical motif as a "bubble", owing to the two branches that separate at A and rejoin at H, forming a bubble-like shape. To discover and type the variant in the graph, one must traverse each side of the bubble until the paths rejoin. Ignoring the annotations, a traversal on the blue sample that starts at vertex A would yield a short sequence: $A \rightarrow B \rightarrow C \rightarrow D \rightarrow H$. On the red sample, however, we see an ambiguity. After reading $A \rightarrow E \rightarrow F$, we hit a junction. It is not clear if the next vertex in the variant is G or I. Such an ambiguity may arise from uncleaned sequencing error. Faced with such error, it is not immediately clear what the boundaries of the variant are and which vertices we're aiming to rejoin on either side of the branch. Searching the I branch may be very expensive as there is no way to tell that it will eventually hit H until we look. However, if we assume the novel kmers link together, we can extend our traversal to yield $A \rightarrow E \rightarrow F \rightarrow G \rightarrow H$, ignoring the I branch completely. These novel kmers are the hallmark of *de novo* variation, and by utilizing these annotations, we can navigate parts of the graph that are otherwise unnavigable. This allows us to discard branches of the graph from consideration, enabling us to overcome sequencing error and surpassing what conservative traversals can give us.

4.2.1 Definitions

Let us establish some definitions. We denote a **graph** as $G = \{\mathcal{V}, \mathcal{E}\}$ where $\mathcal{V}$ represents a unique set of **vertices**, and $\mathcal{E}$ a set of **edges**.[94] We shall assume the set of vertices $\mathcal{V} = V_1, V_2, ..., V_n$. A pair of vertices may be connected by an edge. For the purposes of this dissertation, we shall require that all edges are **directed** (though for some applications outside this work, it is also possible for edges to be **undirected**). Written as $V_i \rightarrow V_j$ (or equivalently $V_i \leftarrow V_j$ and $V_j \rightarrow V_i$), directed edges restrict graph traversal in one direction

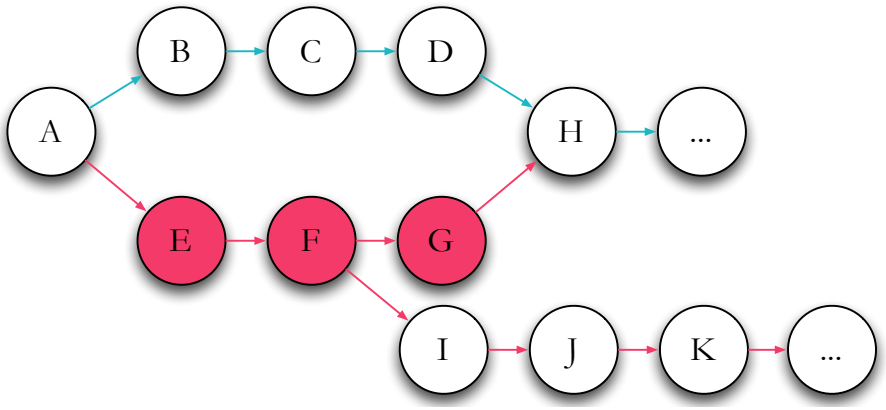


Figure 4.1: Graph for a variant between a parent (blue edges) and a child (red edges), with novel kmer annotations in red.

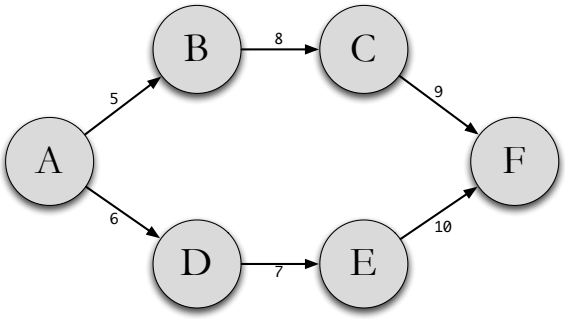


Figure 4.2: An example directed graph with six vertices, six edges, and coverage information written on each edge.

(thus enforcing a traversal order when reconstructing a region of the genome). A vertex may have a number of incoming (outgoing) edges, the precise count being referred to as a vertex's **in- (out-) degree**. We shall also refer to any vertex with an in-degree or out-degree greater than 1 as a **junction**.

A **path** is a sequence of adjacent vertices that respects these edge relationships, i.e. $V_i, ..., V_k$ such that for every $j = i, ..., k$, we have $V_j \rightarrow V_{j+1}$. In Figure 4.2, the vertex sequence A, B, C, F forms a path. Similarly, a **trail** denotes a sequence of adjacent vertices, but does not respect the directionality of the edges. In Figure 4.2, the sequence F, E, D, A forms a trail. Edges can optionally carry **weight**, indicating an arbitrary cost for traversing from one vertex to another via particular edges. We shall assume the edge weight is always 1 (however, see Chapter 8 for a discussion on circumstances where weights might be a fruitful parameter in a graphical model of sequence assembly).

A **multi-color graph** is a useful extension for multi-sample scenarios. Each edge carries additional metadata used to denote sample identity (e.g. each sample is assigned a unique "color"). All kmers in all samples are added to the graph, and following the

edges with the same color yields the genome of that sample. Many kmers will not be accessible when traversing paths or trails in one color versus another. These motifs are the hallmark of most variation in a graphical setting.

Often in this dissertation, we will perform operations on a limited graph region wherein we process only vertices of interest and all edges present between them in the original graph. This is referred to as an **induced subgraph** (which is *not* technically synonymous with a subgraph, but for simplicity in this work and consistency with terminology in the literature, we will often drop the "induced" qualifier). Let $\mathbf{V} \subset \mathcal{V}$. The **subgraph** $G[\mathbf{V}] = \{\mathbf{V}, \mathcal{E}'\}$ where $\mathcal{E}'$ are all the edges $V_i \rightleftharpoons V_j \in \mathcal{E}$ such that $V_i, V_j \in \mathbf{V}$.

A **de Bruijn** graph represents fixed length symbols (vertices) and their overlaps (edges), with the added restriction that each symbol may only appear once in the graph.[95] Applied to assembly, each symbol is taken to be a genomic sequence of length k (or **kmer**), with edges connecting vertices overlapping by $k-1$ bases[2].

To construct the initial graph, each sequenced read is broken up into $l-k+1$ kmers (where l is the read length) and added to the graph as vertices, with edges being placed between adjacent kmers. Overlapping reads should share kmers, and the adjacency information stored in the graph can be updated accordingly. Thus processing one read at a time in this manner will yield a graph of overlaps for the entire dataset.

Connectivity information beyond $k+1$ bases is not stored a in de Bruijn graph; it is the fundamental limitation of de Bruijn graphs that such connectivity information is lost when reads are decomposed into kmers to construct the graph. Only the kmer adjacencies are stored, which means a de Bruijn graph alone encodes no information about a kmer's relationship to another kmer seen in the same read but separated by two or more bases. This may limit us on occassion if we encounter a repetitive region that is longer than our kmer size. This also provides some insight into why the kmer size is such a critical parameter: it must simultaneously be small enough that it captures enough true genomic sequence (i.e. error free subsequences in the read) and must be large enough to implicitly encode connectivity information within a kmer and overcome many small genomic repeats. Work to incorporate all connectivity information in reads by means of an auxillary file of annotations specifying junction choices supported by reads spanning repeats in the graph is ongoing by a colleague in the lab, but is not ready for use as of this writing.

Additionally, the de Bruijn graph does not encode information about base quality scores, as each kmer is only represented once in the graph no matter how many times it

[2]According to the original mathematical definition, a de Bruijn graph should represent *every* possible symbol of length k, which makes the assembly formulation a subgraph rather than a graph. However, none of the relevant literature refers to assembly graphs in this manner, so we shall carry on abusing the term.

appears in reads, storing all base quality scores from every instance of the kmer would be prohibitively expensive.

4.2.2 *Overview of the Cortex graph format*

We rely on the Cortex assembly software for construction of the graph[3]. Cortex represents the graph on disk as a list of fixed-width records consisting of a odd-length sequence of length k or its reverse complement, whichever is alphanumerically lower. Each record also stores coverage in each color, and a list of incoming and outgoing edges. The edge lists utilize a minimalist description. For incoming edges, only the first base of the preceeding kmer is stored; the rest of the kmer is assumed to be the 0 to $k-1$ substring of the current kmer. Likewise, for outgoing edges, the subsequent kmer is assumed to be the 1 to k substring of the current kmer plus the last base of the next kmer. Figure 4.3 lists three example records. In the first record, the specified kmer has coverage 9 in color 0 (for our purposes, the child), 8 in color 1 (mother), and 5 in color 2 (father). The trailing three columns encode edge information per color. The first four characters of each column denote incoming edges to kmers with A, C, G, or T prefixes (or "." for no edge to that prefix). The last four characters of each column denote outing edges to kmers with A, C, G, or T suffixes (or "." for no edge to that suffix).

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATATATGTATATATATA 9 8 5 a......T a......T a......T
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATATATGTATATTAACT 7 6 3 a...A... a...A... a....C..
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATATATGTATGTATATA 4 9 0 a......T a......T ........
```

Figure 4.3: Three example Cortex record entries, each showing a kmer, coverage, and edge information across three samples.

The problem of building the entire graph decomposes to the problem of creating one of these records for every single read or contig in the input dataset. Iterating through every kmer within every read/contig is straightforward.

The Cortex de Bruijn graph can be easily represented as a hash table where each key is a kmer and the associated value a list of coverage levels and incoming/outgoing edges per sample. This enables lookups to be performed in $\mathcal{O}(1)$ time, but requires the entire graph be loaded into memory. While this makes sense from the perspective of a software developer writing assembly software (where one would assume that every kmer will be processed during the lifetime of the program's execution), it is cumbersome for our purposes. DNMs are anticipated to be scant, which implies that the number of kmers that need to be examined is modest. Furthermore, should we wish to scale to larger

[3]Or equivalently, the McCortex software, the next version of the Cortex software and 100% compatible in terms of its graph construction procedure and output format.

genomes in the future (e.g. mammalian genomes on the order of 3 gigabases), we must be aware of the much greater difficulty in loading such a graph entirely into memory (unless one happens to be operating on a computer with several hundred gigabytes of available RAM). Instead, we lexicographically sort the graph records based on the kmer of the record and perform a binary search whenever we need to load the record associated with a particular kmer. This increases the time complexity for kmer lookups to $\mathcal{O}(\log n)$, but eliminates the need to load the entire graph to memory.

4.2.3 The Java-based Cortex graph API

We now introduce our Cortex graph Application Programming Interface (API). Our API is a Java-based, object-oriented library that forms the basis of the *de novo* mutation caller and visualization software introduced in this chapter.

4.2.3.1 CortexKmer

The lowest level data structure in our API is a **CortexKmer**. A CortexKmer represents the lexicographically lowest version of an arbitrary input kmer. On initial construction of this object, both the forward and the reverse-complemented version of the input kmer are compared, and the lexicographically lowest version is retained. The object also keeps a boolean value set to `true` or `false`, depending on whether the input kmer was reverse-complemented or not.

4.2.3.2 CortexRecord

The CortexRecord is a composite data structure including a CortexKmer, an array of integers representing coverage per sample (or per "color" in Cortex parlance), and an array of strings representing incoming and outgoing edges.

Two utility functions are provided by the CortexRecord object to facilitate graph traversals. These methods, **`prev`** and **`next`** retreive lists of kmers connected to incoming and outgoing dges of the specified kmer, orientation-matched to the input kmer.

We define a series of utility functions that aid graph traversal. The **`find`** method retrieves the CortexRecord indexed by a kmer via a binary search over the sorted disk-based graph, or throws an error if the graph is not sorted. The CortexKmer within the CortexRecord, in addition to carrying the text of the kmer, also stores a bit that indicates whether the stored orientation matches the requested orientation, which is useful for navigation. The methods **`prev`** and **`next`** retrieve lists of kmers connected to incoming and outgoing edges of the specified kmer. Algorithm 5 shows our procedure for generating the next kmers from the current CortexRecord. Briefly, we first create an empty list. We

then extract a substring from the kmer (the last $k-1$ bases). We iterate over all edges for that sample (color), concatenating that to the of our extracted kmer. We then store this new kmer in the list. When all edges are processed, we return the populated list. If no outgoing edges are present, the list will be empty.

Algorithm 5 Get next kmers for a color, c, from CortexRecord, rec

```
1: function NEXT(rec, c)
2:     nextKmers = []                                   ▷ Create an empty list
3:     partialKmer = record.kmer[1..k]   ▷ Remove the first (left-most) nucleotide from
   the kmer's sequence
                                                        ▷ Iterate over edges
4:     for edge in rec.outedges[c] do
5:         newKmer = concatenate(partialKmer, edge)
6:         add newKmer to nextKmers
7:     return nextKmers
```

The **`prev`** method works similarly, as shown in Algorithm 6.

Algorithm 6 Get previous kmers for a color, c, from CortexRecord, rec

```
1: function PREV(rec, c)
2:     prevKmers = []                                   ▷ Create an empty list
3:     partialKmer = record.kmer[0..k-1] ▷ Remove the last (right-most) nucleotide from
   the kmer's sequence
                                                        ▷ Iterate over edges
4:     for edge in rec.inedges do
5:         newKmer = concatenate(edge, partialKmer)
6:         add newKmer to prevKmers
7:     return prevKmers
```

4.2.4 CortexGraph

The CortexGraph is a simple abstraction to the sorted graph file, enabling random-access to records based on a binary search of kmers in the graph. The CortexGraph object provides **`find`** method to do this, which takes as input a kmer and returns the relevant record or `null` if the record is not found in the file. It also provides trivial convenience accessors to graph metadata (i.e. kmer size and number of colors in the file).

4.2.5 CortexUtils

The previous objects and methods enable access to single records at a time, without regard to context. That is, because Cortex stores the alphanumerically lowest version of a kmer, and its edges match that orientation, we may often find ourselves in a situation

where walking from one kmer in the graph to the next suddenly takes us to a record in the forward orientation to another record in the reverse orientation. We simply our work by providing the CortexUtils object, which houses a set of methods aimed at simplifying traversals.

The `nextKmers` method provided by this object return the next kmers, matched to the orientation of the input kmer. To do so, we obtain the CortexRecord for a kmer and its outgoing edges. If the CortexKmer in the rec object has been reverse complemented from our input orientation, we replace our list of outgoing edges with reverse-complemented incoming edges. We then proceed to construct the concatenations of the $k-1$ bases of the input kmer and the outgoing edges. This final list is returned to the caller of the method. An identical procedure is provided for `prevKmers`, but reversed.

Algorithm 7 Get next kmers for a color, c, matched to expected orientation

```
1:  function getNextKmers(g, kmer, c)
2:      rec = g.find(kmer);              ▷ Find the CortexRecord for the given kmer
3:      edges = rec.outedges[c]
4:      if rec's CortexKmer is flipped then
5:          edges = rec.inedges[c]
6:          reverse complement edges
                                                              ▷ Iterate over edges
7:      for edge in edges do
8:          newKmer = concatenate(partialKmer, edge)
9:          add newKmer to nextKmers
10:     return nextKmers
```

For convenience, we provide these methods in two flavors: methods that operate on a single graph, and methods that operate on two graphs: a **clean** graph and a **dirty** graph. The clean graph represents data where errors have been removed by the assembly software's automatic filtering process (coverage and contig length considerations). The dirty graph represents data prior to this cleaning step. When a kmer is absent from the clean graph or lacks incoming/outgoing edges, traversal can still continue if the dirty graph contains the requested kmer and edge information. This overcomes an issue wherein *overcleaned* data (data filtered with too stringent a threshold) causes erroneous gaps in the graph. If carefully used, the dirty graph can be used to patch these holes without causing significant errors in the traversal. Almost all methods with this double graph signature have a form that look like Algorithm 8, wherein a single graph version of the same method is called, only to be called again if the operation on the clean graph was unsuccessful.

Algorithm 8 Get next kmers from the clean graph, failing back to the dirty graph

```
1: function getNextKmers(clean, dirty, kmer, color)
2:     nextKmers = getNextKmers(clean, kmer, color)
3:     if (dirty != null && nextKmers.isEmpty()) then
4:         nextKmers = getNextKmers(dirty, kmer, color)
5:     return nextKmers
```

Because this form of falling back to the dirty graph when data is unavailable in the clean graph is so common in our functions, we shall take for granted that all methods implement this fallback. We shall omit writing it explicitly in all future code examples for clarity.

Finally, CortexUtils provides a depth-first search function, used to traverse the graph over great distances, and an implementation of Dijkstra's shortest path method for discovering the lowest-cost route between a source vertex and a destination vertex. These are detailed in the sections to follow as we discuss our variant-calling strategy.

4.3 Variant motifs

Just as reference-based methods search for motifs in the data representing variants (e.g. recurrent mismatches, gaps, or unusual truncations in the read alignments; read pairs aligning much further apart than expected; chimeras or inter-chromosomal alignments; etc.), so must we scan for indicative motifs in the assembly graph. Before we discuss the precise nature of these motifs, it is useful to draw a distinction between "simple" and "complex" variants. A "simple" variant is a SNP, insertion, or deletion that occurs within a single chromosome. A "complex" variant could be a homologous or non-homologous recombination, translocation, or other interchromosomal exchange - something that does not fit within the confines of the straightforward SNP or indel classification. The patterns inherent to these two categories of variants are very different.

4.3.1 Simple variant motifs

Simple variants in *de novo* assembly data typically manifest as "bubbles" in the de Bruijn graph: regions where a variant has broken the homology between sequences, resulting in flanking kmers that are shared between the samples and spanning kmers that differ through the variant itself. In a single diploid sample, this could be a heterozygous SNP or indel between two homologous chromosomes. In a collection of haploid samples, one or more samples may differ from the others, resulting in the bubble.

As an illustration, consider three sequences from a mother-father-child pedigree, shown in Figure 4.4a. While the maternal and paternal haplotypes (green and blue, respectively) are identical, the child's haplotype (red) differs by a single C to G SNP. Figure 4.4b shows the resulting multi-color de Bruijn $k = 3$ graph built from this data. The mutation has given rise to the canonical bubble motif in the graph. Three novel kmers (kmers present in the child and absent in the parents) spanning the variant allele are present. Figure 4.5 is an equivalent graph for another simulated SNP, shown with more context and constructed with a much larger value of k appropriate for 76 - 100 bp read lengths, typical of second-generation sequencing datasets (in this case, $k = 47$).

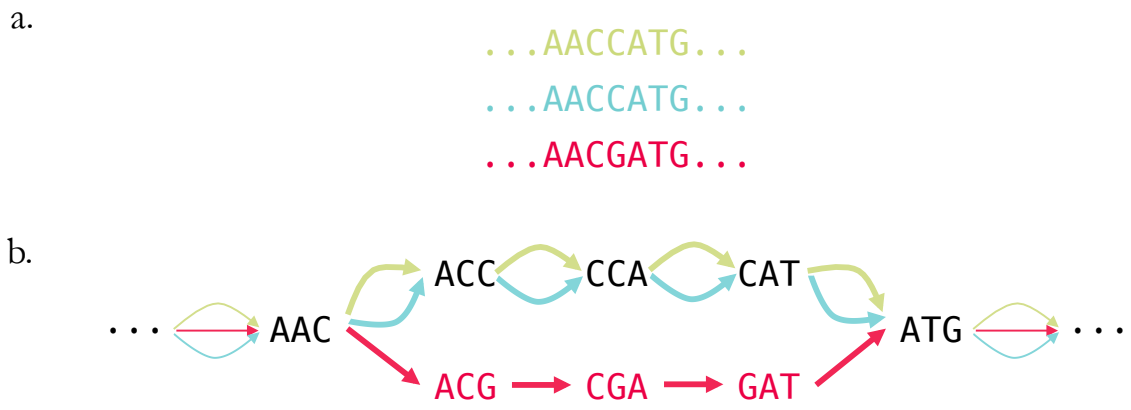


Figure 4.4: A simple variant motif for a C to G SNP. a. Haploid sequences from a mother (green), father (blue), and child (red), the last differing from the first two by a single base substitution. b. The resulting multi-color de Bruijn graph for $k = 3$. Red vertices denote kmers that are deemed "novel", i.e. present in the child and absent in the parents. Edge colors reflect the samples in which the connected pairs of kmers are found. Edges that are part of the bubble (variant call) are displayed with thicker lines.

All simple variants will have this basic structure: a bubble in the graph that separates the variant samples from the non-variant samples. The variants are distinguished by the length of the variant and non-variant branches. For an event that does not alter the length of the sequence in the genome, the lengths will be equal (for SNVs, the length of each branch is k vertices; for MNPs, it is $k + l - 1$, where l is the length of the MNP). For insertions, the length of the non-variant branch is k, while variant branch will be greater than k. For deletions, it is reversed: the non-variant branch is greater than k, while the variant branch has length k. This behavior can be observed in the graph visualizations. We do acknowledge, however, due to the force-directed graph layout which does not display data in the intuitive linear fashion to which we are typically accustomed, it is sometimes difficult to observe this at a glance. This is evidenced by Figures 4.6 and 4.7.

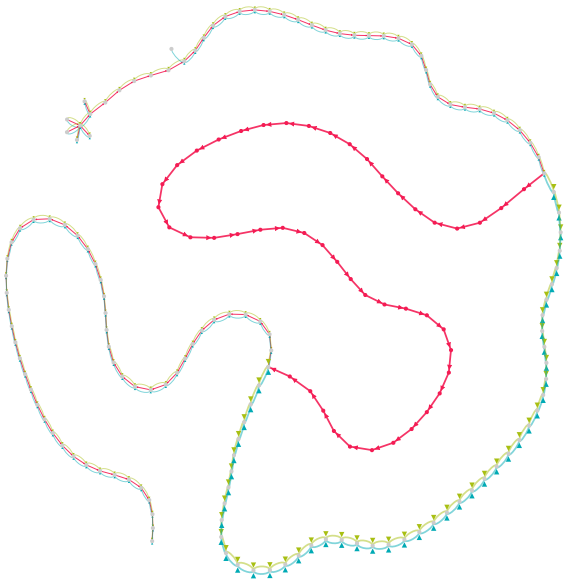


Figure 4.5: A multi-color de Bruijn graph at $k = 47$ for a haploid pedigree spanning a simulated *de novo* SNP, produced by our VisualCortex software. Vertex labels have been supressed for clarity. Spatial layout is arbitrary and for display purposes only.

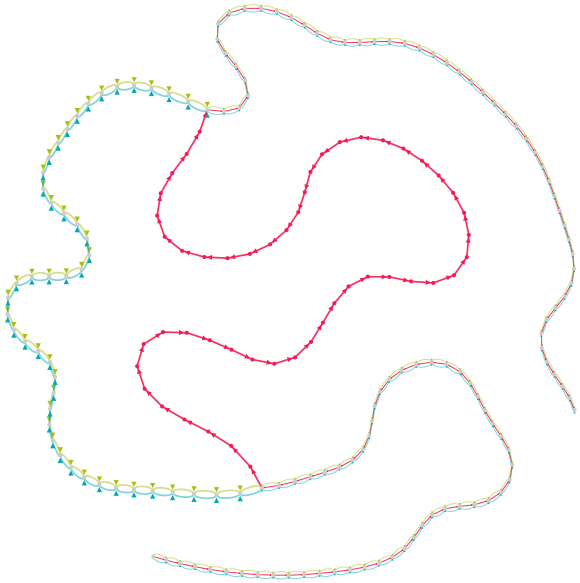


Figure 4.6: A 5 bp insertion in the child (red), with the two parents (blue and green) sharing a homologous and distinct allele, evidenced by the shared path taken by the parental samples and unshared path taken by the child.

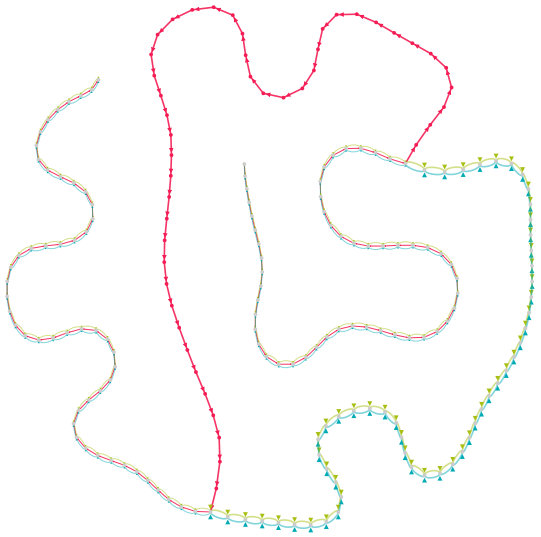


Figure 4.7: A 5 bp deletion in the child, with the two parents (blue and green) sharing a homologous and distinct allele, evidenced by the shared path taken by the parental samples and unshared path taken by the child.

Many variants may occur on the haplotypic background of one parent and not the other. This is common in regions of the genome that are divergent between the two parents. Figure **??** depicts one such simulated event. A 41-bp tandem duplication has occurred on the background of the mother (evidenced by the presence of green edges), but not the father (thus the absence of blue edges). In the flanking tails, edges shared between all three samples are present until a blue edge separates from the graph and connects to different vertices. While not shown, these branches continue along the genome of the father.

Finally, it is possible to encounter variants where the path through the graph taken by the child can appear to follow both the variant and non-variant paths, as demonstrated by Figure 4.9. Such a scenario may arise by a mutation on a sequence with copy number greater than 1: both the unaltered and altered sequences would then exist simultaneously in the child's genome.

4.3.2 Complex variant motifs

Recombination events (allelic crossovers or gene conversion events; NAHR events) will not necessarily appear as bubbles. Bubbles form in the graph when parental and progeny

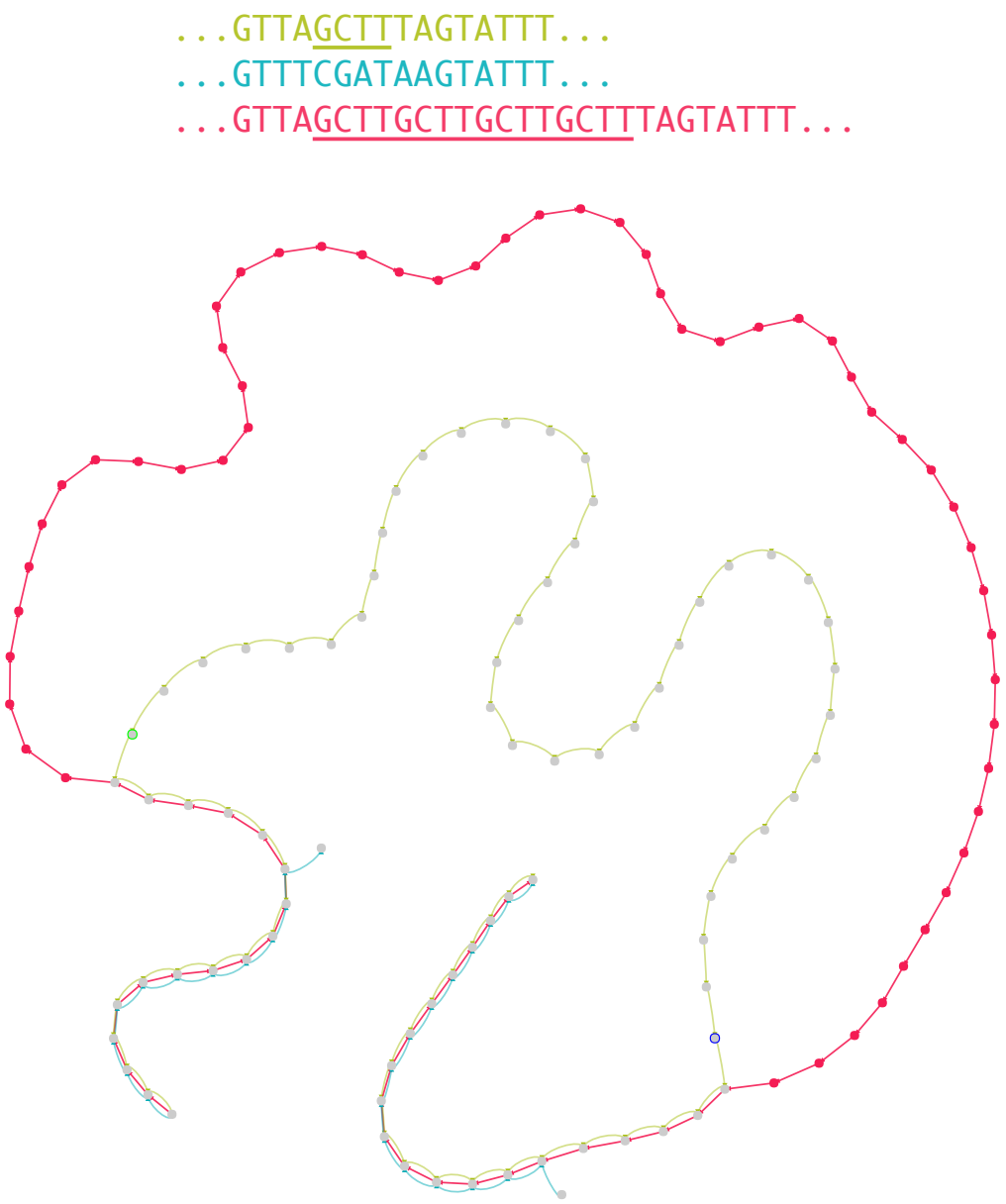


Figure 4.8: A *de novo* tandem duplication in the child, with parental and child sequences shown in inset above.

haplotypes diverge (at the site of a variant) and reconnect (at the flanking homologous regions). In a recombination event, the haplotypes do not necessarily reconnect. Consider the gene conversion event depicted in Figure 4.10. Initially, the progeny's graph follows both parents in the regions of homology. A variant between the parental haplotypes (A in the mother - green, C in the father - blue) causes the parental graphs to diverge. The child, having copied its genome from the mother at this point, initially tracks with her

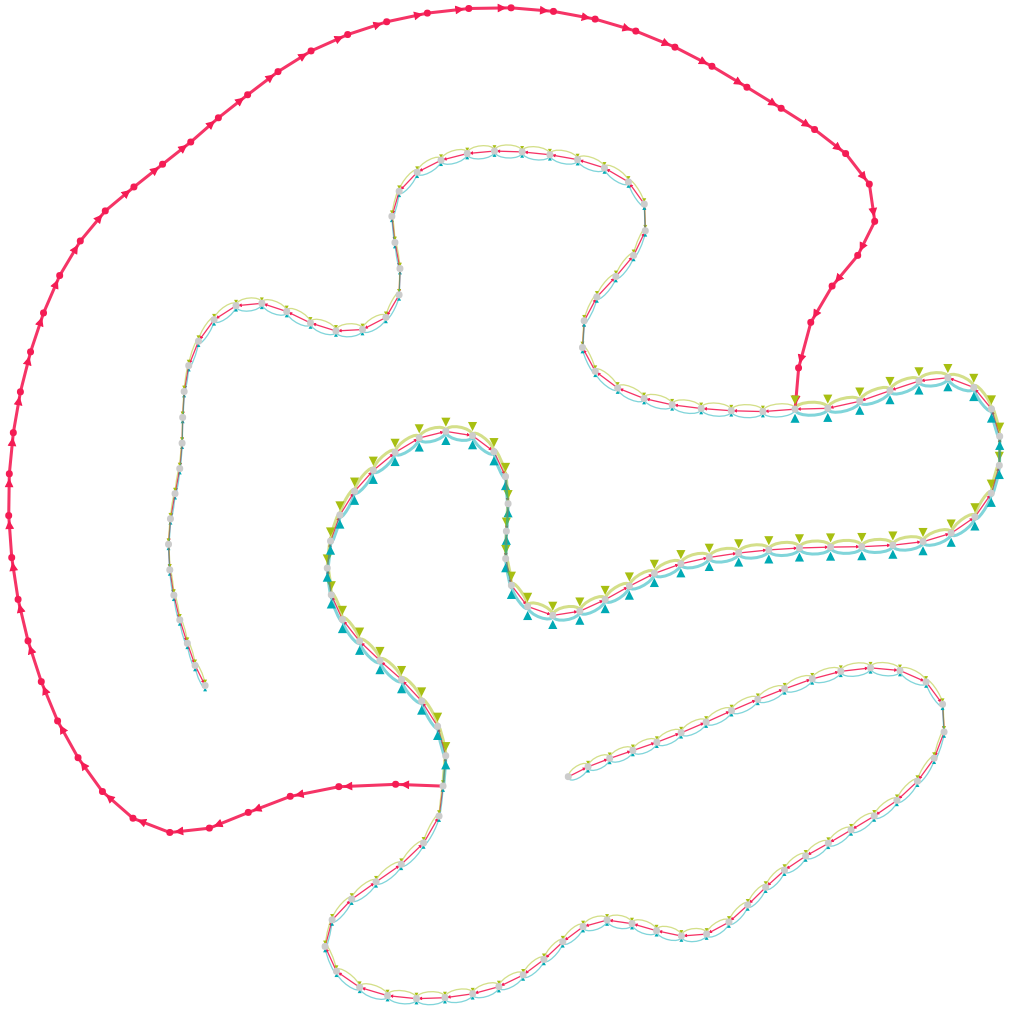


Figure 4.9: A variant wherein the child's path does not simply diverge from that of the parents, but rather navigates both.

genome (starting from the left-hand side of the figure). As we pass through the sequence, we encounter a switch: the apparent genome from which the child is copying is now the father, inheriting the father's T variant rather than the mother's C. The child's graph forks from the mother and tracks with the father. Finally, copying switches back to the mother, and the child picks up the mother's G variant rather than the father's A. The child's graph departs from the father's and tracks with the mother's once again.

Because our kmer size in this example is 7, two novel kmers are apparent: ATCAGCT (which spans the first and second variants), and TCCATAG (which spans the second and third variants). Neither of these kmers is present in the parents. However, had our kmer size been smaller than the inter-mutation distance, the kmers spanning each variant would not have been considered novel, and we would fail to identify this event.

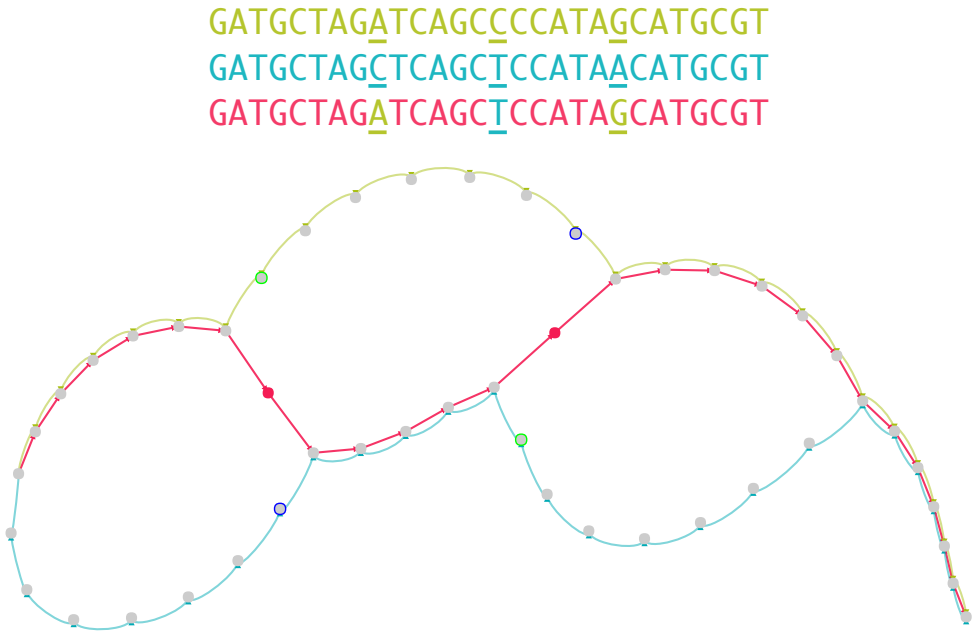


Figure 4.10: A gene conversion event in the child, switching between mother (green), father (blue), and back to mother. Parental and child sequences in the inset. For illustrative purposes, kmer size is 7.

We therefore caution the reader that it's quite likely that many events of this nature will likely go uncalled or improperly called. For such recombination events to be detected, a kmer spanning two proximal variants must be present. This is unlikely to occur for simple crossovers, particularly in genomes of reasonably low heterozygosity, as neighboring variants beyond a kmer's length away will not give rise to novel kmers necessary for the event's detection. It is perhaps slightly more likely for gene conversion events, where the multiple crossovers proximal to a variant exclusive to one of the parents will generate the sought-after novel kmer signal.

NAHR events are trickier; in *P. falciparum* and other small genomes, these events are generally mitotic rather than meiotic events. The expected motif is that a child's graph should follow the same parent, but connect disjoint components of the graph (e.g. telomeres of different chromosomes) through novel kmers. In principle, one should be able to detect such an event by testing whether removing the child's contribution to the pedigree subgraph results in disrupting the otherwise connected components.[96] In practice, however, NAHR events are mediated by homology between low-complexity regions of the subtelomeric genomic regions. The homology will result in confounding connections between disparate regions of the graph. An easier solution is to track the parent apparently being copied from and determine the chromosome of origin for each kmer present

in the flanking regions of the novel kmers. This is only feasible if one happens to have draft reference genomes for which a kmer's chromosome of origin can be approximately determined.

4.3.3 *Handling errors in sequencing*

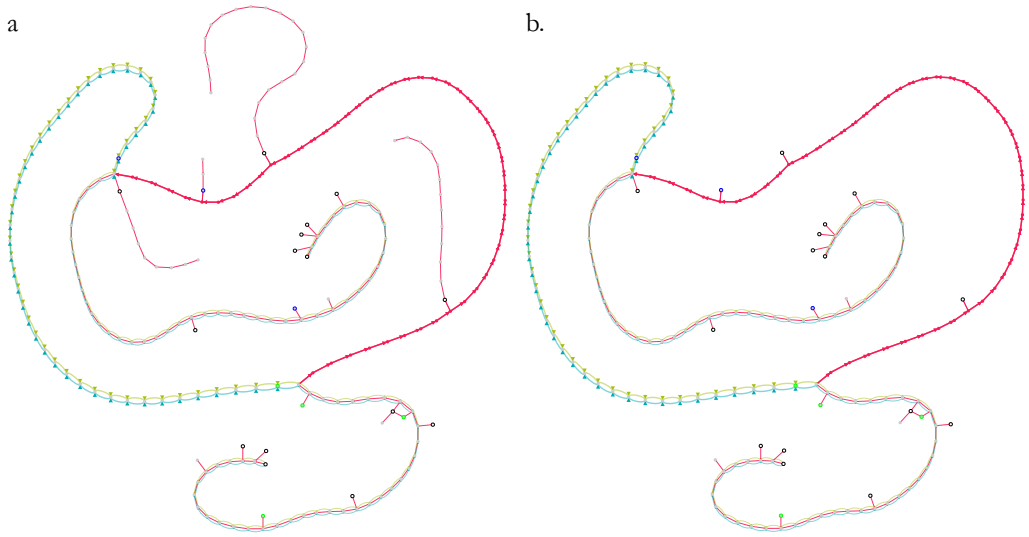


Figure 4.11: Indel with sequencing errors in graph. a. Selected extraneous branches expanded for illustrative purposes b. All extraneous branches collapsed.

Bubble and non-bubble motifs are trivial to find and navigate if there are no errors in the sequence data, but this situation rarely arises in real data. Real data is subject to sequencing errors, making graph traversal much more difficult. Errors manifest as extraneous branches in the graph, adding ambiguity to traversals. Without a guide as to which branch to explore at a junction, we are either forced to abandon the traversal, make a guess, or explore all possible branches. The first option is overly conservative; many variants will go untyped. The second is hazardous; there is the very real potential for choosing erroneously and typing the variant incorrectly. The last is computationally expensive; errors explode the complexity of the graph, making it prohibitively costly to find the correct path through the graph.

To solve this, recall that DNMs do more than open up a bubble motif in the graph. Ideally, each kmer along the variant path is novel. In the naive approach, if we begin walking in the graph along adjacent novel kmers and encounter a junction, we would need to explore all available branches from that point until we meet some stopping condition (e.g. reaching a kmer that's shared between child and parents). This could be very expensive if there are many branches to explore, and it's reasonable to assume that eventually, one may rejoin the parental genomes after walking sufficiently far anyway. Instead,

rather than exploring all possible branches, we examine each branch for a certain number of kmers (50 by default, but this is arbitrary and can be overriden by the user). If there are novel kmers along that branch, we assume the branch is part of the current variant being explored and we continue our traversal along this route. In this sense, we use the novel kmers as "sign posts", indicating the direction to walk in the graph. Navigating a branch and not seeing a sign post gives us adequate cause to abandon the branch in favor of another.

Figure 4.11a-b depict a simulated indel in imperfect data, the former with some of these extraneous branches displayed, the latter with them suppressed. Branches with novel kmers (the red vertices) clearly complete the variant, while branches lacking these novel kmers are superfluous to the traversal and can be discarded without loss. This limits the amount of unnecessary traversal in the graph, making it easier to find paths through the parental and child colors to form the variant.

4.4 Calling and classifing de novo variants

Armed with an intuition as to how graphs behave in regions of *de novo* variation, we can now describe the procedure for identifying and classifying a variant. The overview involves five big steps (and associated substeps):

1. Identify confident and trusted novel kmers
2. Construct multi-color de Bruijn "trio" graphs (child, mother, father)
3. Load subgraph local to a novel kmer
4. Identify and classify variants in the subgraph
5. Evaluate performance

We discuss these in details in the sections below.

4.4.1 Identify confident and trusted novel kmers

We first seek to build a list of novel kmers that are both *confident* (i.e. unlikely to be sequencing errors) and *trusted* (i.e. are unlikely to be the result of contamination). Identification of the novel kmers themselves is trivial; we simply build a list of kmers that appear in the child but are completely absent in the parents. The subsequent filtering steps are described below.

4.4.1.1 *Remove low coverage kmers*

For a deeply sequenced sample, we expect all kmers in the genome to be of similarly high coverage. While there will inevitably be regions of the genome where coverage is poor owing to failure to amplify regions with high GC content, the Lander-Waterman statistics in Chapter 1 suggest we should easily see many copies of the entire *P. falciparum* genome at a coverage of $100x$. We can therefore assume that failure to reach a certain coverage threshold is indicative of sequencing error, and such kmers can be removed as candidates from the novel kmer list.

The problem remains as to where the threshold should be set. For each sample, we plotted the histogram of kmer coverage, shown in Figure 4.12, smoothing the resulting distribution with a non-parametric LOESS fit. We assumed that sequencing error was fairly random and followed a geometric distribution, such that the kmers generated due to error rarely reoccurred and should consequently have low coverage. We assumed that real data was Poisson distributed. The full kmer coverage distribution is therefore the sum of both distributions. In Figure 4.12, the initial decay of kmers from 0 to $10x$ would appear to be the result of sequencing error, while the later regularly-spaced local maxima should indicate kmer coverage for regions that are copy-number 1, 2, 3, etc. Thus, we attempted to place our threshold between these two distributions.

4.4.1.2 *Remove possible contaminants*

Contamination can occur during sequencing library preparation due to handling from human operators (transferring either bacterial or human genetic material to the library) or from other samples (often from different species) being processed in the same laboratory. Contamination would result in kmers that appear novel - owing to their absense in the parents - but are irrelevant for our study. It is perhaps not a paramount concern when processing *P. falciparum* data, as the genome is very different than any other sample likely to be a contaminant. However, it may contribute false-positives that we can easily mitigate.

To remove the effects of contamination, we run every confident novel kmer through BLAST.[97] Specifically, we used the `blastn` package and all available BLAST nucleotide databases to identify the likely species of every confident novel 47-mer in each sample. Any unidentified kmer or apparently *P. falciparum* kmer was retained; all others were removed. In practice, this removes anywhere from dozens to thousands of kmers from needing to be considered; as evidenced by Figure 4.13, the exact number varies as each sample is prepared independently, at different times and by different personnel.

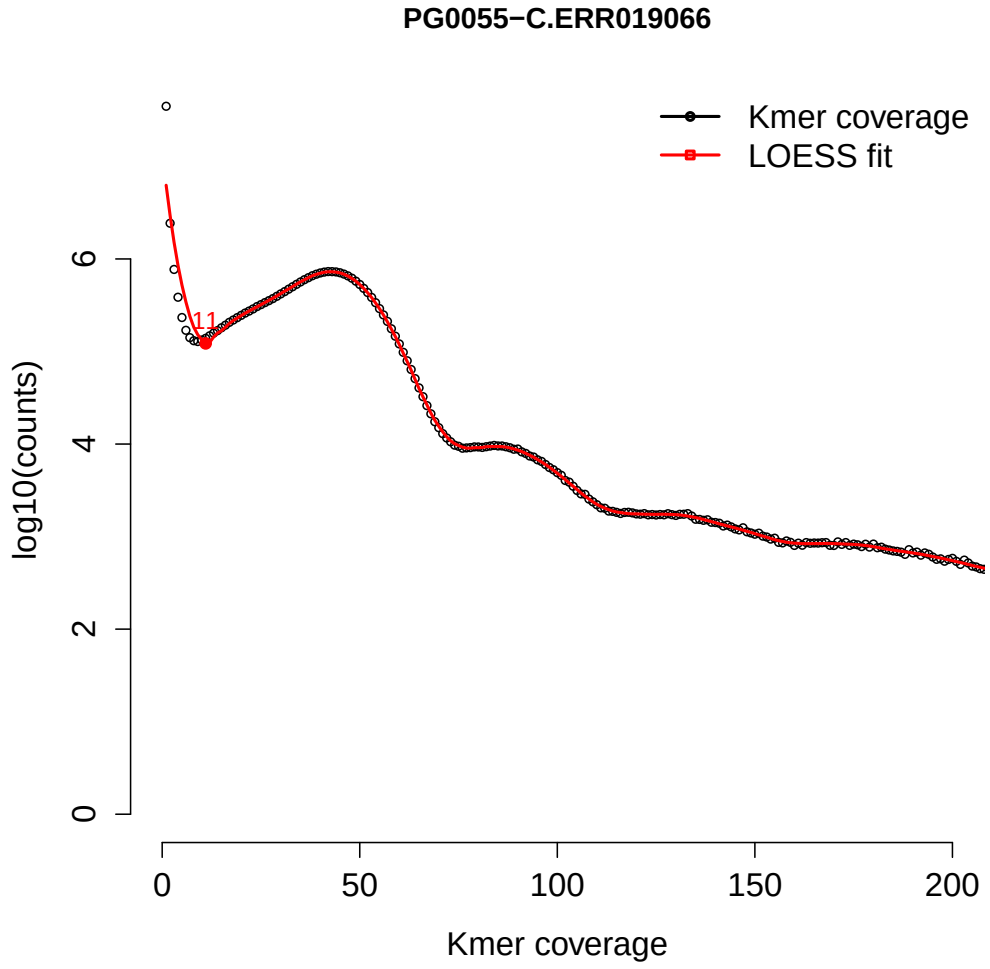


Figure 4.12: Kmer coverage histogram for a real sample, with LOESS fit

4.4.2 *Construct multi-color de Bruijn "trio" graphs*

To construct the "trio" graphs, we perform assembly on each sample with the *Cortex de novo* assembly software[33] following the recommended workflow.[93] Briefly, each sample is assembled using the `build` command at a kmer size of 47 bp[4] and ignoring nucleotides with an Illumina quality score less than 5. Each sample was then cleaned of likely sequencing errors with the `clean` command, described in greater detail below. Finally, the graph for the child was merged into a *clean* trio graph, and separately, a *dirty* trio graph (one using the uncleaned graph data) using the `join` command. This multi-color graph consists of child, mother, and father. For our purposes with *P. falciparum*, "child" is assigned color 0, "mother" (the first parent of the cross) to color 1, and "father" to color 2.

[4]This setting results from evaluations on optimal kmer size for maximizing contig length, despite not needing to produce contigs for our analyses

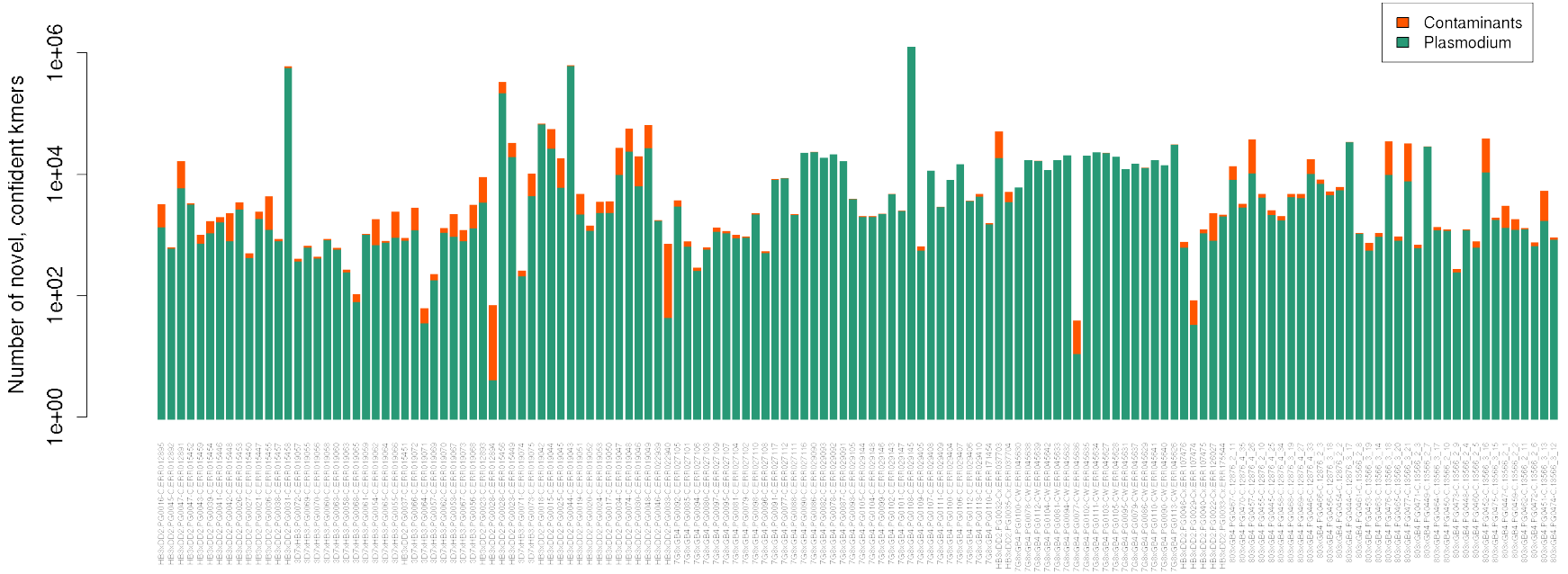


Figure 4.13: Removal of contaminating kmers; *P. falciparum* kmers are shown in green; the putative contaminants are shown in orange.

Read threading was *not* applied, as in our analyses, there is no need for contigs, just the graph data structure.

4.4.3 Graph cleaning

Sequencing error generates kmers in the dataset absent from the genome from which the reads were sampled. In runs of adjacent kmers with only one incoming and outgoing edge, or "unitigs", such sequencing errors serve to prematurely truncate the traversal by forming "junctions", kmers with more than one incoming and outgoing edge (in-degree or out-degree greater than one). Sequencing error limits our ability to produce high-quality assmblies and discover interesting biological variation in the dataset. We discuss our methods for removing such errors here.

The `clean` command considers two error modes. First, "tips", a unitig with the sum of in-degree and out-degree less than two, can form due to sequencing errors at the beginning or end of reads, or due to gaps in coverage. Tips are not useful to assembly and are thus easily discarded. By default, we remove tips shower than k nodes (the maximum number of kmers produced by a single sequencing error near the end of a read). This parameter can be overridden by the user.

Second, sequencing errors in the middle of reads (such that kmers on either side of the read provide full connectivity) can produce unitigs spanning the error and reconnecting to the graph. If sequencing errors are random, few reads will support the unitig. Thus, coverage considerations can eliminate many erroneous unitigs. We compute the coverage of each unitig as the coverage mean of each kmer in the unitig. We assume that total unitig coverage in the dataset is bimodal, with a low coverage peak (random sequencing errors) and higher coverage (true genomic data). Unitigs caused by errors are assumed to have a geometric coverage distribution, while error-free unitigs should have Poisson coverage. We fit these distributions to the empirical data, automatically choose a threshold between the two, and remove unitigs with mean coverage below this limit. This parameter can also be overridden by the user.

4.4.4 Load subgraph local to a novel kmer

To facilitate variant calling, we must process regions of the graph likely to harbor DNMs. While it is technically possible to load a *P. falciparum* graph into RAM (owing to its small 23 megabase genome), it is unlikely such a solution would scale to larger genomes. It is also cumbersome to do so when there are only on the order of dozens, perhaps hundreds, of variants to be discovered per genome. Therefore, we adopted a solution of

fetching only relevant parts of the genome as necessary, operating on the local subgraph surrounding the putative variant, rather than on the entire graph at once.

4.4.4.1 Depth-first search

Graph exploration is an *online* problem, meaning the structure of the graph cannot be known until it is explored. As a non-linear data structure, it is not trivial to determine where to start and stop exploring a graph. During traversal, one may encounter junctions (due to errors or homology) without any additional information as to which branch to choose. Graphs sometimes loop back onto themselves, necessitating that we keep track of our walk so that we do not end up traversing endlessly in circles. Incorrect decisions in the traversal cannot always be detected, requiring appropriate stopping conditions to abort a traversal if no fruitful data is discovered.

Typically, graph exploration can be accomplished with a so-called *depth-first search*, or DFS. The basic premise of a DFS is summarized in Algorithm 9: start at a vertex, walk in a chosen direction until a junction is encountered, choose one branch and repeat the DFS from that point until it is deemed appropriate to stop, then jump back to the junction to choose the next branch, and repeat until completion. In practice, this approach has some drawbacks that manifest quickly in real data. First, the basic DFS algorithm terminates only when there are no more vertices left to traverse, which is overkill for our purposes. Second, all branches are treated equally. For our purposes, branches containing errors are not important and should be discarded lest they confuse later algorithms trying to find paths through bubbles in order to type variants. Third, this requires us to explore three graphs separately, which is inefficient.

Algorithm 9 A basic, iterative depth-first search

```
1:  function IDFS(graph, vertex)
2:      s = new Stack()
3:      visited = {}
4:      s.push(vertex)
5:      while !s.isEmpty() do
6:          current = s.pop()
7:          if (visited.contains(current)) then
8:              next
9:          visited.add(current)
10:         for v in graph.nextVertices(vertex) do
11:             s.push(v)
```

The solution is to instead conduct a DFS with stopping conditions and recursion. Stopping conditions, specifying the conditions upon which a traversal is deemed "suc-

cessful" or "failed" allow us to decide certain branches in the subgraph are unfruitful for analysis. The recursive traversal allows us to act on the result of the stopping condition, adding it to the subgraph if successful, discarding it if not. This is described in Algorithm 10.

We begin this algorithm by starting at a kmer. Depending on whether we are moving forward or backwards in the graph, we pick up all next or previous kmers, adding all of them to a depth-first search subgraph. We then evaluate whether we should keep going with our traversal (the conditions for when this happens will be discussed in the next few pages) or if our history of seen kmers contains this kmer (meaning we've entered a graph loop and should abort). If the traversal should continue, we add the next kmer to the graph and continue. If there are multiple next kmers, we recursively initiate new depth-first searches for each of them.

If the traversal should not continue, we investigate why. Should the traversal have been successful (see below), we return our subgraph. If it fails, we return nothing.

Note how when a kmer has multiple incoming or outgoing edges to traverse, we initiate new depth first searches. Stopping conditions get evaluated on those branches of the graph separately. This means that if a traversal is deemed to have "failed", that branch is not returned to the greater subgraph. As we shall see in a moment, the novel kmers enable this decision making, preventing us from growing the subgraph to a scope beyond what is needed to call the variant in question.

Stopping conditions are implemented as a callback object, permitting programmer-specified limits for different situations. To type DNMs, we begin traversal at a novel kmer, walking forward and backward in the graph until the stopping conditions are met. We then explore the parental graphs based on the subgraph we loaded for the child. As the parental traversals have some additional information (namely, the presence of the child's subgraph allows us to check if a parental traversal has diverged and rejoined from the graph), the stopping conditions are different.

The stopping condition callback object implements two boolean method prototypes: `hasTraversalSucceeded` and `hasTraversalFailed`. Both of them may return `false`, in which case the traversal continues. If either returns `true`, traversal is halted and the branch is evaluated for retention or rejection.

4.4.4.2 Stopping conditions for child

The child's stopping conditions on traversal success or failure are provided in Algorithm 11 and 12, respectively. For the child, success is approximately determined by having explored a novel kmer stretch in the graph to the point that it has rejoined the parental

Algorithm 10 The recursive depth-first search with arbitrary stopping conditions

```
1: function RDFS(g, kmer, color, g, stopper, depth, goForward, history)
2:     firstKmer = kmer
3:     dfs = new AnnotatedGraph()
4:     repeat
5:         cv = kmer
6:         cr = g.find(cv)
7:         prevKmers = getPrevKmers(clean, dirty, cv)
8:         nextKmers = getNextKmers(clean, dirty, cv)
9:         adjKmers = goForward ? nextKmers : prevKmers
10:        addVertexAndConnect(dfs, cv, prevKmers, nextKmers)
11:        if (stopper.keepGoing(cr, g, depth, dfs.vertexSet().size(), adjK-
   mers.get(color).size()) && !history.contains(kmer)) then
12:            history.add(kmer)
13:            if (adjKmers.get(color).size() == 1) then
14:                kmer = next(adjKmers.get(color))
15:            else if (adjKmers.get(color).size() != 1) then
16:                childrenWereSuccessful = false
17:                for (ak in adjKmers.get(color)) do
18:                    branch = dfs(clean, dirty, ak, color, g, stopperClass, depth + isNovel-
   Kmer(cr) ? 0 : 1, goForward, history)
19:                    if (branch != null) then
20:                        addGraph(dfs, branch)
21:                        childrenWereSuccessful = true
22:                if (childrenWereSuccessful || stopper.hasTraversalSucceeded(cr, g,
   depth, dfs.vertexSet().size(), 0)) then
23:                    return dfs
24:        else if (stopper.traversalSucceeded()) then
25:            return dfs
26:        else
27:            return null
28:    until (adjKmers.get(color).size() == 1)
29:    return null
```

graphs. However, we purposefully continue reading another 50 kmers before returning success. If more novel kmers are recovered in that span, we reset our counters and continue walking. This facilitates two things: the absence of novel kmers due to sequencing errors or overthresholding, and typing complex variants that may have short stretches of non-novel kmers. Failure is determined by the following criteria: absence of novel kmers, having no more branches to explore, having reached a maximum graph depth of 5, or having reached a maximum graph size of 5,000.

Algorithm 11 Child's traversal success determination method

```
1: function hasTraversalSucceeded(cr, g, depth, size, edges)
2:     if (goalSize == 0 && (cr.getCoverage(1) > 0 || cr.getCoverage(2) > 0)) then
3:         goalSize = size
4:         goalDepth = depth
5:     if (goalSize > 0 && isNovel(cr)) then
6:         goalSize = size
7:         goalDepth = depth
8:     return (goalSize > 0 && (size >= goalSize + 50 || edges == 0))
```

Algorithm 12 Child's traversal failure determination method

```
1: function hasTraversalFailed(cr, g, depth, size, edges)
2:     return !isNovel(cr) && (edges == 0 || depth >= 5 || size > 5000)
```

4.4.4.3 *Stopping conditions for parents*

The parents' stopping conditions on traversal success or failure are provided in Algorithm 13 and 14, respectively. Success is determined by having diverged from the child's graph and rejoined it. Care is taken to ensure that we have rejoined the graph at a boundary of the novel kmer stretch, rather than some ther part of the graph obtained when trying to read some extra flanking data in Algorithm 11. Failure is determined by the following criteria: having reached a maximum graph size of 5,000, or having reached a maximum graph depth of 5.

4.4.5 *Identify and classify variants in the subgraph*

With the relevant subgraph now loaded, we can now attempt to type variants. Typing involves three steps:

1. Annotate vertices in the graph that can be possible start and end points of the variant bubble

Algorithm 13 Parents' traversal success determination method

```
1: function hasTraversalSucceeded(cr, g, depth, size, edges)
2:     fw = cr.getKmerAsString()
3:     rc = reverseComplement(fw)
4:     v = null
5:     rejoinedGraph = false
6:     if (g.containsVertex(fw) || g.containsVertex(rc)) then
7:         rejoinedGraph = true
8:         for av in g.vertexSet() do
9:             if (av.getKmer().equals(fw) || av.getKmer().equals(rc)) then
10:                if (!sawPredecessorFirst && !sawSuccessorFirst) then
11:                    if (av.flagIsSet("predecessor")) then sawPredecessorFirst = true
12:                    else if (av.flagIsSet("successor")) then sawSuccessorFirst = true
13:                if ((sawPredecessorFirst && av.flagIsSet("predecessor")) || (sawSuccessorFirst && av.flagIsSet("successor"))) then
14:                    rejoinedGraph = false
15:                    break
16:    return size > 1 && rejoinedGraph
```

Algorithm 14 Parents' traversal failure determination method

```
1: function hasTraversalFailed(cr, g, depth, size, edges)
2:     return size > 5000 || junctions >= 5
```

2. Find the shortest path from start to end that satisfies some conditions

3. From the haplotypes, remove the flanking homologous sequence to reveal the variant alleles

Let us first detail how we annotate the viable starts and ends of the variant, which simply amounts to iterating through each vertex in the subgraph and checking that it means two conditions. First, a variant start (end) vertex should have out degree (in degree) greater than 1. Second, the branches should be color-specific to reflect the opening of a bubble between the different samples in the graph. This is summarized in Algorithm 15.

4.4.5.1 Dijkstra's shortest path algorithm

With all of the start and end points now annotated, we now need to find a path from one end of the putative variant to the other. The number of possible paths could be very large. We shall make the assumption that the correct path is the shortest path. While this will often be the case, we note that the shortest path is not necessarily the biological path.

Algorithm 15 Annotating possible variant starts and ends of the subgraph

```
1: function annotateStartsAndEnds(b)
2:     for av in b.vertexSet() do
3:         if (b.outDegreeOf(av) > 1) then
4:             aes = b.outgoingEdgesOf(av)
5:             childEdges = {}
6:             parentEdges = {}
7:             for (AnnotatedEdge ae in aes) do
8:                 if (ae.isPresent(0) && ae.isAbsent(1) && ae.isAbsent(2)) then
9:                     childEdges.add(ae)
10:                if (ae.isPresent(color)) then
11:                    parentEdges.add(ae)
12:            if (childEdges.size() > 0 && parentEdges.size() > 0) then
13:                av.setFlag("start")
14:                candidateStarts.add(av)
15:        if (b.inDegreeOf(av) > 1) then
16:            aes = b.incomingEdgesOf(av)
17:            childEdges = {}
18:            parentEdges = {}
19:            for AnnotatedEdge ae in aes do
20:                if ae.isPresent(0) && ae.isAbsent(1) && ae.isAbsent(2) then
21:                    childEdges.add(ae)
22:                if ae.isPresent(color) then
23:                    parentEdges.add(ae)
24:            if (childEdges.size() > 0 && parentEdges.size() > 0) then
25:                av.setFlag("end")
26:                candidateEnds.add(av)
```

This could be the case in highly repetitive regions longer than a kmer length. Since de Bruijn graphs tend to collapse long repeats, we may miss some events.

E. W. Dijkstra described his shortest path algorithm in 1959.[98] We describe it below in Algorithm 16. Briefly, we begin at the source vertex. We record all vertices' distance from this source vertex, setting the source's distance to itself to 0 and all others to infinity (to denote as-of-yet unvisited vertices). We then select the vertex with the minimum distance to the source (in the first iteration, the source itself) and compute a new distance to all immediately adjacent vertices. This new distance is a sum of the distance traversed so far from the source to one of these vertices. For our purposes, we shall assume the distance between two adjacent vertices is always 1. If the distance computed is less than the distance recorded, we replace the old value with the new. We remove this vertex from the processing queue and proceed to the next vertex with the lowest distance from the source.

Algorithm 16 Finding the shortest path in a graph

```
1:  function DSPA(graph, source, destination, color)
2:      dist = {}
3:      prev = {}
4:      q = []
5:      for all v in g.vertexSet() do
6:          dist[v] = infinity
7:          prev[v] = null
8:          push(q, v)
9:      dist[source] = 1
10:     while !q.isEmpty() do
11:         u = vertexWithMinDistance(q, dist)
12:         if u == destination then
13:             break
14:         q.remove(u);
15:         if u != -1 then
16:             for all e in g.outgoingEdgesOf(u, color) do
17:                 v = g.getEdgeTarget(e, color)
18:                 alt = dist[u] + 1
19:                 if alt < dist[v] then
20:                     dist[v] = alt
21:                     prev[v] = u
22:     s = []
23:     Integer u = destination
24:     while u != null && prev.containsKey(u) do
25:         push(s, u)
26:         u = prev[u]
27:     return s
```

To use this algorithm for allele identification, we iterate through all possible combinations of start and stop vertices, obtained as described in the previous section. We then iterate through the three graph colors (representing the child, mother, and father). For each color, we apply Algorithm 16. For the child, we place the additional constraint that the path accepted must contain at least one novel kmer.

This algorithm will return the child and parental haplotypes. To identify the precise alleles of the event, we simply trim back the homologous regions of the haplotypes with Algorithm 17.

Algorithm 17 Trim back haplotypes to reveal alleles

```
1:  function trimHaplotypes(child, parent)
2:      e0 = child.length() - 1
3:      e1 = parent.length() - 1
4:      s = 0
5:      length = (child.length() < parent.length() ? child.length() : parent.length())
6:      for (s = 0; s < length && child[s] == parent[s]; s++) do
7:          (do nothing)
8:      while (e0 > s && e1 > s && child[e0] == parent[e1]) do
9:          e0–
10:         e1–
```

4.4.5.2 *Classify event*

For simple variants (those that conform to the bubble motif), event classification is reasonably straightforward. We simply inspect the recovered alleles and evaluate them against simple rules. For example, a SNP should be a single nucleotide in length. An insertion should have a parental allele length of 1 and a child allele length > 1.

To classify complex variants, we rely on other heuristics. GC events are classified by keeping track of which parent the child's genome appears to be exclusively copying from (simply by measuring when kmer coverage has dropped from one parent and returned in the other), detailed in Algorithm 18. NAHR events, involving recombinations between different chromosomes, are detected by examining if any kmers are uniquely found on different chromosomes, detailed in Algorithm 19.

The classification algorithm can be described by the decision tree in Figure 4.14.

4.4.5.3 *Choosing haplotypic background*

When calling DNMs, we do not know the haplotypic background upon which a variant has occured. We must compare the child's graph to the mother's and then to the father's, making separate calls. This results in two variant calls for the same event. A single

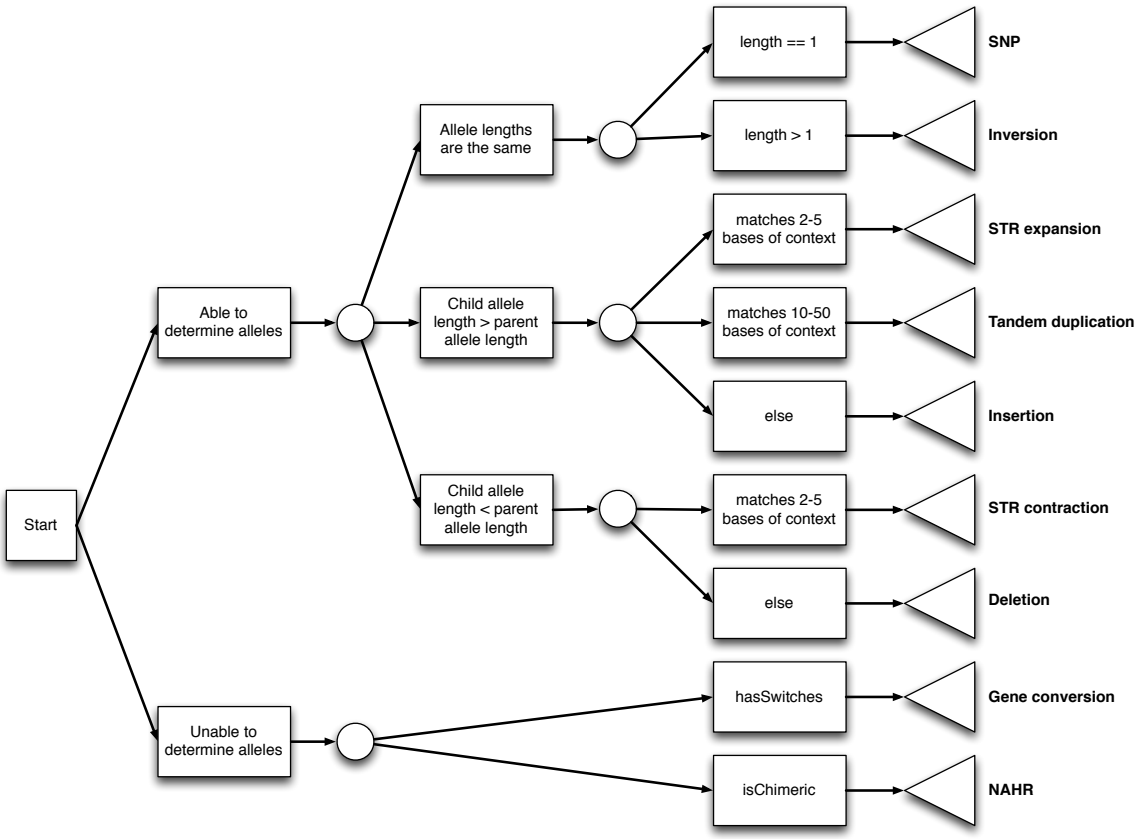


Figure 4.14: Decision tree for *de novo* variant classification.

Algorithm 18 Stretch has switches

```
1: function hasSwitches(graph, stretch)
2:     inherit = []
3:     for all 47-bp kmers k in stretch do
4:         cov0 = graph.getCoverage(k, 0)                    ▷ child
5:         cov1 = graph.getCoverage(k, 1)                    ▷ mother
6:         cov2 = graph.getCoverage(k, 2)                    ▷ father
7:         if (cov0 > 0 && cov1 == 0 && cov2 == 0) then
8:             append(inherit, "C")
9:         else if (cov0 > 0 && cov1 > 0 && cov2 == 0) then
10:            append(inherit, "M")
11:        else if (cov0 > 0 && cov1 == 0 && cov2 > 0) then
12:            append(inherit, "D")
13:        else if (cov0 > 0 && cov1 > 0 && cov2 > 0) then
14:            append(inherit, "B")
15:    for i in inherit.length() do
16:        for inherit[i] == 'B' do
17:            prevContext = '?'
18:            nextContext = '?'
19:            for (j = i - 1; j >= 0; j–) do
20:                if (inherit[j] == 'M' || inherit[j] == 'D') then
21:                    prevContext = inherit[j]
22:                    break
23:            for (j = i + 1; j < inherit.length(); j++) do
24:                if (inherit[j] == 'M' || inherit[j] == 'D') then
25:                    nextContext = inherit[j];
26:                    break;
27:            context = '?';
28:            if (prevContext == nextContext && prevContext != '?') then
29:                context = prevContext
30:            else if (prevContext != nextContext) then
31:                if (prevContext != '?') then
32:                    context = prevContext
33:                else
34:                    context = nextContext
35:            inherit[i] = context
36:    return inheritStr.matches(".*D+.C+.M+.*") || inheritStr.matches(".*M+.C+.D+.*")
```

Algorithm 19 Stretch has chimeras

```
1: function hasChimeras(stretch, lookup)
2:     chrCount = {}
3:     for all kmers k in stretch do
4:         if lookup.numAlignments(k) == 1 then
5:             chrCount[lookup.getFirstAlignment(k).getChr()]++
6:     return chrCount.size() > 1
```

event must be chosen. In cases where the child and parental alleles are identical, this is trivial. Additionally, variants where the event can be identified against one parent but not the other are straightforward. The problematic events are those that have conflicting descriptions between parents. In this situation, we heuristically assign a score for various properties of the variant, choosing the representation with the highest score. This has the effect of preferring simpler descriptions (e.g. "SNP") to more complex ones (e.g. "GC").

Algorithm 20 Score variant

```
 1: function chooseVariant(gvc)
 2:     scores = []
 3:     for all colors c do
 4:         if (gvc.traversalIsComplete(c)) then
 5:             scores[c]++
 6:         if (gvc.variantType(c) != "unknown" then
 7:             scores[c]++
 8:         if (gvc.variantType(c) not in ("GC", "NAHR")) then
 9:             scores[c]++
10:     bestColor = whichMax(scores)
11:     return (scores[0] == scores[1] ? 0 : bestColor)
```

4.4.5.4 Mark traversed novel kmers as used

We then mark all the novel kmers we saw in the shortest path (the path that contained the variant allele) as used so we do not process them again. This allows us to keep track of our progress, only acting on kmers that have not been associated to any variant yet. It also prevents us from accidentally processing the same variant several times (once for every novel kmer induced by the mutation).

4.4.6 Evaluate performance

We evaluated the performance of our graphical DNM caller by running it on some simulated data. Note that the way a variant is simulated and the way a variant is called are not necessarily identical. In particular, indels are tricky because there is not a standard representation in graphs. Usually, when indels are called in reference-based methods, the reference is taken to be the forward strand and indels are left-shifted as far as possible. In graphs, there is no information available for determining orientation. When they were simulated, we knew what the forward strand was, but we do not know that in the graph. Thus, when we go to check on the presence of a variant, we must check it in both orientations, and accounting for multiple representations.

4.4.6.1 *Pre-compute novel kmer to variant map*

We iterated through all variants, extracting sequence from the simulated child's reference genome between start $-(2k-1)$ and stop $+(2k-1)$ bp. We then extracted each kmer from this sequence and, if the kmer is novel, emitting a table row mapping the novel kmer to variant ID, variant type, and position in the child's reference genome. Note that because we chose to space our variants out over considerable distance, it is generally not possible for a kmer to map to multiple variants. However, as we have placed very few constraints on variant positioning otherwise, it is possible a simulated variant has landed in a low-complexity region that occurs multiple times throughout the genome, and that multiple variants thus share novel kmers. In that instance, the first variant we see during processing is assigned the novel kmer.

4.4.6.2 *Load variant containing a novel kmer and comparing*

If we are in evaluation mode (that is, if we've a novel kmer to variant map along with a dataset), then after each variant we call, we check if any of the kmers involved in the variant call are in the map, and load the relevant variant information. We then evaluate our call using Algorithm 21.

4.5 *Results on simulated data*

We ran our algorithm on the perfect and realistic simulated datasets presented in Chapter 3. Looking first at purely event recovery without classification (i.e. did we find *de novo* variants, irrespective of whether we classified them correctly), we summarize our stats in Tables 4.2 and 4.3. In both cases, our sensitivity and specificity to DNMs is very high - greater than 98% in nearly all cases. Note that the true negatives (tn) are effectively every kmer that we did *not* call in our dataset, which makes our specificity nearly perfect, as the use of the novel kmers prevents us from making calls in the vast majority of the graph.

Additionally in these tables, we also computed the "redundant call", or "rc" metric. The rc metric reveals an important caveat regarding graph calls: should the traversals fail and the alleles of the variant not ascertained, the novel kmers involved in the variant will not be marked as used. However, we still emit a variant "event", despite our inability to classify it. In the subsequent iteration of the algorithm, the remaining novel kmers will be picked up for calling again, leading to the same variant being emitted multiple times. Thus, we must exercise caution with untyped variants: the rate at which they appear in our callset may be slightly higher (up to 10%) than the rate at which they truly exist in the genome.

Algorithm 21 Evaluate variant

```
1: function evalVariant(call, truth, stretch)
2:     relevantVariants = []
3:     for all kmers in stretch do
4:         if (truth.contains(kmer)) then
5:             push(relevantVariants, truth.get(kmer))
6:     bestVariant = null;
7:     for all vi in relevantVariants do
8:         ref = call.getParentalAllele()
9:         alt = call.getChildAllele()
10:        refStretch = call.getParentalStretch()
11:        altStretch = call.getChildStretch()
12:        found = false;
13:        if (!found) then                                  ▷ left-shift variants and check
14:            pos = call.getStart()
15:            while (pos >= 0 && pos + ref.length() < refStretch.length() && pos + alt.length() < altStretch.length()) do
16:                refFw = refStretch.substring(pos, pos + ref.length())
17:                altFw = altStretch.substring(pos, pos + altLength)
18:                refRc = reverseComplement(refFw);
19:                altRc = reverseComplement(altFw)
20:                if (known ref and alt alleles match called fw or rc alleles) then
21:                    ref = (refFw or refRc)
22:                    alt = (altFw or altRc)
23:                    found = true;
24:                    break;
25:                pos−
26:        if (!found) then                                  ▷ right-shift variants and check
27:            pos = call.getStart()
28:            while (pos >= 0 && pos + ref.length() < refStretch.length() && pos + alt.length() < altStretch.length()) do
29:                refFw = refStretch.substring(pos, pos + refLength)
30:                refRc = reverseComplement(refFw)
31:                altFw = altStretch.substring(pos, pos + altLength)
32:                altRc = reverseComplement(altFw)
33:                if (known ref and alt alleles match called fw or rc alleles) then
34:                    ref = (refFw or refRc)
35:                    alt = (altFw or altRc)
36:                    found = true;
37:                    break;
38:                pos++
            ▷ Maybe what we've found is a truncated version of what we were expecting
39:        knownRef = vi.ref
40:        knownAlt = vi.alt == null ? "" : vi.alt
```

```
41:         if (!found && knownRef.length() > ref.length() && knownAlt.length() > alt.length() && knownRef.length() == knownAlt.length()) then
42:             refFw = ref;
43:             altFw = alt;
44:             refRc = reverseComplement(refFw);
45:             altRc = reverseComplement(altFw);
46:             refFinal = null, altFinal = null;
47:             if (knownRef contains refFw or refRc and knownAlt contains altFw or altRc) then
48:                 refFinal = (refFw or refRc)
49:                 altFinal = (altFw or altRc)
50:             if (refFinal != null && altFinal != null) then
51:                 roindex = knownRef.indexOf(refFinal)
52:                 aoindex = knownAlt.indexOf(altFinal)
53:                 r1index = roindex + refFinal.length()
54:                 a1index = aoindex + altFinal.length()
55:                 if (roindex == aoindex && r1index == a1index && knownRef.substring(0, roindex).equals(knownAlt.substring(0, aoindex)) && knownRef.substring(r1index, knownRef.length()).equals(knownAlt.substring(a1index, knownAlt.length()))) then knownRef = ref; knownAlt = alt;
56:         vi.found = true;
57:         vi.matches = (knownRef.equals(ref) && knownAlt.equals(alt)) || vi.type.equals(call.getType())
58:         if (bestVi == null || vi.matches) then
59:             bestVi = vi;
60:         if (bestVi != null) then
61:             knownRef = bestVi.ref;
62:             knownAlt = bestVi.alt != null ? bestVi.alt : "";
```

Table 4.2: ROC metrics on simulated perfect data

sn	sim	fp	fn	tp	tn	rc	sens	spec	prec	npv	fpr	fnr	fdr	acc
5	perfect	1	5	126	22240910	9	0.9618	1	0.9921	1	0	0.0382	0.0079	1
3	perfect	2	3	154	22200660	4	0.9809	1	0.9872	1	0	0.0191	0.0128	1
0	perfect	0	2	106	22241343	7	0.9815	1	1.0000	1	0	0.0185	0.0000	1
12	perfect	0	2	113	22233659	3	0.9826	1	1.0000	1	0	0.0174	0.0000	1
13	perfect	0	2	114	22217297	5	0.9828	1	1.0000	1	0	0.0172	0.0000	1
17	perfect	0	2	117	22242890	6	0.9832	1	1.0000	1	0	0.0168	0.0000	1
1	perfect	1	2	137	22249556	8	0.9856	1	0.9928	1	0	0.0144	0.0072	1
14	perfect	2	2	162	22196410	7	0.9878	1	0.9878	1	0	0.0122	0.0122	1
11	perfect	1	1	114	22207761	2	0.9913	1	0.9913	1	0	0.0087	0.0087	1
6	perfect	1	1	118	22203683	2	0.9916	1	0.9916	1	0	0.0084	0.0084	1
8	perfect	1	1	129	22229379	5	0.9923	1	0.9923	1	0	0.0077	0.0077	1
10	perfect	0	1	130	22238480	2	0.9924	1	1.0000	1	0	0.0076	0.0000	1
15	perfect	1	1	131	22208991	10	0.9924	1	0.9924	1	0	0.0076	0.0076	1
16	perfect	1	1	166	22213928	4	0.9940	1	0.9940	1	0	0.0060	0.0060	1
2	perfect	1	0	105	22225262	3	1.0000	1	0.9906	1	0	0.0000	0.0094	1
4	perfect	2	0	72	22210167	2	1.0000	1	0.9730	1	0	0.0000	0.0270	1
7	perfect	1	0	57	22225612	1	1.0000	1	0.9828	1	0	0.0000	0.0172	1
9	perfect	0	0	107	22255460	2	1.0000	1	1.0000	1	0	0.0000	0.0000	1
18	perfect	2	0	133	22242522	0	1.0000	1	0.9852	1	0	0.0000	0.0148	1
19	perfect	1	0	98	22236313	6	1.0000	1	0.9899	1	0	0.0000	0.0101	1

Table 4.3: ROC metrics on simulated realistic data

sn	sim	fp	fn	tp	tn	rc	sens	spec	prec	npv	fpr	fnr	fdr	acc
5	realistic	4	2	123	77566161	7	0.9840	1	0.9685	1	0	0.0160	0.0315	1
0	realistic	1	1	101	77516712	7	0.9902	1	0.9902	1	0	0.0098	0.0098	1
6	realistic	2	1	107	77289511	2	0.9907	1	0.9817	1	0	0.0093	0.0183	1
13	realistic	0	1	107	77362132	3	0.9907	1	1.0000	1	0	0.0093	0.0000	1
17	realistic	3	1	109	77390562	5	0.9909	1	0.9732	1	0	0.0091	0.0268	1
12	realistic	2	1	111	77440516	3	0.9911	1	0.9823	1	0	0.0089	0.0177	1
16	realistic	1	1	155	77369055	5	0.9936	1	0.9936	1	0	0.0064	0.0064	1
2	realistic	4	0	93	77425427	1	1.0000	1	0.9588	1	0	0.0000	0.0412	1
4	realistic	3	0	65	77312705	2	1.0000	1	0.9559	1	0	0.0000	0.0441	1
3	realistic	6	0	150	77296380	1	1.0000	1	0.9615	1	0	0.0000	0.0385	1
1	realistic	1	0	133	77509177	5	1.0000	1	0.9925	1	0	0.0000	0.0075	1
7	realistic	4	0	56	77445270	1	1.0000	1	0.9333	1	0	0.0000	0.0667	1
8	realistic	6	0	126	77469034	3	1.0000	1	0.9545	1	0	0.0000	0.0455	1
9	realistic	2	0	95	77503142	2	1.0000	1	0.9794	1	0	0.0000	0.0206	1
10	realistic	1	0	124	77398963	2	1.0000	1	0.9920	1	0	0.0000	0.0080	1
11	realistic	2	0	108	77422535	2	1.0000	1	0.9818	1	0	0.0000	0.0182	1
14	realistic	6	0	150	77355056	7	1.0000	1	0.9615	1	0	0.0000	0.0385	1
15	realistic	1	0	125	77435743	9	1.0000	1	0.9921	1	0	0.0000	0.0079	1
18	realistic	4	0	123	77355144	3	1.0000	1	0.9685	1	0	0.0000	0.0315	1
19	realistic	2	0	95	77480228	6	1.0000	1	0.9794	1	0	0.0000	0.0206	1

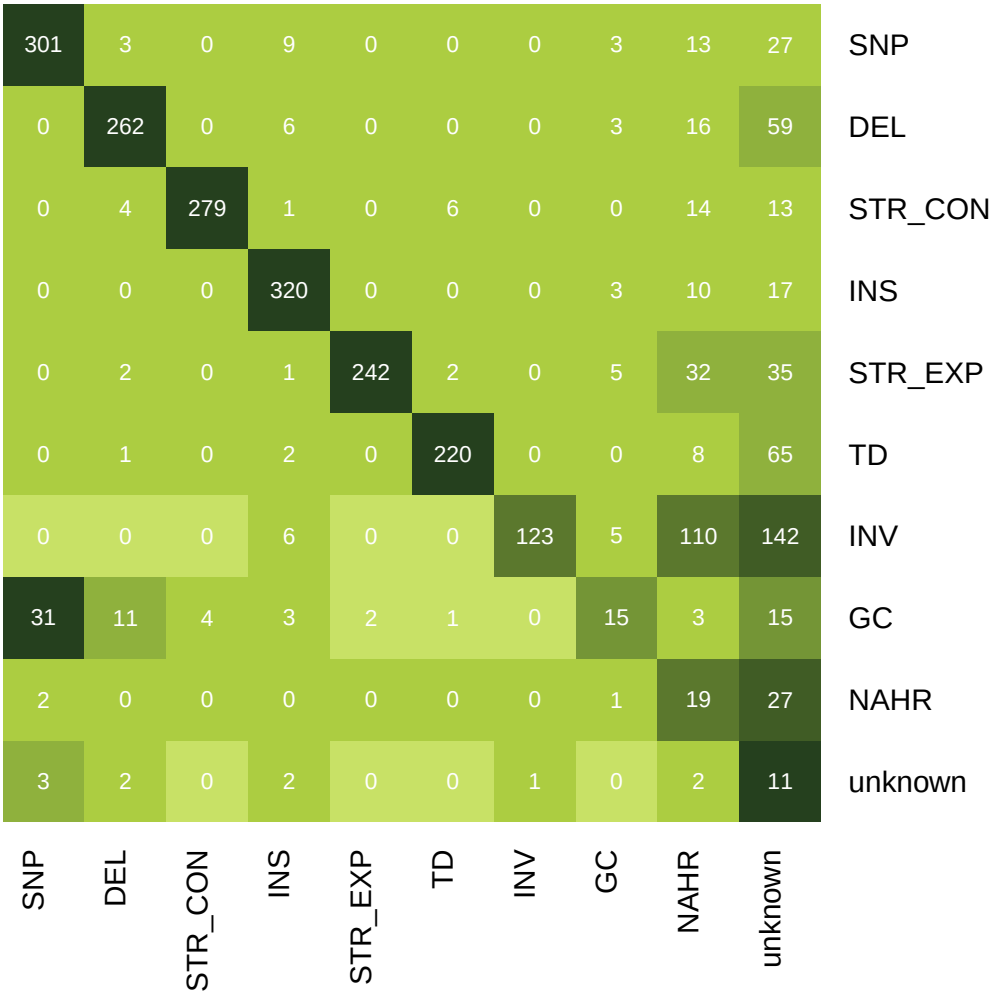


Figure 4.15: Confusion matrix for observed events (below) versus expected events (right), in simulated perfect data.

Results on event classification are shown as heatmaps in Figures 4.15 and 4.16, with observed events on the bottom and expected events on the side. On the observed axis, an "unknown" event denotes a variant that could not be typed. On the expected axis, an "unknown" event is one that does not appear in the simulation list (i.e. a false positive). The total number of events simulated that could possibly be recovered via novel kmers is the sum of each row (except for the "unknown" row, which instead enumerates the false positives and the types they've been assigned). For the most part, variants appear on the diagonal, indicating proper classification. Performance degrades when we move into the non-bubble motif variants (i.e. GC and NAHR) events.

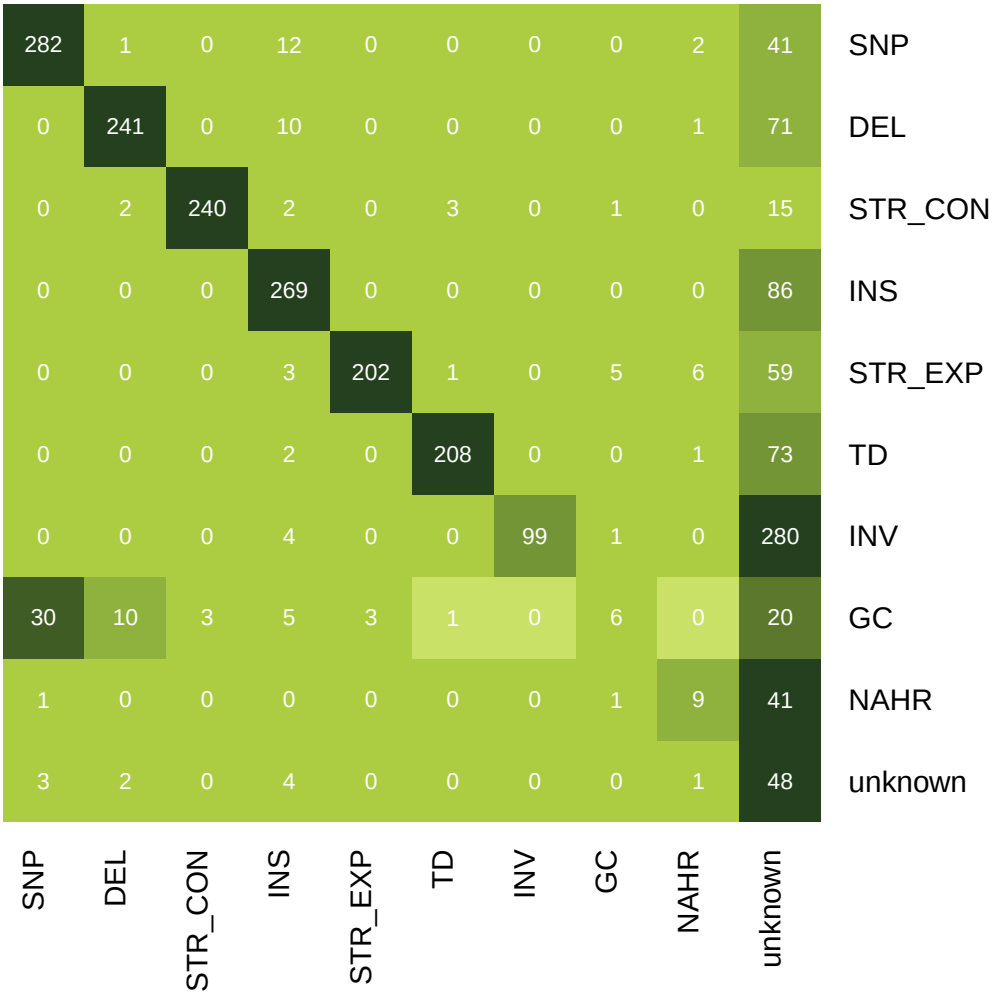


Figure 4.16: Confusion matrix for observed events (below) versus expected events (right), in simulated realistic data.

Gene conversion events are quite often erroneously described as SNPs. This is likely due to the fact that the classification algorithm is looking for clear switches of parental copying in the child (copying from mother, then father, then mother again, or vice versa). However, if variants that appear on different haplotypic backgrounds are too close to one another, 47-bp kmers might span both, and the recombination motif that we seek may be obscured and cause the algorithm to call a SNP instead.

Many NAHR events are classified as "unknown". This is perhaps not unreasonable. Our classification algorithm for NAHR events (Algorithm 19) requires that the stretch have kmers from different chromosomes. However, if the stretch truncates early due to

errors or homology, there may be insufficient genomic context in order to determine the chromosomes of origin, leaving us unable to classify the variant accordingly.

4.6 Summary

We have presented our algorithms for identifying and typing *de novo* variation in cross and pedigree data. At the heart of our approach is a graphical model based on a de Bruijn graph representation of sequencing data from parents and children. We exploit the novel kmer generation property of DNMs to discover regions of the graph worth exploring for DNM activity, being careful to discard kmers seemingly arising from error. This approach appears to work well in simulation, typically recovering bubble-motif events well and exhibiting predicted difficulty with more complex, linear-motif events (those involving some form of recombination).

5 Validation

5.1 Introduction

Real data tends to carry more surprises than even the most sophisticated simulation software can reveal. While the simulation framework we presented in Chapter 3 was useful to establish the basic operations and performance of the graphical variant calling algorithm presented in Chapter 4, true validation data is useful in establishing filters, traversal parameters, and other parameters for treating real data. In this chapter, we discuss validation data obtained for this purpose and the insights gleaned by comparing an initial set of calls to this dataset.

5.1.1 Data processing for validation isolates

We focused our validation efforts on the 803xGB4 cross, solely because samples were already in culture, reducing turnaround time from a full year to a few months. We selected four samples for long-read sequencing on the PacBio RS II instrument. As of this writing, only two samples have been completed: PG0051-C (the 3D7 clone, useful for verifying assembly quality by comparison with the finished 3D7 reference), and PG0446-C (one of the 803xGB4 progeny); it is currently not certain if and when the other two samples will be reattempted. The PG0051-C isolate was provided by Susana Campino of the Kwiatkowski lab at the Wellcome Trust Sanger Institute. The PG0446-C isolate was provided by Michael Krause and Rick Fairhurst at the Malaria Pathogenesis and Human Immunity Unit at the National Institute of Allergy and Infectious Disease. In both cases, approximately 15 μg of high molecular weight gDNA for each isolate was vacuum-sealed and shipped to the CSHL Pacific Biosciences Sequencing Service[1]. Sequencing libraries with an average fragment size of 20 kb were constructed for each sample and size-selected to remove fragments smaller than 7 kb and larger than 20 kb using BluePippin. Optimal loading concentration was determined by choosing a wide range for each of the first 4 SMRT cells and selecting the concentration that yielded the highest read yield for the

[1] http://cshl.edu/Research/PacBio.html

Table 5.1: Assembly statistics for PacBio RS II data on PG0051-C and PG0446-C isolates

	PG0051-C (3D7)	PG0446-C (36F11)
Chemistry	P5-C3	P6-C4
SMRT cells	8	9
Number of reads	245,661	282,459
N50 read length	12,968 bp	14,339 bp
Assembler	HGAP 2.0	HGAP 3.0
Average contig coverage	78x	94x
Polished contigs	34	33

remaining SMRT cells. The PG0051-C isolate was sequenced in late 2014 with the P5-C3 chemistry and assembled with PacBio's HGAP 2.0 software (assuming a genome size of 23 megabases). The PG0446-C isolate, run 16 months later, was sequenced with the newer P6-C4 chemistry and HGAP 3.0 software with the same genome size parameter. Table 5.1 summarizes the sequencing and assembly results on these two isolates.

5.1.1.1 Establishing assembly quality

We examined genome recovery and access to massively repetitive sequence (subtelomeric and centromeric regions) that tend to be inaccessible with Illumina sequencing. PacBio reads were aligned to the finished 3D7 genome using PacBio's `blasr` long read alignment tool[2], while Illumina reads were mapped with the `bwa mem` tool. Two such regions in the Illumina and PacBio data for the PG0051-C isolate are shown in Figure 5.1. It is evident that the PacBio coverage is roughly uniform across the entire length of the chromosome. In contrast, the Illumina coverage spikes and dips as it moves along, reaching zero coverage in many regions (especially the biologically interesting subtelomeric repetitive regions).

We compared the PacBio-produced assembly of the PG0051-C isolate to the finished reference sequence by performing an all-by-all (contigs versus chromosomes) alignment with MUMmer.[99] The alignments are visualized as a multi-dotplot in Figure 5.2, an extension of a dot plot that depicts alignments as two dimensional matricies with target and query sequences on the x and y axes, aligning regions of the two sequences shaded accordingly.[100] In this plot, each aligned section of a contig is represented by a line segment terminated by circles. For this reason, more segments than contigs are apparent. Red (blue) segments indicate alignments in forward (reverse) orientation. Most segments adhere closely to the diagonal, indicating substantial full-length alignment between the reference and the draft assembly. Breaks in the diagonal resembling jumps in the x- or

[2] `https://github.com/PacificBiosciences/blasr`

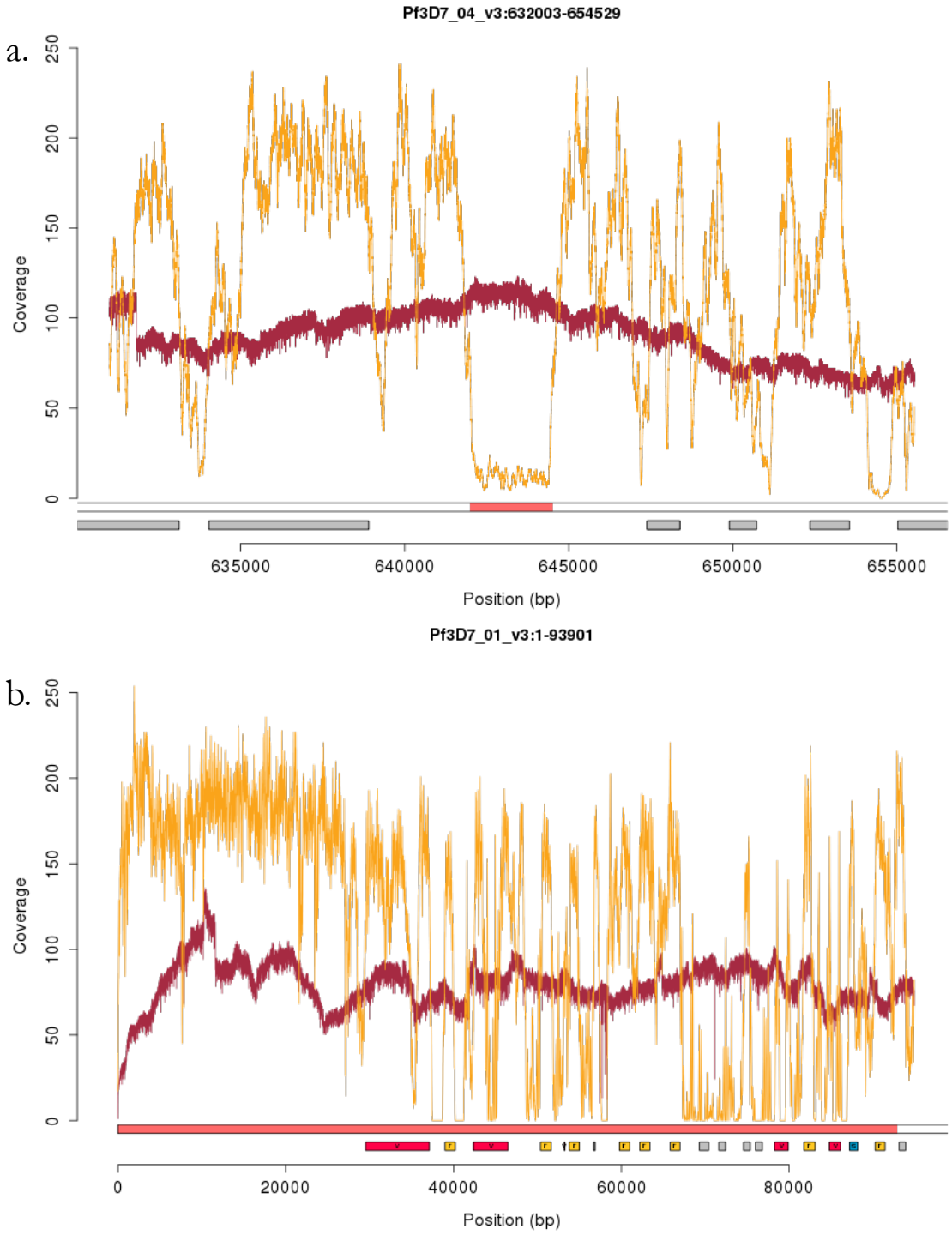


Figure 5.1: Coverage for Illumina data (orange) and PacBio data (red) of the same sample: PG0051-C (the reference isolate, 3D7). a. Coverage over the chromosome 4 centromere. b. Coverage over the 5′ subtelomere of chromosome 1. Genes from the *var*, *rifin*, and *stevor* antigenic gene families are highlighted as red, orange, and blue rectangles, respectively.

y-axis indicate insertions or deletions in the assemblies. Most chromosomes here are assembled completely, and the overwhelming majority of the assembly appears on-diagonal (indicating successful one-to-one reconstruction). Elements appearing off-diagonal could represent misassembly. However, note that most of these off-diagonal elements occur towards the extremes of each chromosome. Given that the reference genome was constructed with Sanger reads substantially shorter than the PacBio reads, it is possible some repetitive regions have been collapsed or misplaced, contributing to this nominal error rate.

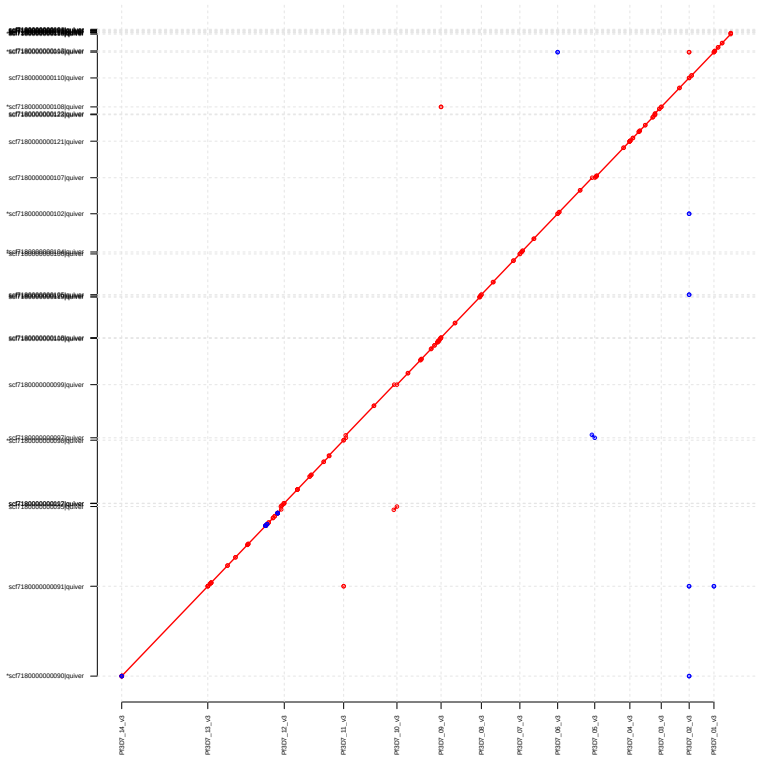


Figure 5.2: Alignment of contigs from PG0051-C to 3D7 reference assembly, with sequences from reference and draft assembly on the x and y-axis, respectively. Each aligned section of a contig from the draft assembly is represented by a line segment terminated at either end with circles. Red denotes forward alignment of contigs to the reference, blue denotes reverse alignment.

As we have sequenced DNA from the 3D7 parasite, any differences should likely reflect errors in the sequence. We therefore called SNPs between the two assemblies to find these errors. The sums are presented in Table 5.2, as well as the percent of bases per chromosome these errors represent. Overall, the SNP, insertion, and deletion rates are exceedingly low: amounting to 19,580 events in a 23 megabase genome (0.17%). The insertion rate is much higher than that of deletions and SNPs, perhaps due to the domi-

Table 5.2: Apparent errors per chromosome in the PG0051-C assembly

	SNP	INS	DEL
Pf3D7_01_v3	119 (0.02%)	763 (0.12%)	291 (0.05%)
Pf3D7_02_v3	164 (0.02%)	475 (0.05%)	162 (0.02%)
Pf3D7_03_v3	272 (0.03%)	523 (0.05%)	290 (0.03%)
Pf3D7_04_v3	141 (0.01%)	856 (0.07%)	713 (0.06%)
Pf3D7_05_v3	111 (0.01%)	531 (0.04%)	175 (0.01%)
Pf3D7_06_v3	504 (0.04%)	731 (0.05%)	180 (0.01%)
Pf3D7_07_v3	194 (0.01%)	609 (0.04%)	320 (0.02%)
Pf3D7_08_v3	134 (0.01%)	597 (0.04%)	251 (0.02%)
Pf3D7_09_v3	61 (0.00%)	713 (0.05%)	259 (0.02%)
Pf3D7_10_v3	732 (0.04%)	810 (0.05%)	308 (0.02%)
Pf3D7_11_v3	310 (0.02%)	1,008 (0.05%)	459 (0.02%)
Pf3D7_12_v3	232 (0.01%)	1,202 (0.05%)	390 (0.02%)
Pf3D7_13_v3	231 (0.01%)	1,493 (0.05%)	409 (0.01%)
Pf3D7_14_v3	152 (0.00%)	1,309 (0.04%)	396 (0.01%)
Total	3,357 (0.03%)	11,620 (0.10%)	4,603 (0.04%)

nant insertion error mode of the PacBio sequencing instrument. All chromosomes appear reasonably similar in performance.

We examined the recovery of the 62 members of the *var* gene family by aligning their full-length genomic sequences (exons and introns) to the PG0051-C assembly using `bwa mem`. All 62 *var* genes were successfully aligned to the assembly (all had mapping quality greater than 0; only 1 had mapping quality less than 60). 21 were found to map with 100% identity. The remaining have, on average, 2.46 mismatches, 1.39 insertions, and 0.63 deletions. The overwhelming majority of indels are a single nucleotide in length.

It seemed likely that many of these errors occur in intronic regions where high repetitive sequence content might contribute to misassembly. We investigated this hypothesis by aligning the exons of the *var* genes separately and enumerating errors observed in exons and introns. We ignored 11 genes with poor exon alignments (i.e. with mapping quality less than 10). 91.78% of the errors are found in intronic regions. In all cases, exon 2 of the *var* gene (the short exon) is base-for-base perfect when compared to the canonical reference.

Based on these measurements of the error rate, we estimate the quality of the PacBio assembly of the PG0051-C (3D7) isolate to be approximately $Q31$[3], or less than one error per thousand bases. We note that this is a pessimistic estimate, based on the assumption that any differences between this and the reference assembly indicate errors in our assembly. In long, repetitive regions of the genome, this assumption may not be accurate.

[3] $Q = -10\log_{10}(q) = -10\log_{10}((11,620 + 4,603 + 3,357)/23,332,831)$

5.1.1.2 *Preparing the validation isolate reference*

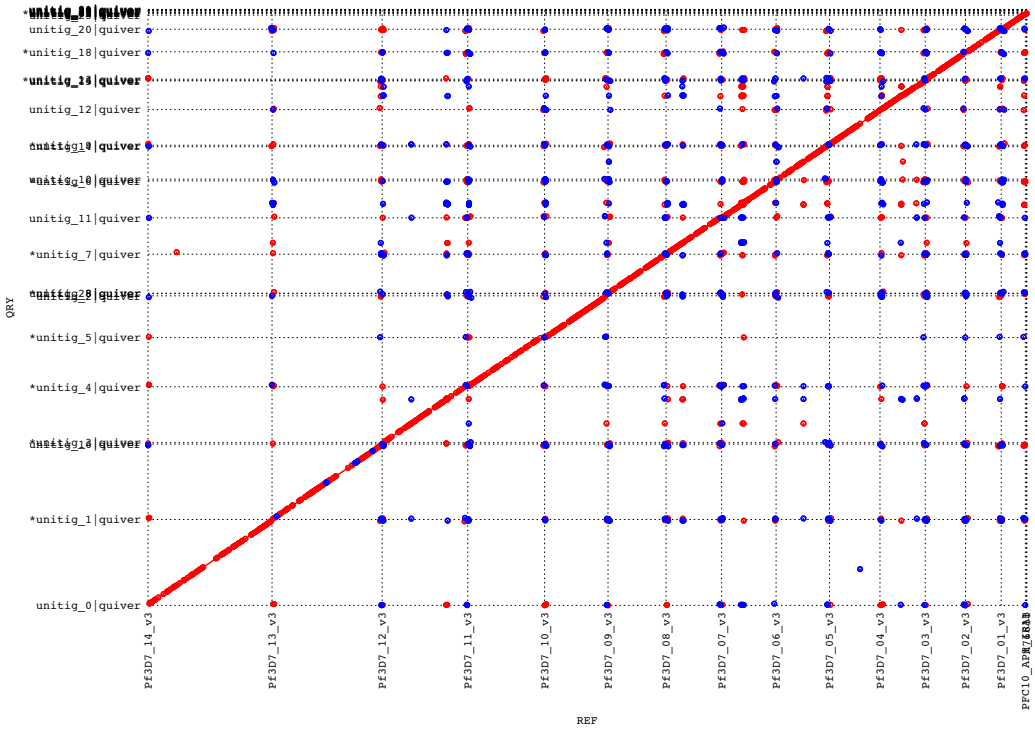


Figure 5.3: Alignment of contigs from PG0446-C to 3D7 reference assembly, with sequences from reference and draft assembly on the x and y-axis, respectively. Each aligned section of a contig from the draft assembly is represented by a line segment terminated at either end with circles. Red denotes forward alignment of contigs to the reference, blue denotes reverse alignment.

Red points/lines indicate an alignment in forward orientation, and blue points/lines an alignment in reverse orientation (with respect to the reference sequence).

The dominant feature of the alignment is the broken red line from bottom-left to top-right. This indicates a substantial full-length alignment between the two genomes. Breaks in the diagonal that look like 'jumps' in the X- or Y-axis represent insertions/deletions in either the query or reference sequence Diagonal runs of alignments that are off the main diagonal indicate repeated similar sequence, or sequence rearrangements. It is usual with mummer output to see many individual alignments as 'noise' off the main diagonal.

Having established the performance of the instrument and assembler in providing a viable assembly for a *P. falciparum* isolate, we turned our attention to the validation sample from the 803xGB4 cross, PG0446-C. We aligned the isolate's assembly to the 3D7 reference

sequence using MUMmer, shown in Figure 5.3. Note that compared to the PG0051-C isolate, the validation isolate has much more off-diagonal activity, particularly towards the telomeres. This is to be expected. While the core genome between isolates is expected to be relatively stable, tremendous immune pressure has forced the antigenic repertoire to diversify rapidly. These genes, primarily located in the subtelomeric regions, are the regions that land off-diagonal, indicative of the extensive recombination and mutation history.

We relabelled and reoriented each contig as necessary based on the alignment so as to more easily establish genomic positioning of events during analysis. The assignments and orientation were further validated by transferring 3D7 gene model annotations onto the PG0446-C assembly by aligning each exon with `bwa mem`, operating under the assumption that the core genome is reasonably stable between isolates and that properly labelled and oriented contigs will yield exon alignments that match orientation and approximate positioning between the two assemblies. Figure 5.4 shows an example for chromosome 1. The vertical offset of all points is due to the fact that the assembly length of chromosome 1 in the PG0446-C assembly is longer than the reference length. Only exons towards the telomeric ends of the chromosome are misplaced, evidently originating from chromosomes other than chromosome 1. All other chromosome 1 exons are aligned with the expected position and orientation.

After chromosome identification, we observed a handful of contigs that could not be assigned a place in the nuclear genome. After running each unplaced contig through BLAST, we discovered two long contigs (thousands of bp) and nearly perfect hits to various species in the *Pseudomonas* genus. Discovering this contamination in the PacBio assembly and the Illumina samples strongly suggests the samples are contaminated at the source, contributing to the inflated 803xGB4 assembly lengths. We removed these contigs from our assembly.

We verified sample identity by slicing the PacBio assembly into 1,000 bp tiles (non-overlapping), aligning each tile to the 3D7 reference genome using `bwa mem`, and genotyping sites found to be variant in the MalariaGen 803xGB4 callset using the Genome Analysis Toolkit module, `UnifiedGenotyper` (specifying the genotype likelihood model to "SNP", ploidy to 1, genotyping mode to "GENOTYPE_GIVEN_ALLELES", assuming default base quality scores should be $Q30$, and providing the 803xGB4 alleles from MalariaGen)[4]. Chromosome 14 is shown in Figure 5.5. Note that while the PacBio sample's

[4]This procedure, while admittedly a bit indirect, provides vastly better results than genotyping the PacBio reads directly. The assembly contains far fewer errors than the reads, even after error correction. It is also much more straightforward than processing the MUMmer variant output, which uses a different file format to VCF, making comparisons cumbersome.

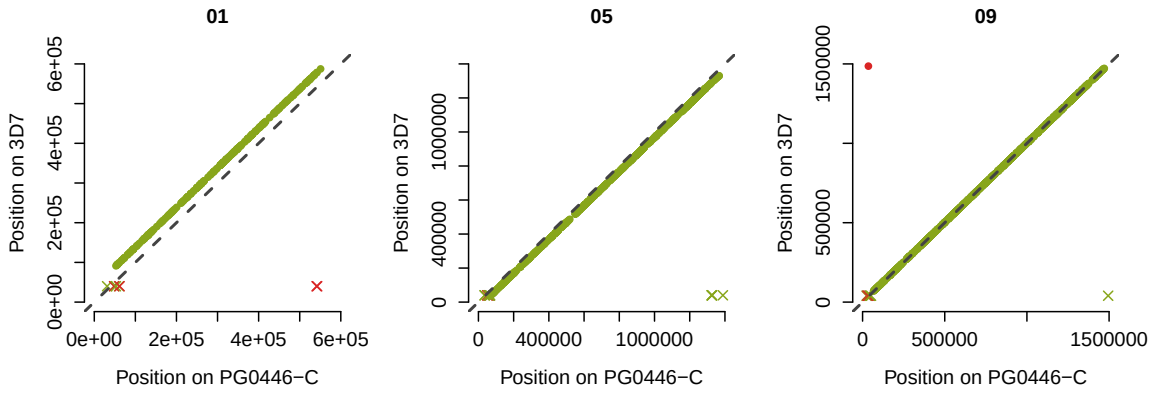


Figure 5.4: Placement and orientation of exons from the reference assembly mapped to the PG0446-C assembly. Chromosomes 1, 5, and 9 shown. Each point represents an exon aligned to both the reference and PG0446-C assembly. Green indicates matching orientation of the exon's alignment in both assemblies. Red indicates a forward alignment in one, reverse-complement alignment in the other. Crosses indicate exons from a different chromosome than the one indicated in the plot title.

crossover pattern does appear to match that of PG0446-C, samples PG0445-C, PG0453-C, and PG0457-C also share the same pattern. This common pattern is observed for these samples on all chromosomes. These samples are almost certainly clones of one another.

5.2 *Comparison to validation isolate*

5.2.1 *Verification of novel kmers*

As we have demonstrated that the PacBio assembly of the validation isolate is high-quality, novelty found in the Illumina dataset for the same sample can be verified by examining the PacBio isolate. At the filtration levels of "all", "confident", and "trusted", initial processing of the Illumina data for 803xGB4 sample PG0446-C was found to have 28,904; 17,069; and 8,197 novel kmers, respectively. The PacBio assembly of the same sample recapitulates 1,639; 802; and 48 novel kmers, respectively. The discrepancies can be largely attributed to three (not necessarily independent) factors: the lower threshold for coverage set too high in certain circumstances, no upper threshold on coverage, and significant data contamination.

The first two points regarding coverage can be observed in Figure 5.6a-c. The left-most panel shows a clear valley between the left end of the distribution (presumably sequencing errors, as they are so rarely observed in the dataset) and the first local maximum (likely real data). In the middle panel, kmer coverage in PG0446-C decays more

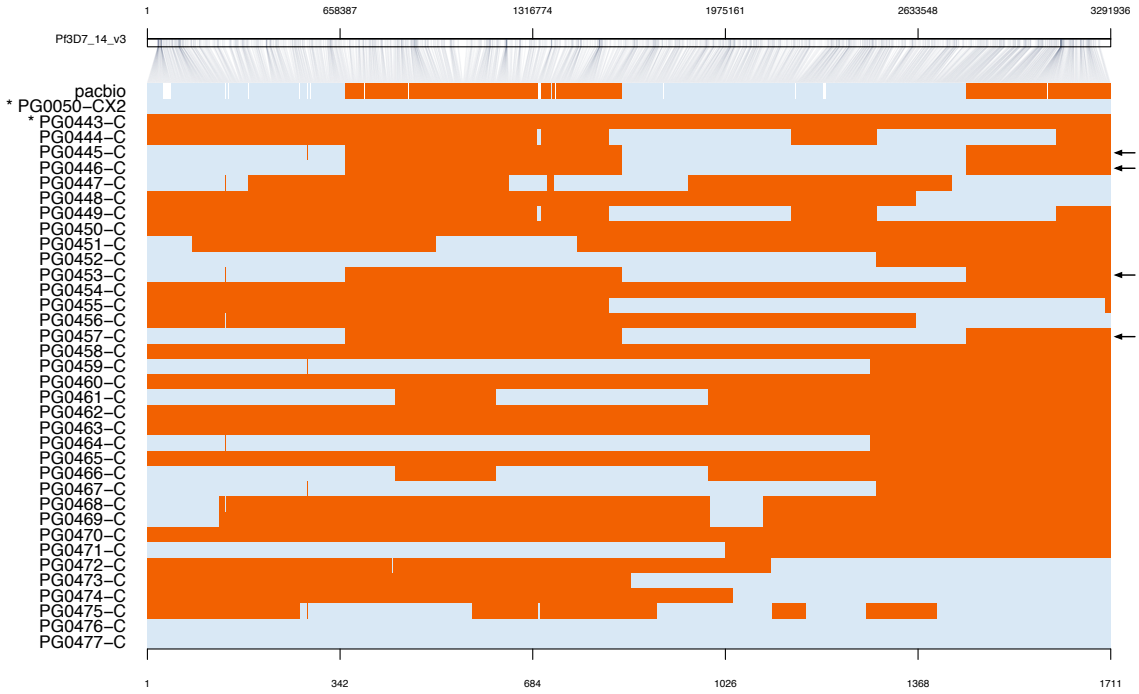


Figure 5.5: Haplotype mosaic for chromosome 14 in the 803xGB4 cross dataset. Parental samples are identified with asterisks. For remaining samples, alleles are color-coded according to the parent from which they are apparently inherited - orange for "mother" (PG0443-C, the 803 isolate), blue for "father" (PG0050-CX2, the GB4 isolate). Missing data (sites where no genotype could be ascertained) are shown in white. Bottom axis represents variant number. Top axis represents variant position along the length of the chromosome. Arrow on right-hand side indicate samples that are apparently clonal copies of one another.

smoothly, and it is far less obvious as to where the lower threshold should be placed[5]. In the rightmost panel, all novel kmers are shown, conditional to being present or absent in the PacBio assembly of the same sample. Our automatically calculated threshold is clearly set too high, as it is slicing through the middle of the coverage distribution for novel kmers recapitulated by the PacBio dataset.

Furthermore, the rightmost panel shows a number of coverage outliers in the set of novel kmers that are not present in the PacBio dataset. While their source is not immediately apparent, it is clear they have a large contribution to the set of novel kmers we examine; $1,057$ kmers present in the Illumina dataset but absent in the PacBio dataset were found to be high coverage outliers ($> 100x$).

We investigated other potentially erroneous novel kmers: putative novel kmers ab-

[5]It is worth mentioning that the PG0446-C sample, sequenced on more modern Illumina HiSeq 2000 platforms, likely enjoys a much lower sequencing error rate than the Illumina GAII used for PG0055-C, which contributes to our difficulty in automatically finding a reasonable coverage threshold.

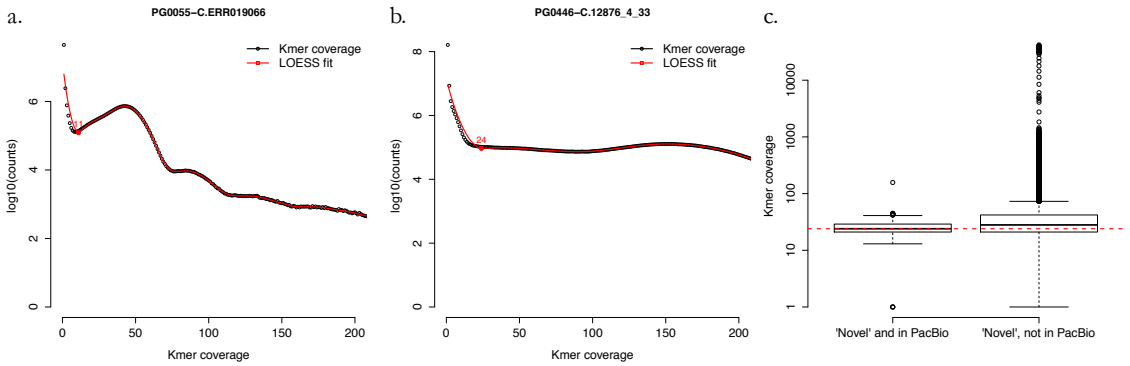


Figure 5.6: a. Kmer coverage in a 3D7xHB3 sample, with LOESS curve and calculated lower threshold at curve's first local minimum shown in red. b. Kmer coverage in the 803xGB4 sample, PG0446-C, with LOESS curve and calculated lower threshold at curve's first local minimum shown in red. c. Kmer coverage distributions for all novel kmers in PG0446-C, conditional on the kmer being present or absent in the PacBio assembly. Calculated lower threshold from panel b shown as red dotted line.

sent from the PacBio assembly and with coverage between $10x$ and $74x$. We discovered many of these novel kmers exist in branches that either never connect with the graphs of the parents (we refer to these as "orphan" branches, Figure 5.7a), or connect suspiciously amidst a number of putatively novel kmers rejected by the contamination checks (we refer to these as "bushy-tailed" branches, Figure 5.7b). Both appear to be symptoms of unrecognized contamination, owing to the fact that the BLAST database is not (and can never be) complete. In attempting to remove any contribution to the graphs from non-target sources, it is insufficient to discard only the kmers that appear in the BLAST database. Sequencing centers will always be sequencing new organisms, and the contents of the BLAST database will necessarily lag behind. Discarding orphan branches is straightforward, but incompletely treats the problem; the bushy-tailed branches erroneously connect with the parental graphs and must be removed as well.

Finally, we discovered a substantial source of erroneous novel kmers stemming from "overcleaning", an issue wherein some kmers representing true genomic sequence are mistaken for sequencing error and removed from the graph via the assembler's error-cleaning or error-correcting process. When this occurs in the parents but not the child (often due to coverage fluctuations), a kmer can appear to be "novel". To mitigate this issue, we inspect the child's cleaned graph and the parents' dirty graphs. However, a region of low coverage can also be flanked by regions of *no* coverage; there are many kmers that are likely present in the parental genomes, but unobserved by chance.

To resolve these issues, we implemented a filtering solution for novel kmers in four parts. First, we implemented a manual procedure for specifying coverage limits, select-

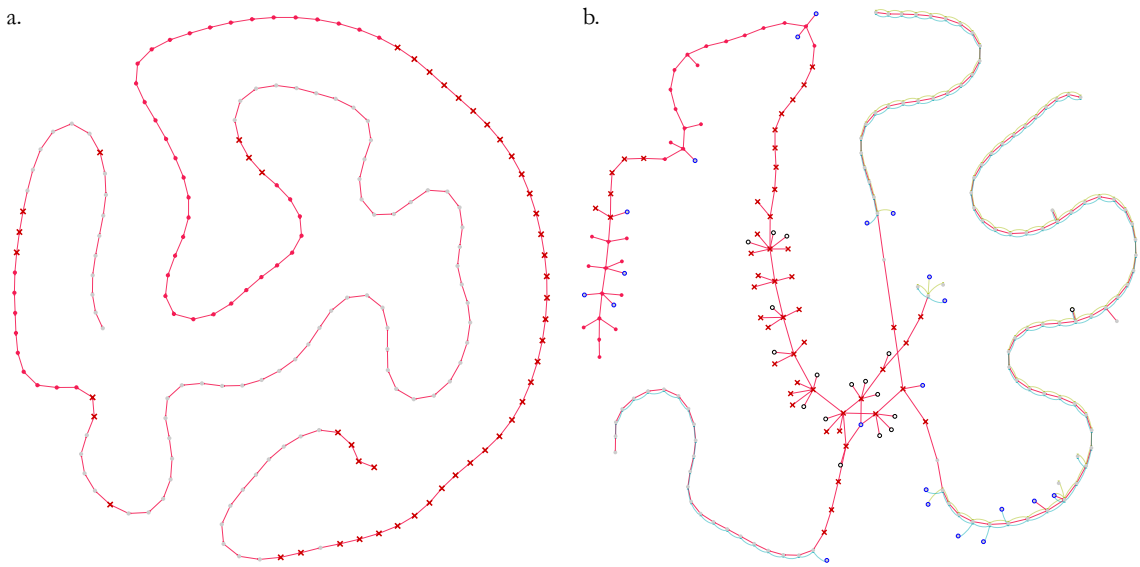


Figure 5.7: Local subgraphs in the Illumina data for PG0446-C near novel kmers that are absent in the corresponding PacBio assembly. Red circles denote "trusted" novel kmers (those that passed our initial coverage and contamination checks). Red crosses indicate "untrusted" novel kmers (those that passed our initial coverage check but failed the contamination check). a. An "orphan" branch: a series of novel kmers not connected to the parental graphs. b. A "bushy-tailed" branch: a series of novel kmers that appear to connect to everything, including the parental graphs, by chance.

ing a lower limit of $10x$ and an upper limit of $74x$. These limits were chosen by direct inspection of the kmer coverage distributions to determine a lower threshold that was early enough in the distribution that it would likely capture only the part of the distribution containing sequencing errors, and an upper threshold that was late enough in the distribution that it would not cut off the Poisson-distributed true genomic data. Next, we implemented software to explore the trio graph starting at confident novel kmers rejected by the contamination filter using our condition-limited depth first search software. Explored branches containing novel kmers that are on the same branch as rejected kmers are similarly tagged for rejection. Third, we explore the trio graph starting with remaining confident novel kmers. If the explored subgraph does not connect with the parental graphs in any way, the kmers in the branch are marked for rejection and removed from further consideration. Finally, we examine contigs with "overcleaned" novel kmers (i.e. any kmer that would be considered novel by comparing to the cleaned parental graphs, but is subsequently denied this label after examining the dirty parental graphs). Novel kmers found on the same contig are considered tainted and removed from consideration. Any putatively novel kmer surviving this battery of tests is considered "filtered".

The results of our novel kmer filtering are shown in Figure 5.8. 95 kmers survive

our battery of filters to be considered truly novel sequence in the genome of the PG0446-C child. We called DNMs in the Illumina data using our graph-based calling software and discovered three events: two SNPs (47 kmers each) and a single kmer occurring in a repetitive region of a telomere; the precise nature of this single-kmer event is not clear. The PacBio dataset contains 51 novel kmers, 48 of which overlap our calls (one SNP and the unknown event). The other 47 kmers from the second SNP are not present in the PacBio assembly. However, manual inspection reveals this SNP to be polymorphic, possibly indicating a *de novo* mutation that has occurred in the sample during mitotic divisions that has not yet reached fixation in the sequenced population of cells. It is likely that the event (shown in Figure 5.10) is a true variant, absent in the PacBio assembly as the Celera assembler is forced to make a choice between which allele to retain in the haploid assembly. The presence of the alternate allele in the uncorrected PacBio read strongly suggests this is event is real and its absence from the assembly is an artifact of the assembler's bubble-popping procedure. The remaining 3 kmers from the PacBio dataset are not called in the Illumina data as they are low coverage (all at $1x$ coverage, all low complexity sequence, possibly representing recurrent sequencing errors in both the PacBio and Illumina data).

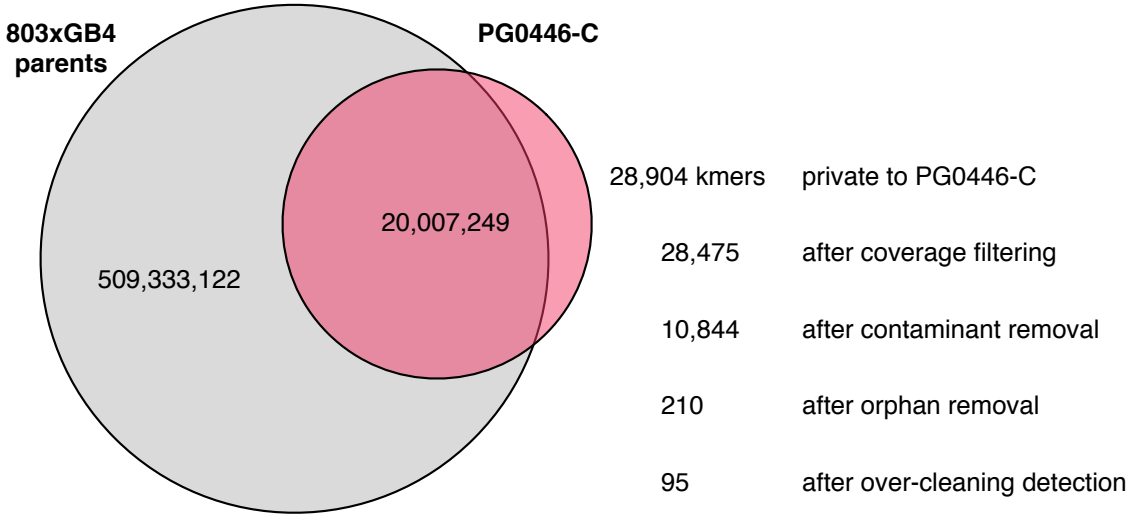


Figure 5.8: The results of filtering kmers private to the PG0446-C child

Based on the kmer recovery results listed above, our sensitivity to novel kmers is between 94% and 100% (depending on whether the 3 missed kmers are considered truly novel kmers or sequencing errors), and specificity is greater than 99%. Both SNPs and the unknown third event are recovered successfully. The two SNPs are shown in Figures 5.9 (A to G, $13 : 2,032,381$) and 5.10 (C to T, $14 : 1,396,035$, a non-synonymous substitution

resulting in a T to I amino acid change in *PF3D7_1433900*, a protein kinase). Figure 5.11 shows the third, unknown event.

5.2.2 *Comparison to reference-based analysis*

We sought to determine our relative ability to detect DNMs accurately compared to that of the reference-based analysis. As the release version of the MalariaGen callset purposefully excluded *de novo* mutations (attempting to minimize Mendelian error rates to ensure high confidence in the inherited variation callset), we set about processing the Illumina data from the PG0446-C trio (PG0443-C (803), PG0050-CX2 (GB4), and PG0446-C) ourselves. Reads were aligned with `bwa mem` to the 3D7 reference genome, release 9.0 from PlasmoDB. PCR duplicates were marked with the `MarkDuplicates` program from the Picard suite; these marked reads were subsequently ignored in downstream processing. A list of genomic regions with reads appearing to span indels were discovered using the GATK's `RealignerTargetCreator` module (supplemented by MalariaGen's 803xGB4 calls as additional candidate regions to consider). We subsequently ran the GATK's `IndelRealigner` to perform the local realignments on these candidate regions. We recalibrated base quality scores using the GATK's `BaseRecalibrator` tool, once again supplying the MalariaGen calls as sites to ignore when computing the data error model. Variants were called across the PG0443-C, PG0050-CX2, and PG0446-C samples simultaneously using the GATK's `HaplotypeCaller` module, which performs local *de novo* assembly to determine the haplotype sequence at so-called "active sites", windows of the genome detected to harbor some sort of variation. For each approach, we specified a ploidy of 1. SNPs and indels were filtered separately using the GATK's `VariantRecalibrator` tool, supplying the MalariaGen calls as training data. We set the desired truth sensitivity to 99% (i.e. requesting that the recalibration tool learn filter thresholds such that at least 99% of the MalariaGen calls were recapitulated). Finally, we isolated putative DNMs by selecting sites confidently called in all three samples, where the genotype of the child differed from that of mother and father, requiring confident genotypes in all three samples (i.e. $GQ > 90$), and sufficient allele depth in all three samples (i.e. $DP > 20$).

Callset metrics are presented in Table 5.3. Note that while MalariaGen chose to include calls only in the so-called "core" region of the genome (comprising 20.7 Mbp of sequence), we have included the accessory genome in our callset as well (the remaining 2.5 Mbp of genomic sequence spanning telomeric, subtelomeric, pericentromeric, and other hypervariable loci). For a fairer comparison, we have partitioned the calls into the core and accessory regions. Furthermore, in addition to the monomorphic sites one expects to see in a haploid genome for a *de novo* variant that occurs during meiosis or in one

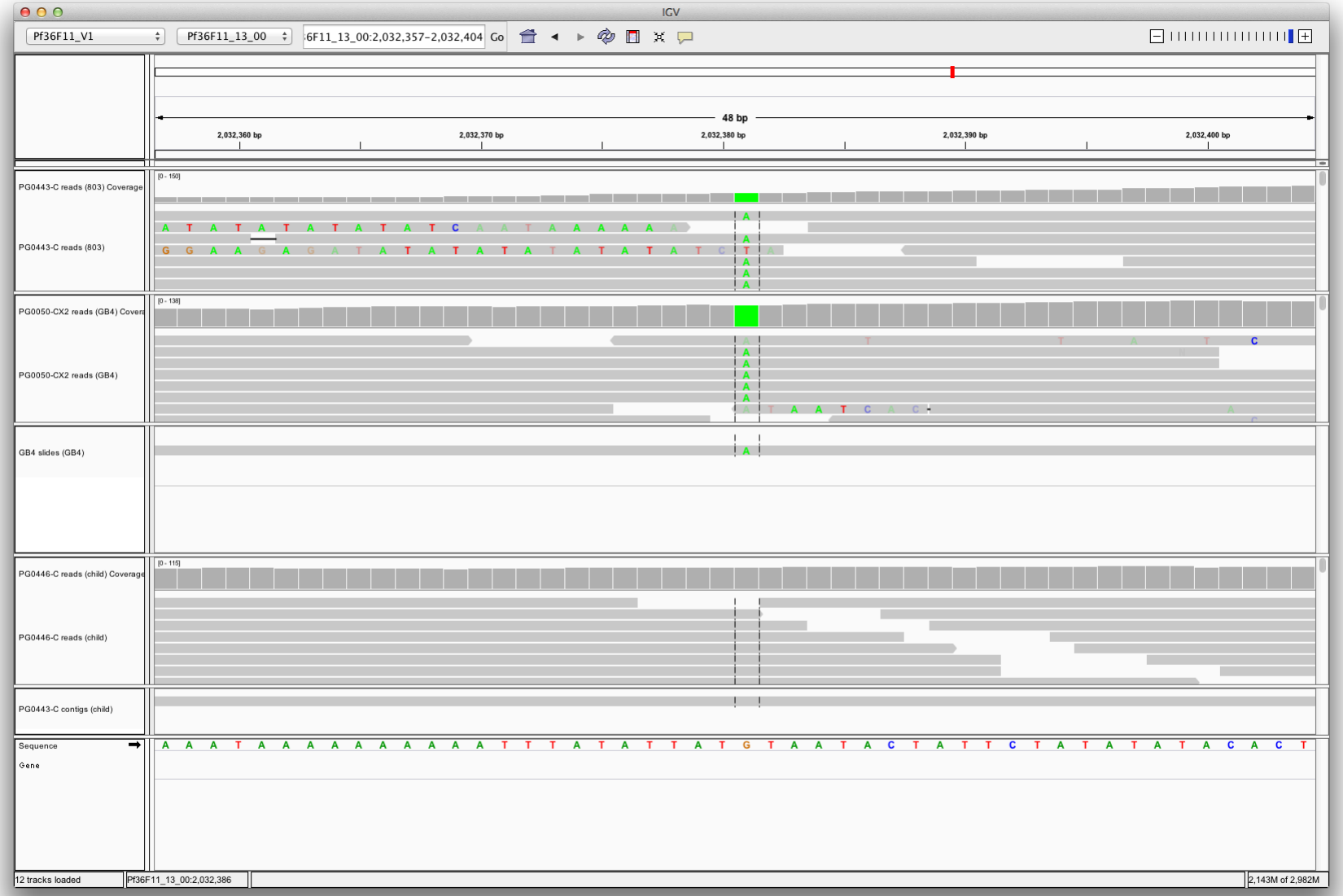


Figure 5.9: IGV screenshot of an A to G SNP, showing reads and assembly information for the trio, aligned to the PacBio PG0446-C assembly. Top panel: 803. Middle two: GB4 reads and PacBio assembly (sliced to facilitate alignment). Lower two: PG0446-C reads and contig containing the variant in question from the a. panel.

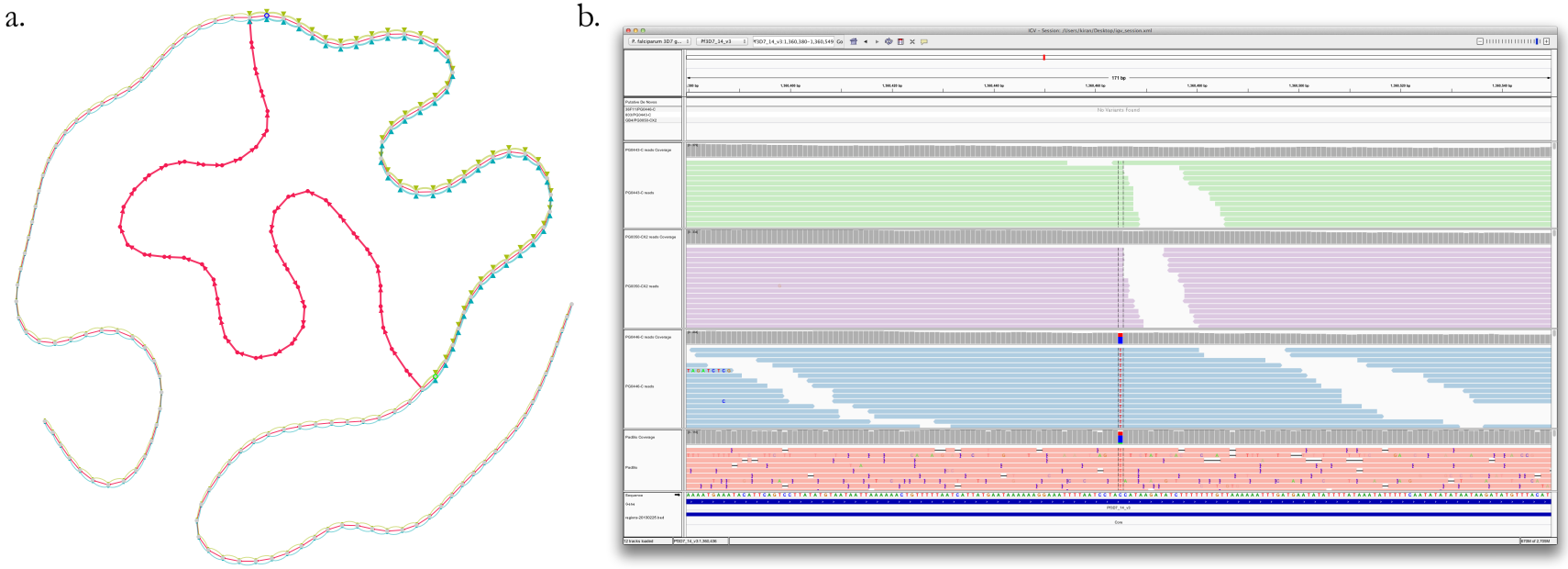


Figure 5.10: A C to T SNP that has likely not reached fixation in the sequenced population of ostensibly clonal parasites. a. Local subgraph at the site, wherein the child's graph contains paths that traverse the series of novel kmers as well as the parental kmers, indicating polymorphic status. b. IGV screenshot of the site showing reads and assembly information for the trio, aligned to the 3D7 reference genome. Top panel: positions of the called variants in the reference-basde analysis. Second panel: PG0446-C (803). Third: PG0050-CX2 (GB4). Fourth: PG0446-C (child). Fifth: Uncorrected PacBio reads from PG0446-C. Bottom: gene model track from PlasmoDB 9.0. Stacked barplots above tracks indicate the proportion of reads supporting each allele.

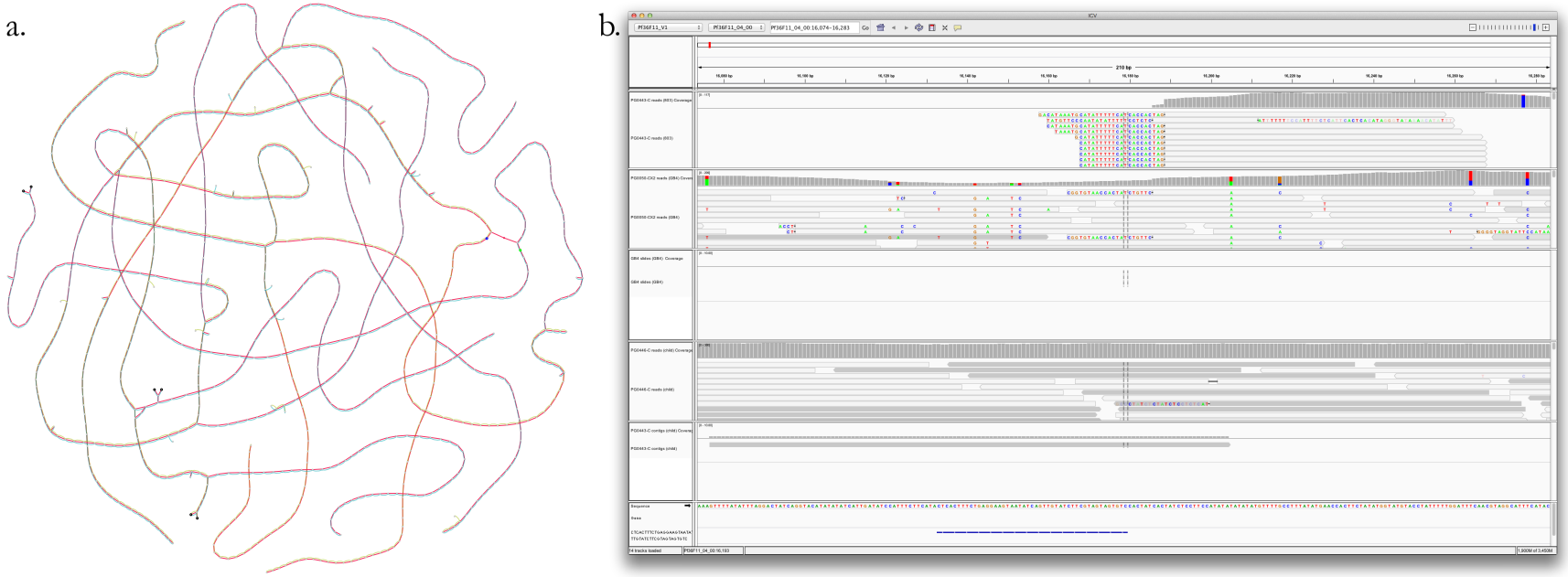


Figure 5.11: A strange event involving a single novel kmer. a. Local subgraph at the site; the novel kmer in question is depicted as a filled red vertex in the right side of the image. b. IGV screenshot of the site showing reads and assembly information for the trio, aligned to the PacBio PG0446-C assembly.

Table 5.3: Callset metrics in the PG0446-C sample of the trio

	All	In MalariaGen	Not in MalariaGen	Core	Accessory
SNPs					
all	28,613	5,742	22,871	13,700	14,913
de novo	128	0	128	7	121
Insertions					
all	15,163	4,543	10,620	12,540	2,623
de novo	38	0	38	8	30
Deletions					
all	17,762	5,160	12,602	15,123	2,639
de novo	30	0	25	5	25
MNPs					
all	144	0	144	65	79
de novo	0	0	0	0	0

of the gametocytes, the trio contains thousands of polymorphic sites (23,004 sites where the reference allele and alternate allele have non-zero coverage, 4,364 with reference and alternate allele coverage greater than $10x$). Both the accessory genome and polymorphic sites are easily removed. However, given that our graph-based approach is theoretically able to access the accessory genome and that we saw at least one polymorphic site in our graph-based *de novo* mutation callset, we chose not to exclude these sites at this stage.

We identified 196 potential *de novo* variants in the PG0446-C sample using the reference-based `HaplotypeCaller` approach and the filters described above. This is two orders of magnitude greater than the number of DNMs we found in the PacBio data. As expected, much of this putative variation appears in the accessory compartments of the genome, owing to the massive divergence between the reference and sampled haplotypes. However, even when we restrict our analysis to the core genome, we still find an order of magnitude more events than we expect in the graph approach.

We sought to establish the veracity of any of these putative *de novo* events. Assuming the child's underlying haplotype is the reference haplotype plus the variants discovered in that sample, we expected to find kmers spanning the DNMs to be present in the PacBio assembly, present in the clean graph of the child, and absent in the dirty graphs of the parents. Revisiting Algorithm 1 from Chapter 1, we combinatorically generated kmers spanning all *de novo* variants and any nearby variants (including variants that were filtered out) that would affect the local haplotype. We generated 30,360 such kmers, 3,740 of which are present in our PacBio draft reference for the child. 3,547 ($\approx$ 95%) of these kmers are present in the Illumina data for the child as well. 3,500 of these are present in the dirty graphs of the parents. This suggests that either all associated events (195/196)

are false positives, or that these represent errors in the PacBio draft assembly. PacBio reads are known to have an error rate between 11 and 15%,[?] and while it's generally thought that errors are random, it is unlikely they are *completely* random. One can easily envision a scenario where long homopolymers in the genome are truncated prematurely in the reads (although the errors are random, no read would have read through the region correctly). Thus many novel kmers could arise from regions where Illumina sequencers read the homopolymer region correctly, but PacBio assembled the region incorrectly.

The remaining 47 kmers represent a single SNP: the clearly homozygous SNP on chromosome 13 discussed in the previous section. This is the only event recapitulated between the PacBio dataset and these calls. However, we note that in selecting our coverage thresholds, we benefitted from prior knowledge. The coverage in the 803 parent is approximately $30x$, while coverage in the GB4 parent and the child are in excess of $60x$; our depth filtering threshold was specifically chosen to recover this validated variant. The SNP is shown in Figure 5.12; the PG0446-C (803) panel shows the drop in coverage in the downstream intergenic region beyond PF3D7_1350600. In this region, sequence complexity drops sharply, dominated by runs of As and Ts. Sequencing error may be accumulating in reads sampled from this region such that they fail to align back, thus contributing to the coverage falloff. Raising the coverage threshold would of course remove more false-positives, but there is no *a priori* reason to choose a threshold of $20x$ versus $30x$ (especially when average coverage in the 803xGB4 samples is around $200x \pm 100x$).

The polymorphic variant on chromosome 14 is not present in the reference-based callset, owing to the fact that we instructed `HaplotypeCaller` to assume a ploidy of 1. If we genotype this site alone without the ploidy specification, we can recover the event. However, removing the ploidy specification does not mean allowing any ratio of reference to non-reference alleles at a putative variant site. Instead, the software simply defaults to a ploidy of 2, which is not necessarily a correct assumption to make for our data. The ploidy setting merely enables the caller to adjust its allele balance expectations, and revising it upwards increases the chance that the caller will accept sequencing error as variation. Thus, removing the ploidy setting would necessarily increase the false-positive rate as well.

We visually inspected the alignments spanning the false *de novo* events using IGV. Many of these sites were seemingly polymorphic as well, in close proximity to one another, and in known high-diversity regions of the genome. Figure 5.13 depicts one such example of false-positives on chromosome 2. It is possible to begin adding heuristic filters to remove these events (e.g. rejecting events on allele balance, inter-marker distance, and

the genomic compartment in which they are discovered). However, there are two very good reasons not to do so.

First, consider the underlying cause of these artifacts. The myriad inconsistencies between the Illumina reads for PG0446-C, the PacBio reads for the same sample, and the presence of many mapping quality 0 reads, strongly suggests misalignment. The true haplotype from which these reads are sampled are clearly not in the reference genome, but are similar enough that many reads map anyway, and the precise placement is a function of read length and repetitiveness of the sampled haplotype. When we align reads from this region in the child to the PacBio reference for the child, we find that most map to chromosome 1, not chromosome 2. Filters can only accomplish one thing: reject potential false-positives. However, filters can do nothing to address the root issue: the reads were sampled from one region of the child's genome and aligned to a sample too genomically divergent in these loci to serve as a useful platform for comparison. Application of such filters would necessarily require sacrificing any ability to say something meaningful about enormous diverse regions of the genome.

Second, consider what these variants in a haploid genome must represent: *de novo* mutations that have occurred while the parasites have been in culture and have not reached fixation in the sample. These events necessarily must have occurred during mitosis, as mutations incurred during meiosis or present in the gametocytes would appear in every read spanning the site. Filtering any events based on allele balance would necessarily mean discarding every potential mitotic *de novo* mutation[6].

5.3 Summary

Despite only having a single PacBio validation dataset, it has taught us a tremendous amount. We have seen how the coverage filter we developed in Chapter 4 is insufficient; we need to relax the lower bound in higher quality data, and we must add an upper bound to remove errors arising from repeats. We have also seen how contamination can have a profoundly negative effect on our list of novel kmers, filling the list with error or unrelated kmers, and forcing us to examine too many places in the genome graph. Our general purpose machinery for traversing the graph until a specified stopping condition is met enables us to explore the graph at these contaminating kmers and remove all connected novel kmers from consideration. Effectively, they are tainted as being guilty by

[6]In a haploid genome, every true variant presenting as seemingly polymorphic must be a mitotic event, but as some mutations could have reached fixation in the sample, not every mitotic event will necessarily present as polymorphic.



Figure 5.12: A validated *de novo* SNP, recovered in the child amidst a great deal of sequencing error. Top panel: PG0443-C (803). Middle panel: PG0050-CX2 (GB4). Lower panel: PG0446-C (child).

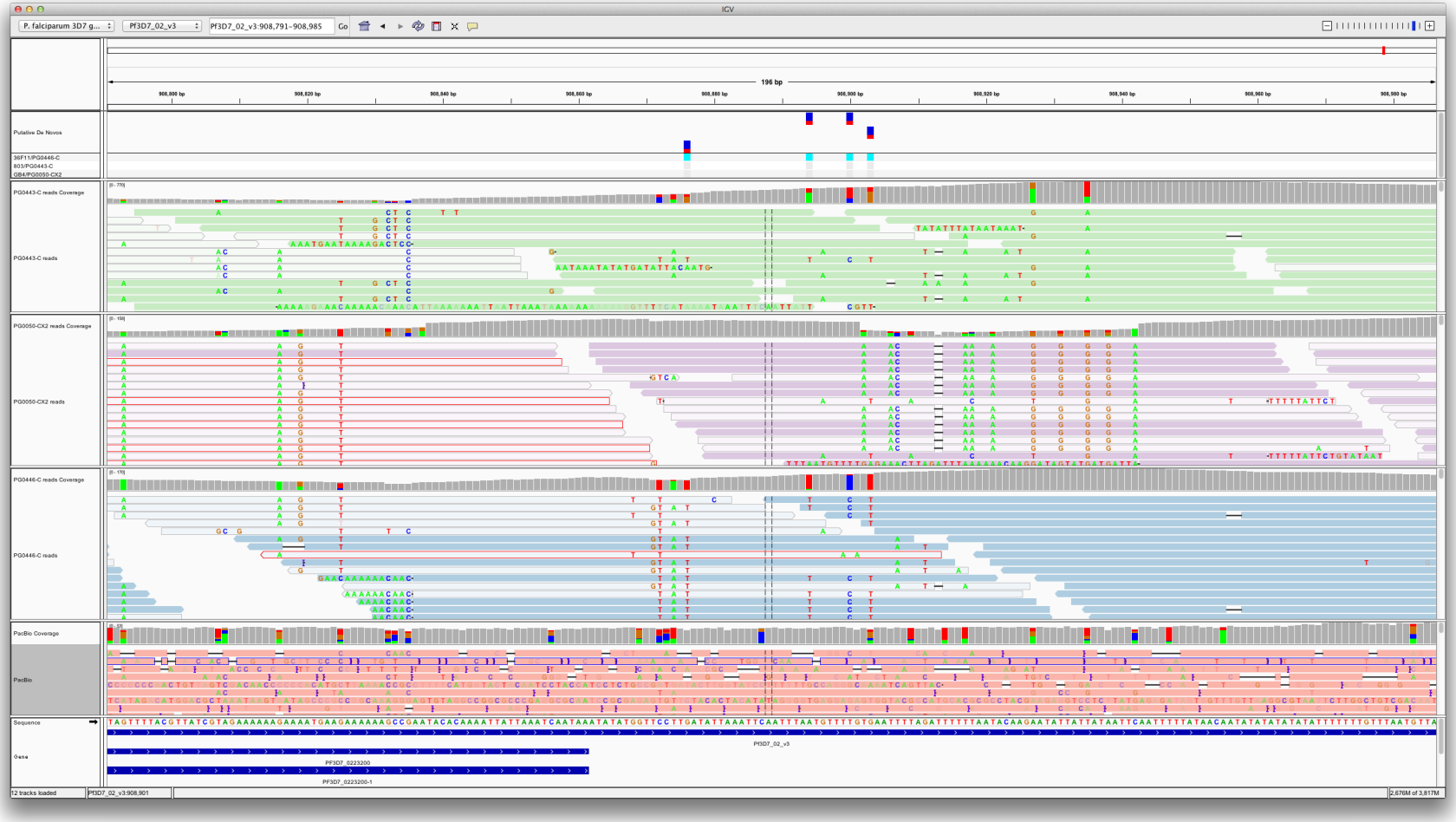


Figure 5.13: False-positive *de novo* variants in the 3′ subtelomeric region of chromosome 2. Top panel: positions of the called variants in the reference-based analysis. Second panel: PG0443-C (803). Third: PG0050-CX2 (GB4). Fourth: PG0446-C (child). Fifth: Uncorrected PacBio reads from PG0446-C. Bottom: gene model track from PlasmoDB 9.0. Stacked barplots above tracks indicate the proportion of reads supporting each allele. Reads with ambiguous alignments are displayed in a lighter shade.

association. With the inclusion of these filters, we finally demonstrate that our sensitivity and specificity in the test sample are essentially perfect, and that we can distinguish between both monomorphic events (presumably events that occurred prior to meiosis or mitotic events that have reached fixation) and polymorphic events (presumably events that occurred in subsequent mitoses after meiosis). This will be particularly important when we move to diploid data, as the vast majority of DNM events will appear as heterozygotes.

6 De novo mutations in Plasmodium falciparum

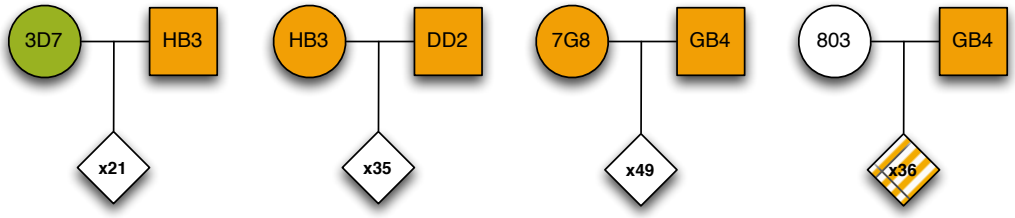


Figure 6.1: Relationships for samples in four *P. falciparum* crosses, with the number of QC-passing Illumina datasets for progeny shown. Green and orange shading indicates the availability of finished and draft assemblies, respectively. Orange stripes denote the availability of a draft assembly genome for a single progeny, used as validation for *de novo* events. Gender assignments for the parents are arbitrary, intended only to simplify discussion.

6.1 Introduction

No comprehensive catalog of *de novo* variation in the *P. falciparum* crosses currently exists. To date, the focus on the 3D7xHB3, HB3xDD2, 7G8xGB4 and 803xGB4 datasets has primarily been on discovering the genomic basis for phenotypic differences between the parents, and on characterizing novel antigenic forms in the children. However, the previous phenotypic studies have verified the presence of some large structural *de novo* variants - namely a handful of NAHR events involving antigenic genes. These known events serve as useful validation data for the most difficult form of variation for our software to detect. Furthermore, these previously observed events have occurred in low-complexity regions. These regions may confound DNA repair machinery and provide a substrate for the generation of *de novo* variation via polymerase slippage.[11] They are typically considered outside the reach of reference-based analyses (which constrain themselves to the roughly 20 megabases of so-called "core" genome wherein divergence between multiple isolates is much lower and short reads tend to align unambiguously[19]), but may be accessible with our graph-based approach. Finally, the *P. falciparum* genome is small enough to

Table 6.1: Summary of sequencing data for four *Plasmodium falciparum* crosses

	3D7xHB3	HB3xDD2	7G8xGB4	803xGB4
Samples	22	42	52	36
Samples QC+	21	35	49	36
Read length (bp)	76	76	76	100
Fragment size (bp)	300 ± 29	253 ± 48	293 ± 21	222 ± 10
Coverage	99 ± 38	121 ± 91	110 ± 40	205 ± 106
Platform	Illumina GAII	Illumina GAII	Illumina GAII	Illumina HiSeq 2000
Sequencing Date	2010	2009-2012	2010-2011	2014

be sequenced inexpensively on current third generation sequencing platforms, enabling high-quality reconstructions of the parental genomes and even one of the progeny. The parental samples facilitate DNM discovery, while the child sample serves as a validation dataset with which to evaluate our performance. These factors make the *P. falciparum* datasets a compelling study target.

6.2 Data processing

6.2.1 Initial data

We obtained short-read, paired-end whole genome sequence data for parent and progeny clones of the 3D7xHB3,[20] HB3xDD2,[22] 7G8xGB4,[101] and 803xGB4 crosses, generously provided to us by the MalariaGen project[1]. All samples were sequenced on Illumina platforms over the past five years using a PCR-free library preparation protocol known to reduce coverage biases associated with AT-rich templates. Avoiding PCR during library construction also removes the issue of replication errors that occur in early cycles being propagated to all subsequent copies, thus masquerading as *de novo* mutations. Across all four crosses, we obtained 152 samples prior to any quality control checks. The data is summarized in Table 6.1 and depicted in Figure 6.1.

6.2.2 Data processing for progeny

To enable our DNM calling process, we must build mother-father-child trio de Bruijn graphs for all of our samples. We rely on an external tool to do this, the McCortex assembly tool, authored by another student in the lab[2]. Although McCortex was originally

[1]More information, including links to the European Nucleotide Archive where the original reads are stored, can be found at `http://www.malariagen.net/apps/pf-crosses/1.0/`. Note that while the 803xGB4 data we were provided does come to us via MalariaGen, it was provided to them (and subsequently to us) as a personal communication from Michael Krause, Rick Fairhurst *et al.* at the NIAID. It is not yet part of the public dataset.

[2]Manuscript in preparation

designed to produce draft assemblies in the form of a set of contigs, the underlying data structure produced by the software to construct the assemblies is useful for us to perform *de novo* mutation calling. Below, we describe how the McCortex software was applied to provide the substrate for our DNM caller.

6.2.2.1 *Graph construction*

We built de Bruijn graph structures from the available Illumina data for each child with McCortex's `build` command, using a kmer size of 47 bp and discarding bases with a Phred-scaled quality score[102] less than $Q5$. The "dirty" graph (raw graph structure before any pruning is applied) was cleaned using McCortex's `clean` command, allowing the software to compute cleaning thresholds automatically, and falling back to trimming contiguous regions of the graph with coverage less than 2 in the event that automatic thresholds could not be calculated. McCortex's contigs, while strictly speaking were not used in our analysis, still serve a useful function in quality control checks. To generate contigs, paired-end reads were thread through the graph with the McCortex `thread` command and specifying a minimum fragment size of 0 bp, maximum fragment size of 400 bp, and choosing the software's (more conservative) one-way gap-filling procedure.

6.2.3 *Data processing for parents*

6.2.3.1 *Graph construction*

To complete the trio graphs, we must also build graphs for the parents. Constructing the parental graphs was a bit more involved than graphs for the children as there were often additional data sources to incorporate. For five out of the six parents, we benefitted from the availability of supplementary draft assemblies using PacBio data from the Pf3k project. These draft assemblies were constructed using version 3.0 of the Hierarchical Genome Assembly Process (HGAP3).[103] Briefly, the pipeline performs error correction via the `Quiver` module, which aligns two-thirds of the data to the longest third, emitting the consensus base and recomputed quality score at each position. The corrected data is then assembled with a modified version of the Celera assembler,[104] an assembler specifically

Table 6.2: Additional data availability for all *P. falciparum* cross parents

	3D7	HB3	DD2	7G8	GB4	803
Finished reference	x					
PacBio assembly	x	x	x	x	x	
Draft assembly		x	x	x		
Illumina data	x	x	x	x	x	x

written to utilize long-read information (it was originally designed for use on Sanger reads, which can be several hundred bp in length). While such reads were typically 500 bp in length, read lengths from PacBio RS II instruments are on average 10,000 to 14,000 bp in length, sometimes as long as 50,000 bp, two orders of magnitude longer than the reads used to assemble the finished 3D7 reference. Well-assembled long-read data should contain vastly fewer ambiguities than graphs produced from Illumina data, leading to fewer junctions that confound traversal in the resulting graphs. At variant bubbles, the PacBio data should assist in more easily determining a path from source to target, allowing us to type the variant properly. Table 6.2 shows all additional data used in constructing parental assemblies.

Parental graph construction was performed on each parental data source separately following the same workflow used for the children's graphs. The separate graphs were then combined into a single graph using McCortex's `join` command.

For most samples, the finished reference assembly and draft PacBio assemblies vastly outperform any assembly that could be produced with the Illumina data, and are an excellent basis for localizing events and determining their proximity to genes. However, as there is no high-quality sequence of the 803 genome, we were forced to construct one ourselves using the available Illumina data. After producing the cleaned graph with the McCortex workflow, we applied the software's paired-end read threading and contig emission steps. The assembly statistics for all six genomes are presented in Table 6.3. The assembly for 803 is exceedingly poor compared to the other assemblies, as expected from short Illumina reads. That the total sequence length is more than 2.5 times larger than a typical *P. falciparum* parasite is artifactual and discussed further below. These results are unchanged when graphs are constructed with a different algorithm (SGA[105]) instead.

We clarify a point of terminology: we refer to the 3D7 genome assembly from Gardner et al. as the **reference**. The genome assemblies constructed from PacBio or Illumina data shall be referred to as **draft assemblies**. While these assemblies can be considered references for the particular parasite from which the data was sampled, we shall avoid using that term for the sake of clarity as to the dataset being discussed.

We also note that although the quality of these draft assemblies may approach that of the reference sequence, as demonstrated in 5, using the assemblies in graph construction does not constitute a "reference-based" analysis. Specifically, a reference-based analysis in our parlance refers to the analysis that follows from aligning sampled reads to a reference genome. As discussed in 1, sampled loci absent from the reference results in mapping issues that compromise mutation detection. In contrast, our assembly-based analysis explicitly incorporates loci from both parents, and can handle multiple sources of data

(PacBio draft assemblies, Illumina reads, etc.). In a sense, our assembly strategy enables us to align data from the child to both parents simultaneously, choosing the appropriate haplotype background for a DNM as an inherent part of the graph traversal and mutation calling process.

6.2.3.2 *Transfer of gene models*

To determine if our variants fell within particular genes (e.g. antigens), we needed to produce gene annotations for each assembly. There is no single source of gene models for malaria parasites. Rather, there are high-quality annotations for 3D7, moderate-to-low quality annotations for HB3 and DD2 (the previously existing draft assemblies contain many errors and thus are comprised of very short contigs, which compromises the accuracy and completeness of the annotations). Still, given our concern about divergence in the antigenic loci, we chose to transfer the HB3 and DD2 annotations onto the respective PacBio assemblies, as these likely contained more specific and thus better quality antigen gene models than 3D7. For 7G8, GB4, and 803, where no pre-existing gene model is available, we transfered just the 3D7 annotations.

We transferred gene models onto the new genomes by examining existing reference and draft assemblies and their annotation sets, selecting each annotated exon sequence from its corresponding FASTA sequence file, and aligning it to the new genome using BWA. For 7G8, GB4, and 803, we transferred only the 3D7 annotations obtained from PlasmoDB release 26.[35] For DD2 and HB3, additional separate annotations exist from the Broad Institute[3]. These undoubtedly contain better representations of genes divergent from their 3D7 counterparts (particularly antigens). However, owing to the poor DD2 and HB3 assemblies to which they are associated, the annotations are likely to be incomplete. We therefore chose to transfer both the 3D7 and DD2 (HB3) annotations onto the new DD2 (HB3) assemblies. This has resulted in a slight over-annotation of genes in these two genomes, as shown in Table 6.3. This happens when the gene models slightly differ between 3D7 and the target (e.g. exon end definitions differ by as much as a single nucleotide), but refer to the same gene. Rather than simply choosing one definition, we permit both to remain as the PlasmoDB annotations are continuously maintained and improved, while the Broad's annotations have not been updated since April 2014.

6.2.3.3 *Generation of visualization resources*

We use the Circos[106] genomic comparison tool to generate whole-genome views of DNM activity in a sample's genome or in all samples from a single cross. To prepare these

[3] `http://www.broadinstitute.org/annotation/genome/plasmodium_falciparum_spp/`

Table 6.3: Statistics on all parental assemblies

	contigs	min length	max length	N50	total sequence	genes
3D7	16	5,967	3,291,936	1,687,656	23,332,831	5,777
DD2	16	6,094	3,257,617	1,661,885	22,682,339	8,284 (3D7, DD2)
7G8	17	6,094	3,311,228	1,560,458	22,832,195	5,530 (3D7)
GB4	26	7,376	3,360,747	1,565,171	23,525,386	5,576 (3D7)
HB3	28	6,094	3,378,065	1,593,993	22,812,563	7,467 (3D7, HB3)
803	148,826	47	17,610	1,042	59,118,463	4,663 (3D7)

visualizations, we require alignments between the contigs of a parental genome and the chromosomes of the reference. For 3D7, DD2, 7G8, GB4, and HB3, this is trivial as nearly the entirety of each chromosome has been successfully reconstructed. For those genomes, we simply lined up each assembled contig with its 3D7 counterpart (at the resolution of the image, details of a true alignment cannot be discerned). In the case of 803, we aligned each contig to the 3D7 genome using BWA to derive the mapping.

Finally, we computed a number of sequence composition and complexity metrics in tiled (i.e. non-overlapping) 2,500 bp windows using the SeqComplex tool[4]. The full list of metrics computed is as follows:

1. gc: GC content
2. gcs: GC skew (defined as $(G - C)/(G + C)$)
3. at: AT content
4. ats: AT skew (defined as $(A - T)/(A + T)$)
5. cpg: CpG content
6. ce: Sequence complexity by Shannon entropy[107]
7. cl: Sequence complexity by linguistic values[108]
8. cwf: Sequence complexity by Wootton and Federhen[109]
9. cz: Sequence compression factor (the ratio of uncompressed to compressed sequence file size, using the `gzip` compression utility)

These metrics were computed for each parental genome except 803, whose contig lengths were too variable to consistently satisfy the 2,500 bp window requirement, making comparison with GB4 cumbersome.

[4]https://github.com/caballero/SeqComplex

6.2.4 *Quality control*

We examined assembly metrics on all of our samples in order to flag low-quality samples for downstream analysis. We examined each sample for outlier behavior per cross in the number of unique kmers (total number of unique kmers in the graph after the automatic cleaning step - too few kmers indicate poor sequencing quality resulting in most data being thrown away) and contig N50 (the weighted median length of the contigs - very short contigs may indicate high sequencing error rate). The number of samples retained for analysis is in Table 6.1.

One expects that a newly assembled *P. falciparum* parasite should have a similar assembly length to the 3D7 reference sequence, or at least the other assembled parasites in this dataset. While this is true for nearly all samples in the 3D7xHB3, HB3xDD2, and 7G8xGB4 crosses, many of the 803xGB4 samples, shown in Figure 6.2, appear to defy this expectation. In this figure, all 803xGB4 samples are presented on the left. Two bars are shown to the right of each sample name: the number of kmers in the graph, and the total length of the assembly. Dashed vertical lines cutting across all of the samples indicate the mean number of kmers and assembly length of samples in the 3D7xHB3 (blue), HB3xDD2 (green), and 7G8xGB4 (orange) crosses. For those three crosses, the mean number of kmers matches expectations (approximately 2.3*e*7), as does the assembly length (23 Mbp, same as the 3D7 reference genome). However, this figure shows us that some 803xGB4 samples (including the 803 parental sample) exhibit assembly lengths well above the 23 megabases we expect. Several samples have assemblies in excess of 40 megabases. This outcome is unchanged even when other assembly software (e.g. SGA) is applied. The source of this excess sequence is unclear, but contamination by an organism not present in the BLAST database is a plausible hypothesis. We note that during investigation of this phenomenon, an update to our BLAST database revealed 64,676 kmers from the *Pseudomonas* genus that had previously been considered "novel", as the kmers in question were not yet present in the database. The *Pseudomonas* genus is a family of common environmental bacterium, one of the leading causes of opportunistic human infection,[110] and around 6.7 Mbp in length. It is the subject of drug resistance surveilence[111] at many labs including the Wellcome Trust Sanger Institute, where our *P. falciparum* samples were processed.

For labs that routinely perform a lot of sequencing, it is common for samples from different projects to be prepared simultaneously, and/or for samples to be multiplexed to maximize the use of sequencer capacity, both of which can contribute to sample cross-contamination.[112] Assembly software does not typically employ contamination checks prior to assembly, instead simply processing any data provided to it. It is likely that any

such contamination would have been assembled and incorporated mostly as disconnected regions of the de Bruijn graph - a series of vertices that do not connect to any other part of the *P. falciparum* genome. This might serve to inflate the novel kmer count if the child sample is contaminated but the parents are not, but such regions are easily identified and discarded. Therefore, we did not treat inflated assembly size to be a QC failure.

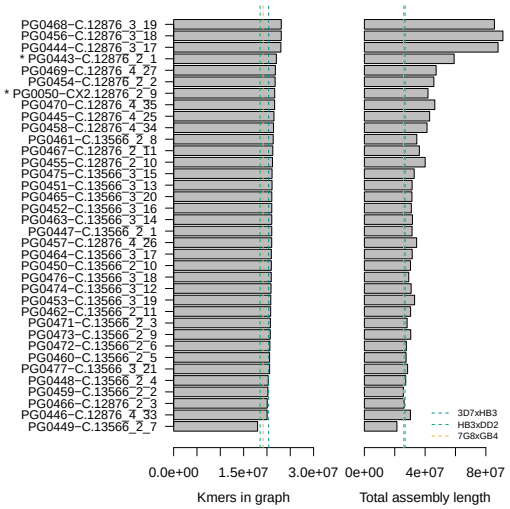


Figure 6.2: Number of kmers and assembly length in the 803xGB4 samples. Vertical dashed lines indicate the mean value of the metric in other crosses.

6.3 *De novo mutations in a single 3D7xHB3 sample*

6.3.1 *List of mutations*

A single validation sample, PG0446-C, is limited in its ability to inform us on the efficacy of the graphical variant calling approach, particularly as it apparently lacks an NAHR or other complex event for us to inspect. We thus turned our attention to a single sample from the 3D7xHB3 cross: PG0063-C. This sample is known to harbor an NAHR event between two *var* genes: *PF3D7_0100100* (situated in the 5′ subtelomere of chromosome 1), and *PF3D7_0223500* (the 3′ subtelomere of chromosome 2). Their validated presence in PG0063-C makes this sample a particularly useful test sample for examination.

Applying our novel kmer identification software (and the new filters described in Chapter 5 for dealing with contamination and establishing new coverage limits for kmers), we identified 314 novel kmers corresponding to 77 putative events. Post-calling filtration brought this list down to 7 events, described in Table 6.4. These events are annotated with the locus, the haplotypic background of the variant, variant type, closest gene (and the

associated product), and whether the variant falls within the MalariaGen variant calling exclusion mask. The variants can be seen in their genomic context in Figure 6.3.

Table 6.4: *De novo* variants in 3D7xHB3 progeny, PG0063-C

	locus	background	event	closest gene	product	within mask
1 (a)	1:27,780	3D7	SNP	PF3D7_0100100	PfEMP1 (var)	yes
1 (b)	1:27,782	3D7	MNP	PF3D7_0100100	PfEMP1 (var)	yes
1 (c)	1:27,854	3D7	MNP	PF3D7_0100100	PfEMP1 (var)	yes
1 (d)	1:29,441;2:923,083	3D7	NAHR	PF3D7_0100100;PF3D7_0223500	PfEMP1 (var)	yes
1 (e)	2:923,083;8:21,877	3D7	NAHR	PF3D7_0223500;PF3D7_0800100	PfEMP1 (var)	yes
1 (f)	8:21,877;1:30,087	3D7	NAHR	PF3D7_0800100;PF3D7_0100100	PfEMP1 (var)	yes
1 (g)	2:925,756;1:27,374	3D7	unknown	PF3D7_0100100;PF3D7_0223500	PfEMP1 (var)	yes
1 (h)	2:919,850	3D7	SNP	PF3D7_0223500	PfEMP1 (var)	yes
2	2:394,194	HB3	unknown	PF3D7_0209600	transporter	no
3	4:568,395	3D7	SNP	PF3D7_0412700	PfEMP1 (var)	yes
4	4:568,401	3D7	SNP	PF3D7_0412700	PfEMP1 (var)	yes

There are several noteworthy features of Table 6.4 and Figure 6.3. First, we discover a variety of event types, including SNPs (depicted in the plot with large circles), indels (triangles), multi-nucleotide polymorphisms (squares), and NAHR events (diamonds). A few events could not be typed (small circles).

Second, we are able to localize all mutational events within the parental genomes, even if the precise nature of the event could not be ascertained. Here, we benefit from aligning the parental sequences rather than the child's sequence or short 76 bp reads. The parental sequences are often longer than a read length, and should obviously match the parental genomes more closely. As described in Chapter **??**, this strategy of aligning the parental sequences to the parental genomes in order to discover the site where a mutation occurs provides the alignment software more genomic context and less variation with which to determine an appropriate home for the mutation.

Third, we are able to identify interchromosomal exchanges, including the known NAHR event between *PF3D7*_0100100 on the 5' end of chromosome 1 and *PF3D7*_0223500 on the 3' end of chromosome 2.[113] Curiously, there is another apparent interchromosomal exchange involving these two *var* genes and one on chromosomes 8. No such recombination is reported in the literature, and the event is supported by a very short contig aligned to chromosome 8 (< 50 bp). While it is conceivable that the NAHR machinery may switch templates to potentially involve content from a third chromosome, it is more plausible that the recombination process gave rise to a short region of low-complexity sequence that happens to occur elsewhere in the genome.

Fourth, we are able to detect mutations that occur in regions of the reference genome typically ignored in reference-based analyses. MalariaGen delibrately excluded a number of regions from processing, including repetitive regions (often telomeric, centromeric, or pericentromeric regions) as short reads could often not be aligned in long repetitive regions with confidence. It also excluded hypervariable regions (typically subtelomeric) as population diversity was too high to yield confident results in the reference-based analysis. As we are comparing each sample to a parental graph rather than a reference sequence that is an undetermined number of generations removed, our graphical approach can reconstruct and localize these events.

Fifth, our calls in this sample appear overwhelmingly concentrated in or near antigenic genes (8/9). Such a skew is interesting, especially given the fact that our algorithm is hypothesis-free and is not biased towards processing any particular region of the genome. It is possible that the skew reflects the selection history of the parasites, which will have been subject to immune pressure in the chimpanzee for several days prior to biopsy, culturing, and sequencing.

Sixth, the call rate is reasonably low; 9 events in this sample total. However, the first several events in the table appear to take place in the same *var* genes. Presumably, these events are all part of the single NAHR event. Counted together, the revised *de novo* mutation count is 4. Both are consistent with the expectation from the validation data that *de novo* mutations should be rare.

Subtelomeric regions (and the antigenic genes contained within) are known to be high in repetitive sequence. To visualize the potential relationship between sequence content and variant location, we examined a number of different sequence properties. Shown in Figure 6.3 above each parental assembly track is one of these metrics: the sequence compression ratio. Many of our DNM calls appear to follow closely with spikes in the local sequence compressibility measure. Quite often these spikes correspond with being in a masked repetitive region.

6.3.2 Manual examination of all de novo mutations

To establish the veracity of these calls, we manually examined all events in Table 6.4 in IGV and our custom graph viewing software. The screenshots and graph visualizations are presented in Appendix A.

6.3.2.1 Event 1 (a-h): NAHR

The first event in Table 6.4 appears to be the known NAHR event between antigenic genes *PF3D7_0100100* and *PF3D7_0223500*. Short reads are insufficient to capture the entire event in a single contig. Instead, we see pieces of the chromosomal exchange manifest as a series of SNPs, MNPs, and longer sequences that co-localize disparate regions of the genome onto a single contig.

The first three variants ($1a-c$: a SNP and two MNPs), are depicted in Figure A.1. Upon initial inspection, it is not immediatley obvious that the child's reads are sampled from the maternal or paternal haplotypes at all. Many apparent variants from the HB3 parent are not present in the child. Several loci appear to be polymorphic. It is tempting to conclude that the results are indicative of sequencing error or myriad mutational events sustained in mitosis. However, given the repetitive nature of the locus, the presence of ambiguously aligned reads (shown in translucent colors), and reads with several mismatches, it is more likely that many reads sampled from similar haplotypes are simply misaligned to the sole copy in the reference sequence.

The local subgraph for these events, shown in Figure A.2, exhibits a complicated structure containing several tendrils to homologous (but irrelevant) regions of the genome.

Despite this complexity, two bubble motifs (drawn with thicker lines) can clearly be observed, encapsulating the first SNP/MNP[5] and second MNP. Inspection of individual vertices in the event reveals all kmers to be of nominal coverage ($> 40x$), and the subgraph itself is devoid of any suspicious kmers (contaminants, exceedingly low-coverage sequence, or dirty kmers) that would be suggestive of error.

The next four events capture the NAHR event itself and are displayed in Figure A.3. The graphical view shows a very simple structure, unmarred by indications of sequencing error. Starting from the upper right, the child and 3D7 assemblies track along chromosome 1 until the child's assembly forks off, apparently passing through a few kmers from chromosome 8, rejoining at chromosome 2. Meanwhile, the 3D7 genome continues to track along chromosome 1 until it rejoins a region of the child's assembly (towards the bottom of the image). Here, 3D7 and the child fork again, with the child's assembly connecting again to chromosome 2. The IGV panels show the reads aligned to these regions of the 3D7 reference genome. In the *b* panel (showing the 5′ end of chromosome 1, the reads appear to be sampled largely from the 3D7 parent until an abrupt change midway through the track (denoted by the red bar). Similarly, the *c* panel shows a region on chromosome 2 that supports the 3D7 haplotype over a window substantially longer than a read length (300 bp).

The last mutation seemingly associated with the NAHR event is a single SNP; the context is shown in Figure A.4. A naive analysis would not assign *de novo* status to such an event as there appears to be some evidence for this allele in the HB3 parent. However, the left and right sections of the HB3 reads flanking the putative variant exhibit an extraordinary number of mismatches and are clipped off in the alignment. A small number of reads have mapping quality 0, indicating that the aligner had no confidence in their placement. While the 3D7 parent and the child are covered over this region to $> 150x$, the HB3 sample is only covered to $20x$, with many apparent holes in the coverage across the 200 bp region shown. Finally, the child's alignments are very clean across this region: no mismatches, no clipped reads, and consistent coverage. It is likely that the HB3 alignments are in error - a homologous region from another *var* gene have landed here because the proper haplotype is not represented in the reference. Our graphical method detects the mutation on the correct haplotypic background and calls it accordingly.

6.3.2.2 Event 2: unknown

Event 2 is an unknown event, seemingly localized to the 5′ telomere of chromosome 2. However, examination of the subgraph (Figure A.5) reveals the event in question likely

[5] As these events are separated by less than a kmer length, they present as a single bubble.

stems from a graphical "tangle": a region of the graph where long, uninterrupted, singly-connected regions of the graph collapse into a messy structure wherein every vertex appears to connect to many others. The IGV screenshot for this region (Figure A.6 shows a number of errorful reads spanning a homopolymer region. If these results are due to a recurrent error by the sequencer (sometimes skipping one, two, or many bases), they may occur slightly differently between samples and often enough to produce enough kmers to overcome the novelty coverage thresholds. This would give rise to a stretch of novel kmers near the low-complexity region that fails to resolve into a *de novo* variant.

One can conceive of a few simple checks to remove these events from our callset (e.g. walking the graph within a certain radius to discover potential nearby tangles). However, such regions may also be biologically interesting, potentially giving rise to small indels induced by DNA repair over those low-complexity sequences. Unclassified events are not harmful to us, as they will not count in measures of SNP, MNP, indel, or NAHR event mutation rates. They simply eat up novel kmers without assigning any identity to the kmers. For these reasons, we have not made strong efforts to filter them out.

6.3.2.3 Events 3 and 4: two SNPs

Figure A.7 show the last events in this sample, two SNPs set 5 bp apart, and reveal an interesting limitation of our graphical approach. While our alignments of the parental sequence suggest these events occur on chromosome 4, no evidence for the events are found at that locus. A BLAST search reveals the proper home (barring a single nucleotide) to be on chromosome 1, within the *PF3D7*_0100100 *var* gene - the same gene involved in the NAHR event described above. This locus is shown in Figure A.8. The misplacement of these events are likely due to an error in the reference sequence. A single nucleotide error in the reference sequence prevents us from finding the true home of the kmers spanning the error. By chance, those aberrant kmer exist elsewhere in the genome: a *var* gene on chromosome 4. This misleads us into choosing the home on chromosome 1 instead. Thus, while the events are likely to be real, our placement is erroneous.

6.3.3 Rejected events

Post-calling filters rejected nine putative *de novo* events. These events were rejected for a variety of reasons: either internal use of dirty kmers exceeded a maximum threshold of 10% (5), the novel kmers were found at the extremeties of a graph with no apparent connection to any other part of the genome (2), or the contig failed to align to any part of the genome (2). Most rejected events were of an "unknown" event type (no label could be

reasonably assigned to the putative variant). One was a potential recombination event, but upon inspection of the data in IGV, a recombination event is not readily apparent.

Events lacking alignments do not visualize well; there is nothing to display in IGV and the graph itself is non-descript (potentially an unrecognized contaminant). An event with dirty kmers and an event with novel kmers without connection to the rest of the graph are shown in Figure 6.4. Notice that these filters are clearly not orthogonal; had the event with dirty kmers remained in consideration, it would have been flagged as a disconnected event as well.

6.3.4 Novel kmer accounting

Figure 6.5 details the cumulative usage of novel kmers in the PG0063-C sample. Of the 314 novel kmers detected, 154 were discarded as part of events suspected not to be true mutational events. The remaining 160 are present in mutational events passing filters. 156/160 (97.5%) of novel kmers could be assigned to a definable mutational type.

6.4 De novo mutations in all crosses and samples

6.4.1 Novel kmer use

We applied our software to determine the number of novel kmers in all samples and all crosses. These per-sample counts are shown in Figure 6.6. Counts are largely consistent across samples and crosses, typically ranging from a few hundred to a few thousand novel kmers. Notably, the 803xGB4 samples have substantially more novel kmers than the other crosses. While the median number of novel kmers per sample in the 3D7xHB3, HB3xDD2, and 7G8xGB4 crosses are 340, 278, and 173 respectively, the median count in the 803xGB4 cross is 1,650 with a very large variance. Given that we've seen the larger-than-expected assembly lengths and a tremendous amount of data contamination, it seems likely that this excess is due to incomplete mitigation of those factors via our filtration process. Rather than the samples harboring a vast amount of *de novo* mutation activity, we expect to find a large fraction of calls in these samples to be unclassifiable.

Next, we applied our DNM calling software to all samples. To see how many of the novel kmers were explained by variants passing filters, unknown events passing filters, or events failing filters, we superimposed the variant calls' novel kmer usage atop the total number of novel kmers per sample. As evident in Figure 6.7, there is substantial variation between samples. As predicted, many of the 803xGB4 variants are unclassified. The median fraction of kmers explained by definable variants passing filters is a mere

29%, but when events failing filters are removed from consideration, this number jumps to 70% of novel kmers explained.

Figures 6.8-6.11 show all monomorphic and typed variation in the 3D7xHB3, HB3xDD2, 7G8xGB4, and 803xGB4 crosses. As we have seen that untyped variation (events classified as "unknown" are typically some sort of artifact from contamination or graphical tangles, we have suppressed all untyped events in these plots).

6.4.2 Variants per sample and cross

Within each sample, we removed allelic variants appearing within the same gene as an NAHR event since, as we saw in the previous section, partial recovery of NAHR events can lead to erroneous accounting of the true mutation rates. This rejects 211 additional events, leaving us with 7,193 that pass filters.

The monomorphic variants per sample and cross are summarized in Tables 6.5 and **??**. Each entry shows the total number events summed across all samples in the cross, along with the median number of events per sample.

Several striking aspects of this table stand out. First, *de novo* variant rates are very low. Each sample harbors only one or two mutations of each class. SNPs are by far the most common variant. Deletion events are the next most abundant event. Although these are not STR contractions, many of the deleted alleles appear quite repetitive in sequence, possibly inexact tandem repeats that suffered a slippage error in replication. Insertions and multinucleotide polymorphisms occur rarely. We detect no inversions.

Second, rates are quite inconsistent between crosses. For example, the 803xGB4 cross has a much higher rate of insertion mutations than the other three crosses. HB3xDD2 has higher rates of MNPs than all other crosses. 3D7xHB3 has lower rates of mutation for every variant class except SNPs. Such heterogeneity is curious, particularly as data quality and contamination metrics would not seem to explain the wide variance. We discuss this more below.

Third, the 803xGB4 cross seems to harbor an unusual amount of NAHR activity. However, we believe skepticism is warranted for this type of variant in this cross. The key to detecting NAHR events is determining that many kmers co-localize on the same contig in the child but originate from different chromosomes in the parent. With a poor 803 parental assembly constructed from short Illumina reads, many events may span multiple parental contigs but be part of the same chromosome. A possible solution would be to align these parental contigs to the 3D7 reference and apply heuristics, rejecting any event where different contigs involved in the event align to the same chromosome. However, this solution is not necessarily workable. The assembled contigs are short, and the

Table 6.5: Monomorphic variants and rates of occurrence per cross and per sample

	3D7xHB3	HB3xDD2	7G8xGB4	803xGB4
Samples	20	36	26	33
SNP	26 (1.37)	45 (0.94)	77 (1.05)	187 (1.34)
Insertions	3	15	13	33
STR expansion	1 (0.00)	8 (0.00)	8 (0.00)	25 (0.45)
Tandem duplication	1 (0.00)	2 (0.00)	3 (0.00)	1 (0.00)
Other	1 (0.00)	5 (0.00)	2 (0.00)	7 (0.00)
Deletions	5	37	39	41
STR contraction	4 (0.00)	14 (0.41)	18 (0.28)	17 (0.42)
Other	1 (0.00)	23 (0.45)	21 (0.41)	24 (0.44)
MNP	1	17	15	50
Inversion	0 (0.00)	0 (0.00)	0 (0.00)	0 (0.00)
Other	1 (0.00)	17 (0.31)	15 (0.00)	0 (0.00)
NAHR	6 (0.00)	13 (0.00)	4 (0.00)	15 (0.00)
Recombination	6 (0.00)	32 (0.00)	19 (0.42)	59 (0.54)
unknown	21 (0.00)	828 (0.00)	1548 (0.00)	1513 (0.00)

divergence between 803 and 3D7 is very high, particularly in the subtelomeric regions where true events almost certainly originate. Therefore, alignment of the parental contigs to the 3D7 reference is not expected to solve the problem. We have initiated an effort to have the 803 parent sequenced on PacBio instruments. At of this writing, such data is not expected to be available for several months.

6.4.3 Manual examination of every event in the 3D7xHB3 cross

To estimate a false discovery rate, we manually examined all 83 monomorphic and polymorphic events in the 3D7xHB3 cross using our graph visualization software and IGV. 50/83 events appear to have clear support in both visualizations. The vast majority of these events are so-called bubble-motif variants, with the exception of the NAHR event in PG0063-C and an unknown event (a misclassification - an event is called in the correct location, but upon closer inspection, appears to be two biallelic SNPs in close proximity, confusing the calling software). Another 12 events had uncertain validation status; one was a putative NAHR event, another was a recombination event, two were polymorphic variants, and the remaining 8 could not be typed. Finally, 25 events were clear false-positives owing to various issues (e.g. proximity to a repetitive region or otherwise appearing artifactual). The bulk of such events could not be typed (12) or were non-bubble-motif (7). It was rare for bubble-motif variants to manifest as false-positives; occuring for a single insertion and 5 SNPs. In the case of the SNPs, all false-positives

appeared as a result of misclassifying an inherited event as *de novo* due to the mutations' proximity to another event - usually a polymorphic variant. The polymorphic event makes it appear that the haplotypic background of the variant is different from the true background which contains the SNP. Thus, when our caller goes to examine the evidence, it finds novel kmers due to the polymorphic event, finds the inherited mutation in close proximity, misassociates the two, and calls the variant.

Ignoring untyped events and pessimistically assuming that any event with uncertain validation status was false, we estimate our FDR to be 18/68, or approximately 26%. However, we should expect that NAHR and recombination events carry a much higher false-positive rate and drive the bulk of this estimate. After removing these events from consideration, we find our FDR is 9/54, or approximately 16%.

6.4.4 *Recovery of published var gene recombinations*

The underlying dataset for our study was partially available to Claessens *et al.* at the time of their study. In their reference-based work, Claessens *et al.* identified many *var* gene recombinations involving exon 1, including subtelomeric members and centrally-located members of the gene family. As core *var* gene recombinations have not been reported in the past, we sought to confirm these events with our reference-free methods. The calls from Claessens *et al.* and our study are summarized in Table 6.6. Of the 12 calls they make in the shared 3D7xHB3 and HB3xDD2 samples, a single event is recapitulated by our graphical method: the well-known PG0063-C event. 6 more of their events find no visual support in our reference-free kmer accounting method, which would otherwise indicate some sort of structural variant activity (even it is unable to prove that the event occured between two specific genes in question). 3 more events do show some signs of NAHR activity, but not between the genes listed by Claessens *et al.*. No recombinations between core *var* genes validate visually.

In contrast, examining the 3D7xHB3 and HB3xDD2 crosses, we find 5 events, all of which are supported by the kmer recovery diagnostic plots. Figure 6.12 shows an example of a seemingly false-positive core *var* gene recombination (a) and a true event (b).

Table 6.6: NAHR events in Claessens *et al.* and this study

Sample	Cross	In Claessens *et al.*	In this study	Support in kmer recovery
PG0015-C	HB3xDD2	n	y	y
PG0018-C	HB3xDD2	n	y	y
PG0020-C	HB3xDD2	y	n	n
PG0025-C	HB3xDD2	n	y	n
PG0030-C	HB3xDD2	y	n	n
PG0034-C	HB3xDD2	y	n	y
PG0035-Cx	HB3xDD2	n	y	n
PG0036-C	HB3xDD2	y	n	wrong genes
PG0038-C	HB3xDD2	y	n	y
PG0044-C	HB3xDD2	y	n	n
PG0054-C	3D7xHB3	y	n	n
PG0055-C	3D7xHB3	n	y	y
PG0058-C	3D7xHB3	y	n	wrong gene
PG0063-C	3D7xHB3	y	y	y
PG0067-C	3D7xHB3	n	y	n
PG0068-C	3D7xHB3	y	n	n
PG0069-C	3D7xHB3	y	n	n
PG0070-C	3D7xHB3	n	y	n
PG0071-C	3D7xHB3	n	y	y
PG0072-C	3D7xHB3	y	n	wrong gene
PG0079-CW	7G8xGB4	y	n	NA
PG0082-C	7G8xGB4	y	n	NA
PG0085-C	7G8xGB4	y	y	NA
PG0087-C	7G8xGB4	y	n	NA
PG0088-C	7G8xGB4	y	y	NA
PG0095-C	7G8xGB4	y	n	NA
PG0100-CW	7G8xGB4	y	n	NA
PG0103-CW	7G8xGB4	y	n	NA
PG0105-CW	7G8xGB4	y	n	NA
PG0107-C	7G8xGB4	y	n	NA
PG0112-C	7G8xGB4	NA	y	NA
PG0113-C	7G8xGB4	NA	y	NA
PG0444-C	803xGB4	NA	y	NA
PG0456-C	803xGB4	NA	y	NA
PG0459-C	803xGB4	NA	y	NA
PG0468-C	803xGB4	NA	y	NA
PG0469-C	803xGB4	NA	y	NA

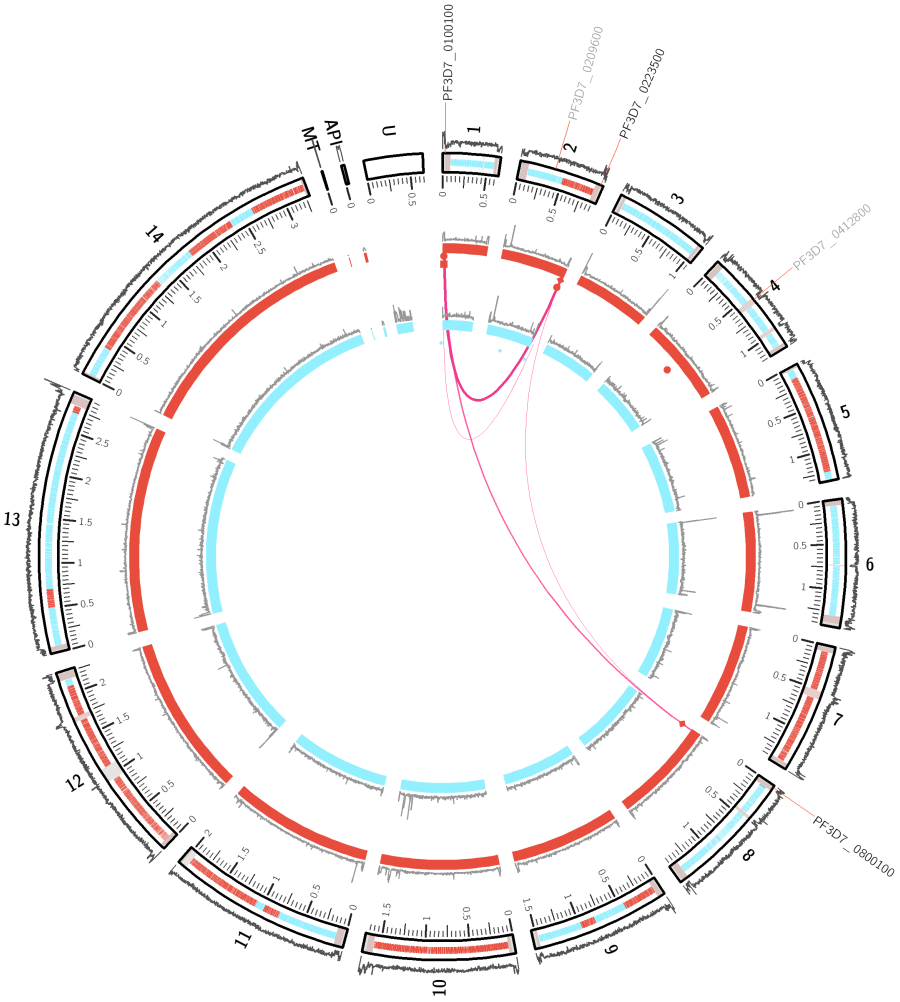


Figure 6.3: Circos visualization of all *de novo* mutations in PG0063-C, positioned on the reference sequence (outer ring), aligned maternal assembly (middle), and aligned paternal assembly (inner). Unplaced parental contigs are concatenated and placed in the "U" pseudochromosome. Each parental ring is annotated with a sequence compressibility score for every 2,500 bp shown above their ideograms, while the reference ring is annotated with mean read coverage per 2,500 bp. Within the reference ideogram, all reference-based variant calls made in this sample from the MalariaGen project are shown, color-coded to reflect their parentage. Regions that were masked out of MalariaGen's variant calling process are shown as grey bars. Variants are shown as glyphs below the parental ideograms, shape-coded for type (small circle: unknown, large circle: SNP, triangle: indel, square: MNP, diamond: NAHR event). The color and positioning of the glyph reflects the haplotypic background upon which the event occurred. Interchromosomal structural variation and gene-conversion events are additionally denoted as arcs linking the relevant parts of the genome. The closest gene to the event is reported in black if the mutation falls within the gene and as grey if it falls outside the gene.

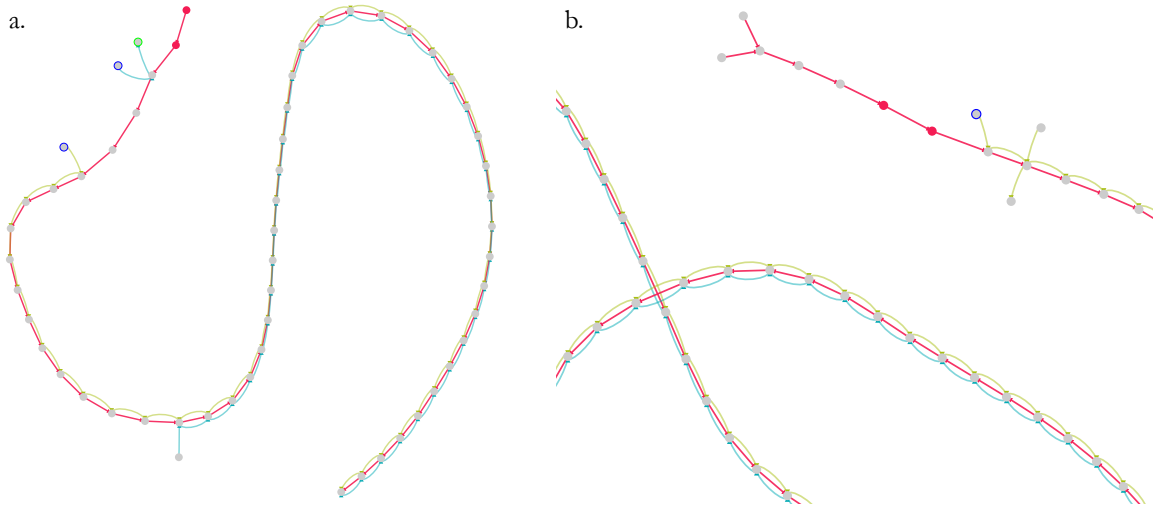


Figure 6.4: Graph motifs for discarded events. a. Disconnected events: novel kmers that fail to link back to the genome. b. "Dirty" events: the dirty kmers (shown in grey with red edges) make up more than 10% of the novel kmers in the region.

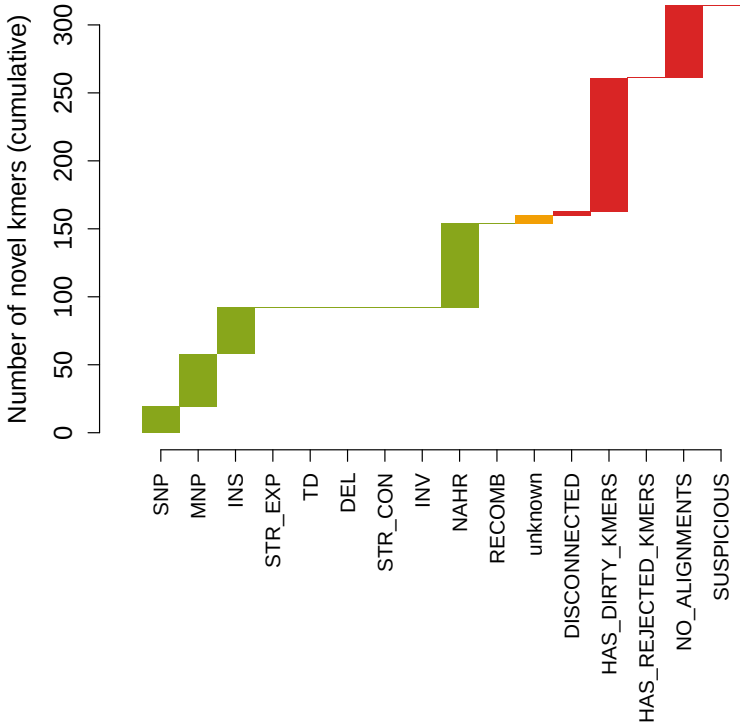


Figure 6.5: Cumulative accounting of novel kmers and the mutation types to which they correspond. Events passing filters are depicted in green. Those failing filters are in red.

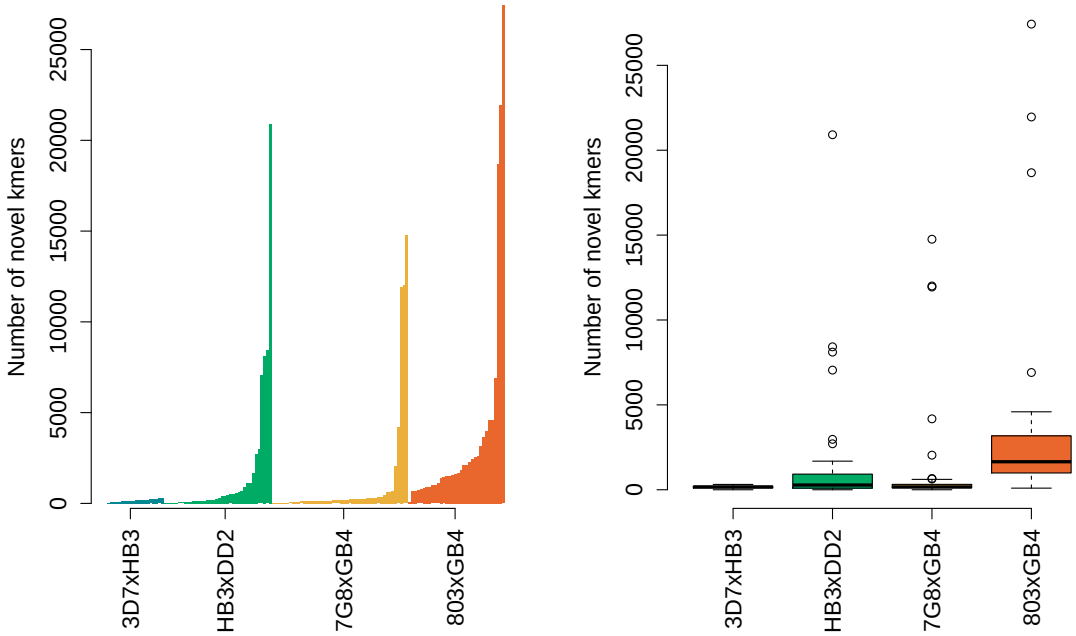


Figure 6.6: Novel kmers per sample and cross. Left: number of novel kmers in each sample, after filtering. Right: distribution of novel kmers per sample in each cross.

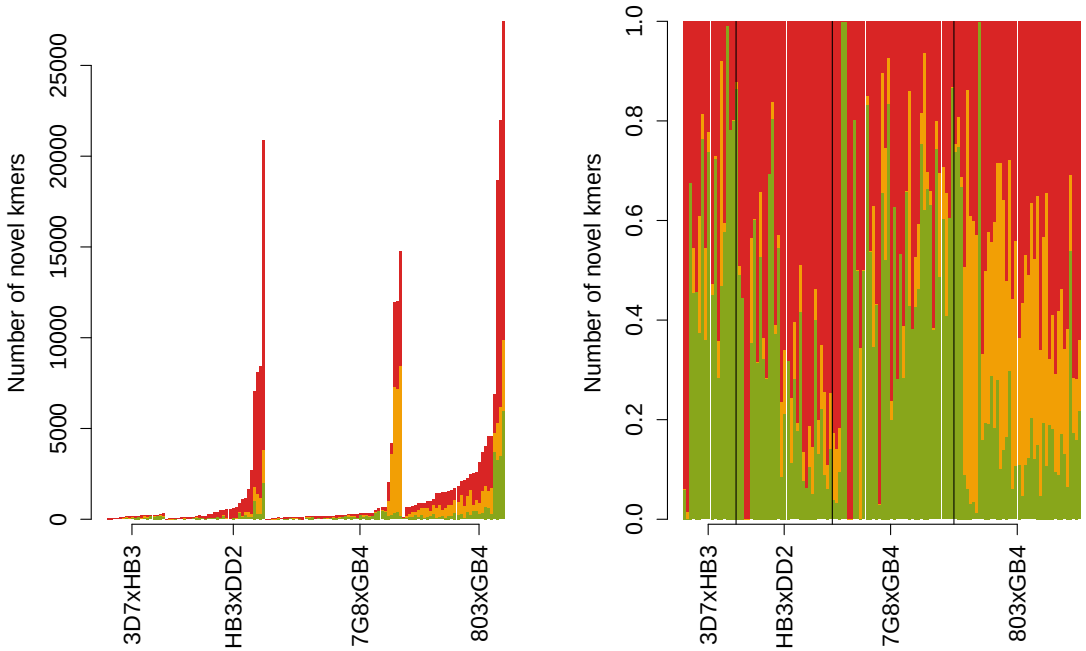


Figure 6.7: Novel kmer usage by variants per sample and cross. Left: novel kmers used in variants (green), in unknown events (orange), and events failing filters (red). Right: fraction of novel kmers used by variants (green), unknown events (orange), and events failing filters (red).

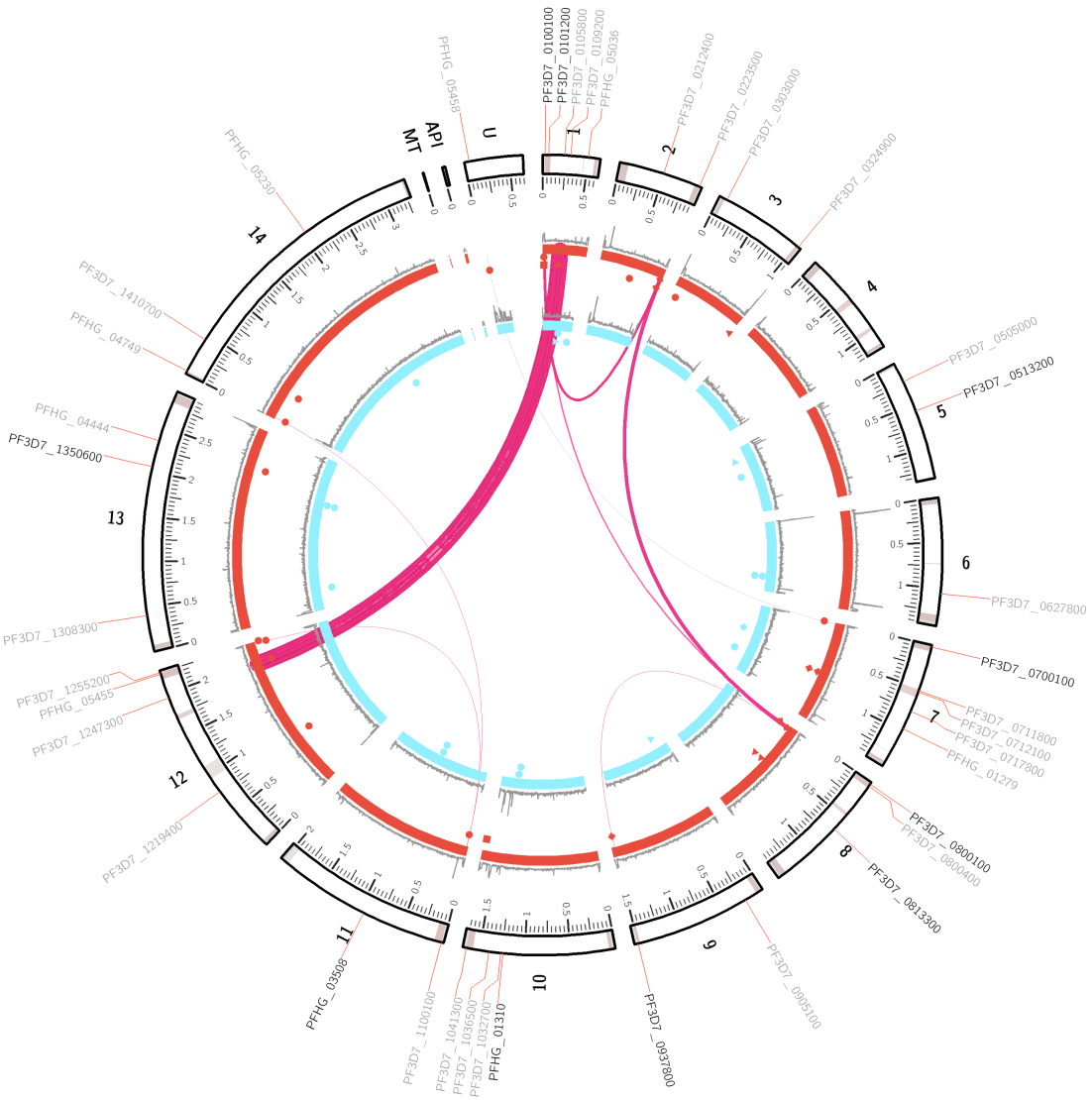
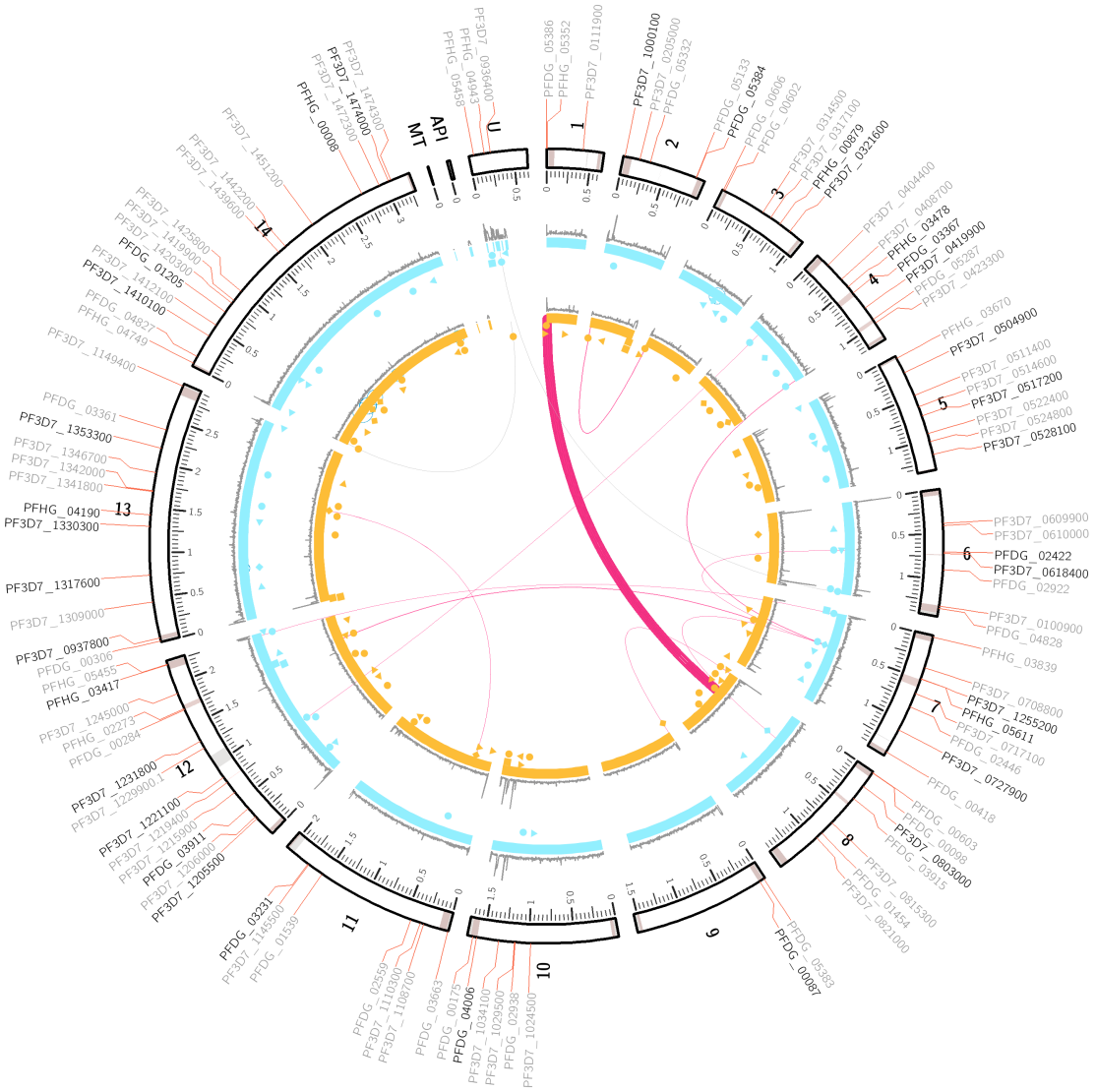


Figure 6.8: All monomorphic, typed *de novo* variants in the 3D7xHB3 cross (untyped events suppressed). Outer ring: chromosome ideogram from the 3D7 reference genome. Middle ring: mapping of chromosomes from the 3D7 parental assembly (finished genome; identical to the 3D7 reference) to the reference. Inner ring: mapping of HB3 parental assembly (draft genome from PacBio sequencing) to the reference. SNPs, MNPs, and indels are depicted by circles, squares, and triangles, respectively. NAHR events shown as arcs linking the involved chromosomes; thickness of each arc is proportional to the contig length.


API
MT
PF3D7_1000100
PF3D7_1474000
PFDG_01205
PF3D7_1410100
PF3D7_1353300
PF3D7_1330300
PF3D7_1317600
PF3D7_0937800
PF3D7_1231800
PF3D7_1221100
PF3D7_1205500
PF3D7_1245000
PF3D7_1229900.1
PF3D7_1451200
PF3D7_1442200
PF3D7_1436600
PF3D7_1425900
PF3D7_1419900
PF3D7_1412100
PF3D7_1149400
PF3D7_0111900
PF3D7_0205000
PF3D7_0314500
PF3D7_0311100
PF3D7_0321600
PF3D7_0404400
PF3D7_0408700
PF3D7_0419900
PF3D7_0423300
PF3D7_0504900
PF3D7_0511400
PF3D7_0514600
PF3D7_0517200
PF3D7_0522400
PF3D7_0524800
PF3D7_0528100
PF3D7_0609900
PF3D7_0610000
PF3D7_0618400
PF3D7_0100900
PF3D7_0708800
PF3D7_1255200
PF3D7_0727900
PF3D7_0803000
PF3D7_0815300
PF3D7_0821000
PF3D7_1024500
PF3D7_1029500
PF3D7_1034100
PF3D7_1108700
PF3D7_1110300
PF3D7_1245500
PFDG_05386
PFDG_05352
PFDG_05133
PFDG_05384
PFDG_00606
PFDG_00602
PFDG_05332
PFDG_00879
PFDG_03478
PFDG_03367
PFDG_05287
PFDG_04223
PFDG_03670
PFDG_02422
PFDG_02922
PFDG_04828
PFDG_03839
PFDG_05611
PFDG_02446
PFDG_00418
PFDG_00603
PFDG_00099
PFDG_03915
PFDG_01454
PFDG_05363
PFDG_00087
PFDG_04006
PFDG_00175
PFDG_03663
PFDG_02938
PFDG_03231
PFDG_01539
PFDG_03911
PFHG_00008
PFHG_04190
PFHG_03417
PFHG_05455
PFHG_00306
PFHG_02273
PFHG_00284
PFHG_04827
PFHG_04749
PFHG_05458
PFHG_04043
PFHG_00879
PFHG_03670
PFHG_05611
PFHG_03839
PFHG_00099
PFHG_03915
PFHG_00087
PFHG_04006
PFHG_00175
PFHG_03663
PFHG_02938
PFHG_03231
PFHG_01539
PFH

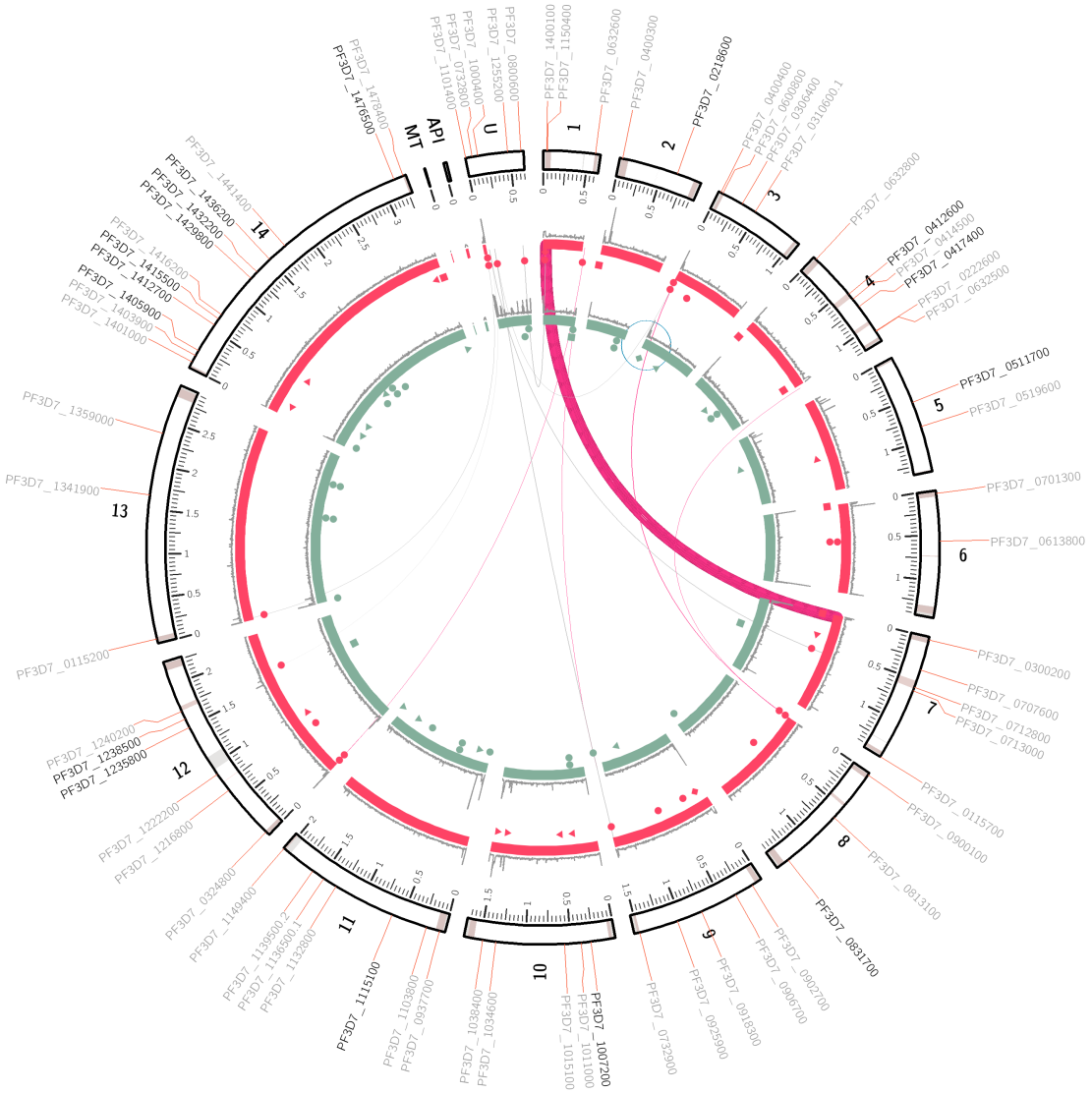


Figure 6.10: All monomorphic, typed *de novo* variants in the 7G8xGB4 cross (untyped events suppressed). Outer ring: chromosome ideogram from the 3D7 reference genome. Middle ring: mapping of chromosomes from the 7G8 parental assembly (draft genome from PacBio sequencing) to the reference. Inner ring: mapping of GB4 parental assembly (draft genome from PacBio sequencing) to the reference. SNPs, MNPs, and indels are depicted by circles, squares, and triangles, respectively. NAHR events shown as arcs linking the involved chromosomes; thickness of each arc is proportional to the contig length.

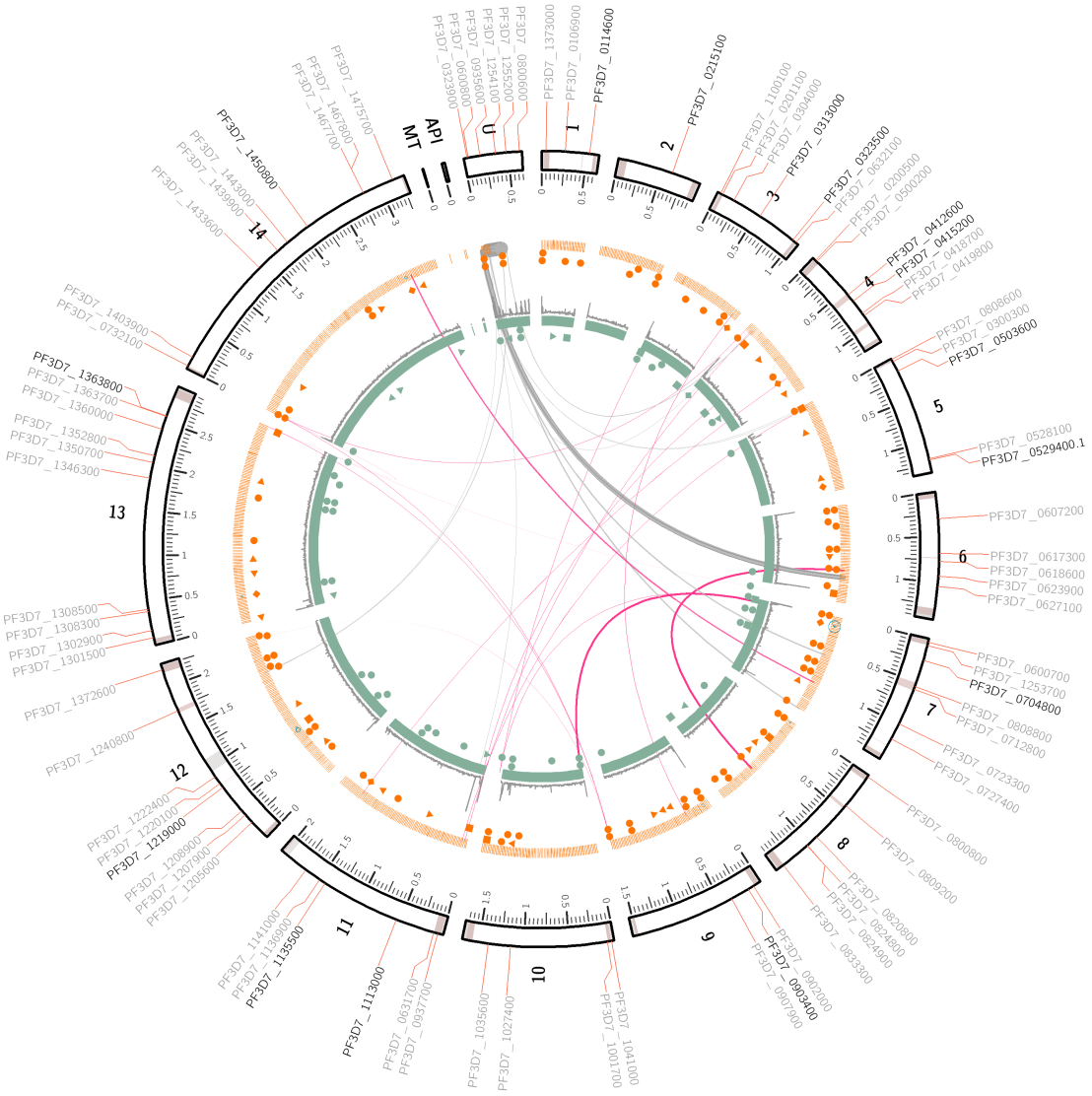


Figure 6.11: All monomorphic, typed *de novo* variants in the 803xGB4 cross (untyped events suppressed). Outer ring: chromosome ideogram from the 3D7 reference genome. Middle ring: mapping of chromosomes from the 803 parental assembly (low-quality draft genome from Illumina sequencing) to the reference. Inner ring: mapping of GB4 parental assembly (draft genome from PacBio sequencing) to the reference. SNPs, MNPs, and indels are depicted by circles, squares, and triangles, respectively. NAHR events shown as arcs linking the involved chromosomes; thickness of each arc is proportional to the contig length.

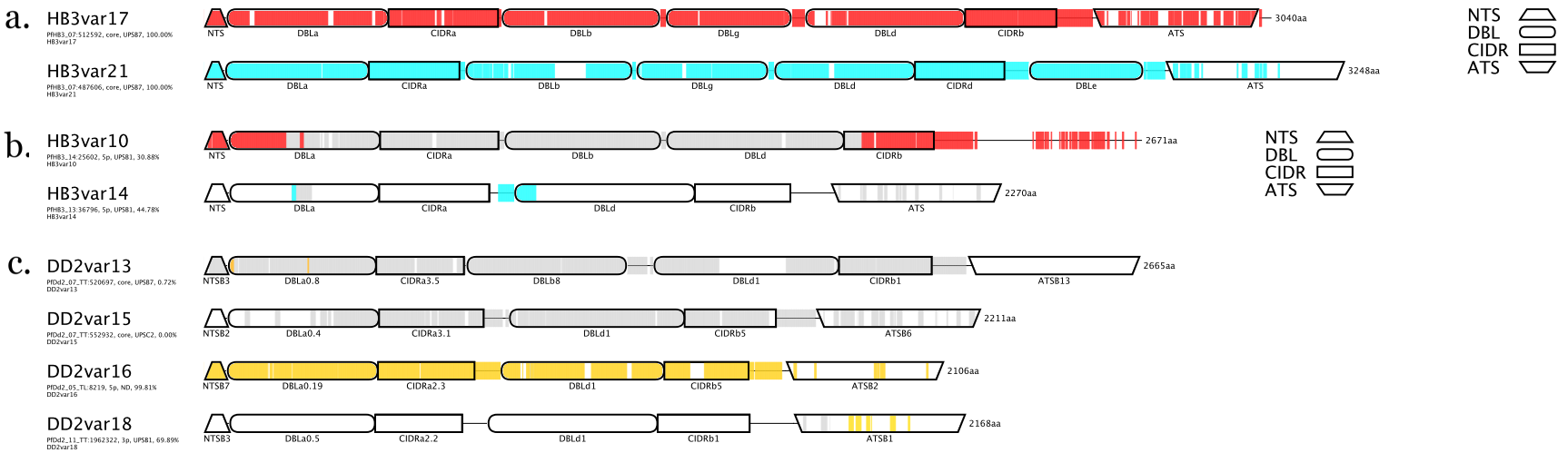


Figure 6.12: Recovery of diagnostic kmers in 3D7 and HB3 *var* genes. White regions indicate kmers that were not diagnostic. Vertical grey indicates a diagnostic kmer that absent in the progeny, while vertical colored lines indicate the kmer's presence. a. Two genes in PG0020-C purported to have an NAHR event in Claessens *et al.*, but exhibiting no kmer loss indicative of an NAHR event. b. Two genes in PG0034-C with missing diagnostic kmers, possibly indicative of an NAHR event. c. Two genes (DD2var13 and DD2var18) which exhibit very slight recovery or loss of diagnostic kmers towards their terminal ends.

6.5 Summary

Our graphical variant caller successfully identifies a wealth of *de novo* mutation in the *Plasmodium falciparum* crosses. The rates per sample for SNPs amount to less than roughly 2 SNPs per sample. Deletions are rarer than insertions, and both tend to arise as STR contractions or STR expansion/tandem duplications. NAHR events are exceedingly rare. Unforunately, we do not have dates for when these samples were placed in culture and number of generations that have elapsed, making per-generation rate estimates impossible to obtain until further information is furnished to us.

While Claessens *et al.* claimed evidence of such events involving exon 1 of core *var* genes, we do not find evidence for this in our graphical variant calls, nor do we see the tell-tale signs of piecemeal recovery in the kmer recovery plots like we saw between *PF3D7_0100100* and *PF3D7_0223500* in Chapter 1.

The widely varying number of mutations per sample between crosses is an interesting mystery. One intriguing possibility is that the variance reflects real biology. Southeast asian *P. falciparum* samples have recently been shown to be rapidly developing drug resistance. This is possibly due to mutations in genes involved in genome repair, preventing the parasite from correcting errors accumulated in the genome and thus contributing to the greater mutational load.[114] One way to determine this would be to study cultures of each parental parasite over time, removing aliquots from the primary culture periodically, performing limiting dilutions until new cultures are produced with just a single parasite, allowing these to grow to a sequenceable mass, and then examining the *de novo* mutation rate.

7 De novo mutations in Pan troglodytes verus

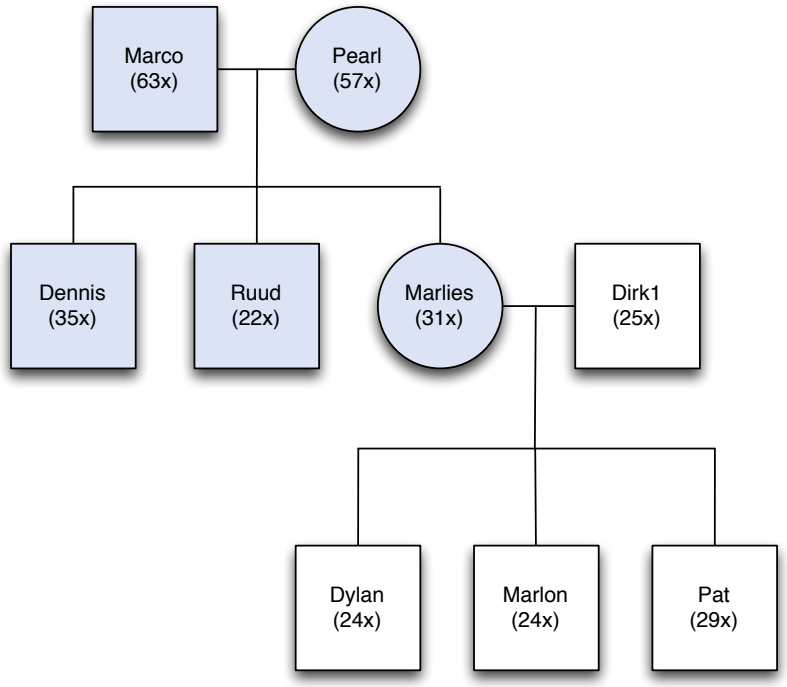


Figure 7.1: The three-generation chimpanzee pedigree. The highlighted quintet is discussed in this chapter. Average read coverage shown in parentheses.

7.1 Introduction

Finally, we briefly explore our software's scaling capabilities by touching on a second dataset: a quintet taken from a three-generation, nine member pedigree of West African chimpanzees (*Pan troglodytes verus*).[9] The processed samples are highlighted in Figure 7.1. Venn et al. sequenced and examined this pedigree using reference-based methods for *de novo* variation discovery, reporting a substantial paternal age effect to the overall mutation rate.

In many ways, the chimpanzee quintet is a more computationally challenging dataset to process than the *falciparum* crosses. First, the genome is roughly two orders of magni-

tude larger; *Plasmodium falciparum* is approximately 23 Mbp, while *Pan troglodytes* is 3,300 Mbp (3.3 Gbp). Second, the genome is diploid rather than haploid, an additional challenge during variant detection as now our software must contend with the presence of heterozygotes. Third, we do not have draft-quality parental assemblies, and owing to the genome size, it is prohibitively expensive to generate such data. We will be forced to rely solely on the Illumina data for the parents and children. Fourth, the chimpanzee reference genome is reported to be of substantially lesser quality than the *P. falciparum* reference, riddled with misassemblies and ambiguous sequence which can contribute to false negative and false positive variant calls.[115] The chimpanzee assembly process employed a whole-genome shotgun sequencing approach yielding $5x$ coverage and 37,846 supercontigs that were aligned to the human reference build to reconstruct the autosomes and sex chromosomes.[116] The *falciparum* employed a whole-chromosome shotgun approach yielding approximately $14x$ coverage per chromosome. With myriad gaps and errors in the chimpanzee reference sequence, a method unbiased towards the reference sequence should prove quite useful. Finally, the coverage is substantially lower: $57x$ on average for the founders, $28x$ on average for the children, or one-half and one-third the coverage we had for samples in the previous chapter.

For precisely these reasons, the chimpanzee dataset is an important test-bed for our *de novo* mutation calling software. It permits a demonstration of our capacity to process much larger genomes than the *falciparum* parasite. Moreover, the problematic chimpanzee reference sequence and infeasibility of generating draft reference assemblies for the parents or children allow us to demonstrate the value of an approach that can leverage both, but requires neither.

7.2 Data processing

7.2.1 Initial data

We obtained the raw genomic sequencing data for the nine members of the pedigree. For simplicity, we chose to focus our efforts on three children in the F_1 generation: Dennis, Ruud, and Marlies. Their parents, Marco and Pearl, were sequenced to higher coverage. This should give us greater confidence in our ability to distinguish novel kmers, kmers present in the child but absent from the parents, from those only absent from the parental genomes due to issues with sequencing. The quintet is summarized in Table 7.1. All samples were sequenced on Illumina HiSeq 2000 platforms using 100 bp reads.

Table 7.1: Summary of *P. troglodytes* quintet data

	Marco	Pearl	Dennis	Ruud	Marlies
Fragment size (bp)	369 ± 23	399 ± 31	377 ± 24	415 ± 27	383 ± 28
Coverage	63 ± 10	57 ± 8	35 ± 6	22 ± 3	31 ± 5
Cleaning threshold	10	9	8	4	6
Kmers					
dirty	7.7*e*9	9.0*e*9	5.9*e*9	4.4*e*9	5.2*e*9
clean	2.6*e*9	2.7*e*9	2.5*e*9	2.7*e*9	2.6*e*9
Novel kmers					
initial	-	-	126,536	348,374	121,834
final	-	-	33,535	30,794	20,923

7.2.2 *Sample processing*

Precisely as we did for the *P. falciparum* samples in the previous chapter, we used the available Illumina data to construct the de Bruijn graph structures for each child with McCortex, using a kmer size of 47 bp and discarding bases with a quality score less than *Q*5. McCortex's automatic cleaning algorithm was applied to the dirty graph for each sample. We did not apply the paired-end read threading and contig emission steps as contigs were not required for our analysis.

7.2.3 *Novel kmers*

We produced a list of novel kmers per child in the manner described in Chapter 4, selecting kmers present in the cleaned graph of the child but absent in the dirty graphs of the parents ("initial" in Table 7.1), then applying our filters described in Chapter 5 for coverage (set at $5x$ in this dataset), non-chimpanzee contamination[1], orphan removal and over-cleaning detection.

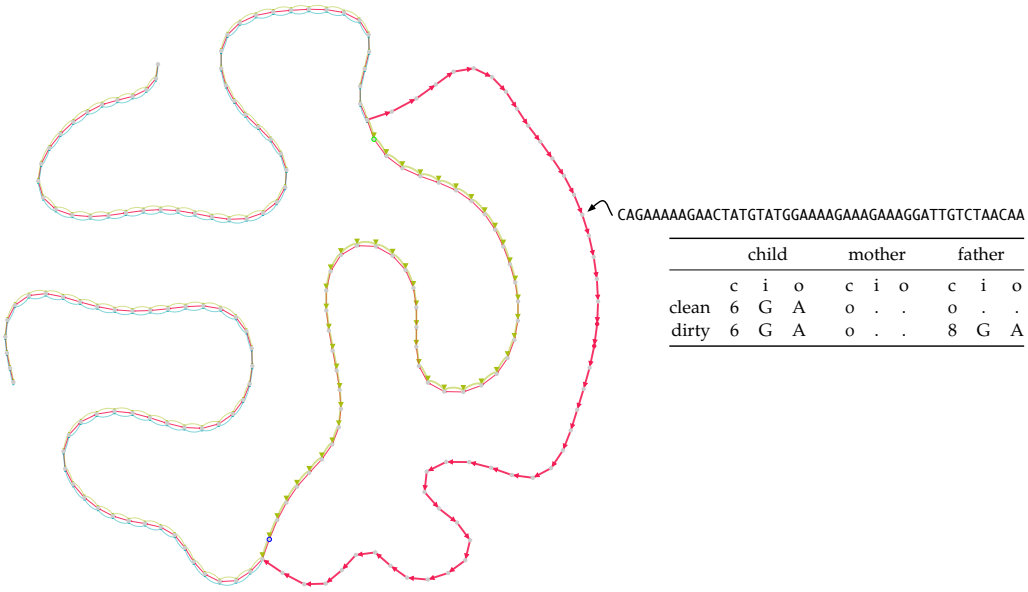
7.2.4 *Additional filters for low-coverage data*

Given the excessively large number of novel kmers in the final sets (an order of magnitude larger than what we might expect assuming 50 *de novo* SNPs per sample, each generating 47 novel kmers), we were concerned that the lower read depth on these samples could contribute to false positives *de novo* mutations. Lower coverage and the presence of heterozygotes (which will halve the coverage seen on each allele) should make it substantially

[1]Examining the results of the contamination check, it is evident that some samples contain small numbers of kmers that BLAST identifies as coming from human (hundreds) or gorilla (dozens). Given the evolutionary relationships between the three species, it is perhaps not unreasonable to allow these kmers into the analysis, as they may misattributed in the BLAST database. Instead, we have opted to remain conservative and exclude these from consideration.

more difficult for the McCortex software to automatically determine a suitable coverage threhold for kmers arising from sequencing error versus true genomic data. This may result in evidence against a putative mutation being discarded. Inspection of subgraphs associated with novel kmers revealed a number of kmers that had been erroneously and incompletely cleaned.

An example SNV found in the pedigree founders, Pearl and Marco, and F_1 progeny, Ruud, is shown in Figure 7.2. At first glance, this image seems to depict an A/C heterozygote on the haplotypic background of the mother. This is evident from the child's path (red) branching off from the mother and father (green and blue, respectively) at the top of the image and rejoining 47 kmers later towards the bottom of the figure. A child's path can also be seen tracking along the parental path between the two junctions, implying heterozygote status. However, two elements of this plot are suspicious. First, paternal edges are present outside the junctions, but absent within. Second, many of the kmers in the alternate branch are not novel (as indicated by the grey circles rather than red). Closer examination of each kmer in the alternate branch (we have highlighted one for clarity) reveals that while the parental edges G and A are present in the dirty graph, they are absent in the clean graph. By chance, two kmers had zero coverage in both the clean and dirty graphs, leading to the false-positive call.



	child			mother			father		
	c	i	o	c	i	o	c	i	o
clean	6	G	A	o	.	.	o	.	.
dirty	6	G	A	o	.	.	8	G	A

Figure 7.2: A putative and suspicious *de novo* SNP in child Ruud with very few novel kmers (filled red circles), and many non-novel kmers (filled grey circles) along the alternate branch. An example kmer is highlighted; its coverage (c), incoming edges (i), and outgoing edges (o) are shown in the inset table.

To guard against these false calls, we imposed two conditions. First, we required that

no kmer in the alternate branch have edges present in the parental colors of the dirty graph but absent in the clean graph. Effectively, the absense of an edge is only to be trusted if it is never observed in the raw data. Second, we required that a DNM contain at least 24 novel kmers. This cutoff is fairly arbitrary, chosen by simply considering that for graphs constructed at $k = 47$, this is $(k+1)/2 = 24$ kmers, or a little more than half of the kmers generated by a single SNP being novel. This ensures that our calls reach some reasonable quality threshold before we consider them.

7.3 *De novo mutations in the P. troglodytes dataset*

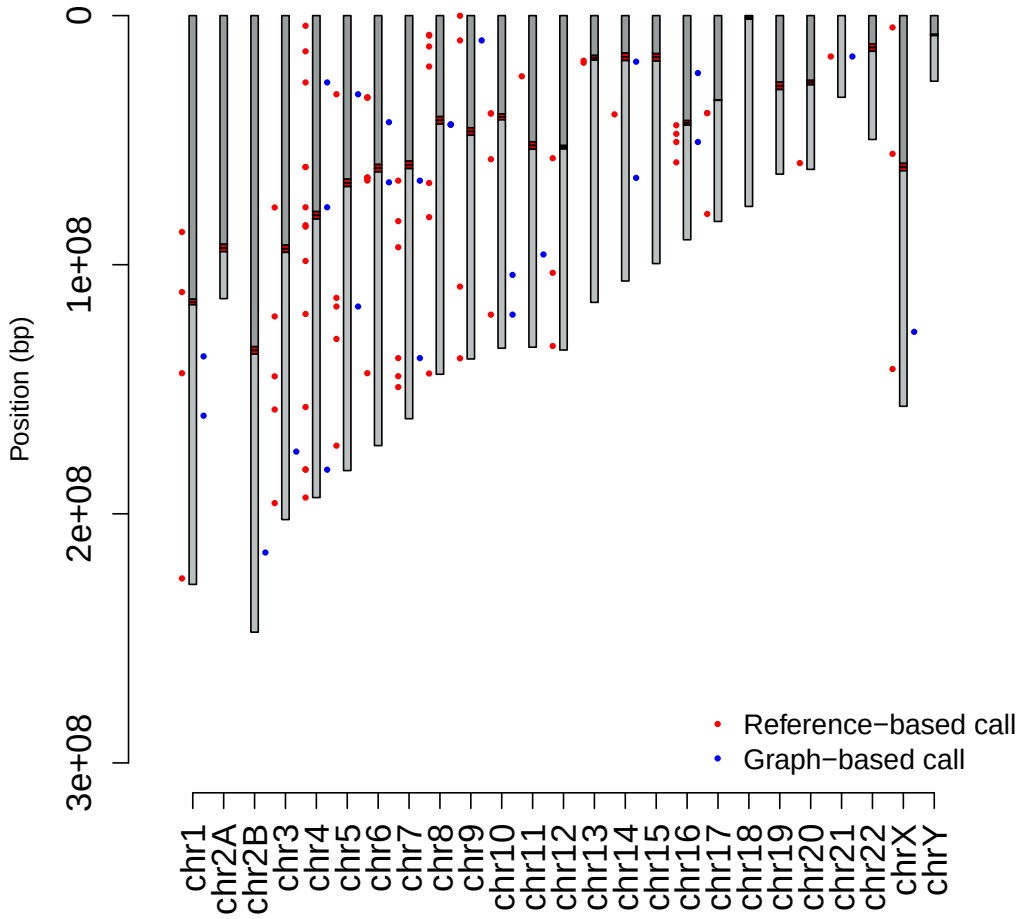


Figure 7.3: Positions of *de novo* mutations from the reference-based analysis (left of ideogram) and graph-based analysis (graph-based analysis).

We produced graph-based DNM calls using our software in three samples: Dennis, Ruud, and Marlies. We have listed all variant calls made in both the reference-based and graph-based analyses in Table 7.2, itself an expansion of a Table *S6* in the Supplementary Online Methods from Venn *et al.*, 2014.[9] Each row in the table lists a variant call made in the sample. Each call is either unique to the reference-based analysis (in which case the GraphCall column is "-"), or the graph-based analysis (in which case the RefBasedCall column is "-"), or present in both. Many of the reference-based calls were validated by Sequenom assays. Validation status is indicated as a three-part mask (Sequenom, visual validation in IGV, visual validation in VisualCortex). 1 indicates validation, 0 indicates failure to validate, and $-$ indicates not applicable. For brevity, we have only included the results for Ruud in this listing, as this sample had the largest number of validated sites.

We considered sites that passed validation and others that passed visual inspection of alignments in IGV to be true positives. The "Status" column indicates our assessment of a site's truth status. 31 sites were recovered in the graphical analysis. We classified 19 as false negatives - that is, present in the reference-based analysis and seemingly real, but missed by our caller. The positions of each DNM are shown in Figure 7.3, with variants from the reference-based analysis to the left of the ideogram and variants from the graph-based analysis to the right.

The reason for the missed variants is clear: in all cases, some (8) or all (11) of the novel kmers needed to seed the traversal were missing. Often, the novel kmers were found in the graph at coverage < 4, the automatically determined coverage threshold applied by McCortex. This problem of missing sites is much more pronounced in Marlies. While we recover 3 of the known and validated DNMs, we fail to recover 23; each one of these is missing a novel kmer somewhere along the variant branch, preventing traversal. All but two sites have novel kmers present in the dirty graph of the child at coverages less than the automatically determined cut-off of 6. Dennis is affected most profoundly; with a coverage threshold of 8, we recover 4 sites and miss 50.

7.4 Summary

Despite incomplete recovery of the known DNM repertoire, the results in Ruud, Marlies, and Dennis demonstrate that a reference-free, graph-based approach to DNM discovery does successfully find true *de novo* variation in a genome two orders of magnitude larger than it was designed for, in a diploid rather than haploid genome, and where coverage is substantially lower. Interestingly enough, the failure to re-discover many sites is not due to our algorithm, but cleaning thresholds automatically chosen by the McCortex assembly software to be too high, eliminating many kmers in variant branches in the graph and

compromising traversal. For this reason, we don't consider false positives to be the biggest problem here, as we do not tend to see a lot of spurious events in the dataset. Rather, our primary concern is sensitivity to DNMs in relatively low coverage data, especially when the presence of heterozygotes halves the coverage available to construct the graph and make the call. As we rely on external *de novo* assembly software, we are subject to some of its design limitations. While our goal is to use the assembly software's graph output to perform variant calling, the assembly software was originally written to emit contigs from the sequenced genome, taking great pains to eliminate any potential sequencing error so that the output would always reflect true genomic data, even if being conservative in the threshold choices resulting in some parts of the genome being omitted from the final assembly. Nearly every site we missed could have been recovered had the graph been cleaned at lower thresholds. A revision of the automatic cleaning code in the McCortex software by the author, or perhaps disabling the feature entirely, would likely resolve many of these problems. Then, Furthermore, one wonders if a probabilistic approach to graph construction (wherein kmers and/or edges are assigned quality scores) would enable their recovery while keeping graph traversal tractable. One also wonders if graphs need be cleaned at all if such a probabilistic construction were to exist.

Table 7.2: Variants found in Ruud in the reference- and graph-based analyses.

Chrom	Pos	NovelKmer	FoundInCleanGraph	FoundInDirtyGraph	WholePathPresent	RefBasedCall	GraphCall	Alleleo	Allele1	Validation	Status
chr1	2700783	GGCTTACGTCCGGGGCGGTCCAGCGTGCCCAGGAGCCGGCGAGCTCT	present	present	no	SNP	-	C	T	010	FN
chr1	130945113	CCACGCAAGTTAGAAGCATCTGATAGTGGAGATTTGAGGGTTGGGGC	missing	present	no	SNP	-	A	G	11-	FN
chr1	136785165	TGTTGATTCAGATGAGAGTTTCTATATCAAGGCAGCTTCCCTCTCAT	present	present	yes	SNP	SNP	C	A	111	TP
chr1	143564354	AAGATTCAGGACCTTGGGGAGAGCACTTATTAATGGAAGCTATAGAT	missing	present	no	SNP	-	T	C	11-	FN
chr1	160615945	ATGGTCTTTTCAAAGAATAGACATGCTTACTGTTCAATATAGGCAAA	present	present	yes	-	SNP	G	A	-11	TP
chr2A	12036323	AATGCAGTGGCACGGTCTCAGCTCAATGCAACCTCCACCTCCCGGGT	missing	present	no	SNP	-	C	A	11-	FN
chr2A	18269979	AAAAGATATGAAATTATTCATCTAGATAAGTTTCAGGACAGCCATCT	present;not_novel	present	no	SNP	-	G	A	110	FN
chr2A	27752205	CCTGATGAAAAGGAACTCCCTTCACCAGCAGATTTCCCCACTTCCTC	missing	present	no	SNP	-	T	C	110	FN
chr2B	127429405	GAGGAGGCTATGTGGTCAATGGAGTGTTATCTTAGTTCCATCTTACA	present	present	yes	SNP	SNP	T	G	111	TP
chr2B	154740624	ACTAGATTAGCTAGATACAGAGTGTTGATTGGTGTATTTACAAACTC	missing	missing	no	MNP	-	CC	TG	00-	FP
chr2B	154740632	AGCTAGATACAGAGTGTCCATTGGTATAATTACAAACTCTGAGCTAG	missing	missing	no	MNP	-	GTAT	ATAA	00-	FP
chr2B	155669031	TCTGATCTACACTCATCTGAAGAAATGAATACCAAAATGAAAAATTA	missing	present	no	SNP	-	C	T	11-	FN
chr2B	215527935	CATCTTTTTTTGCTTTTTTGTCAATAACTAAATATCAATTTAATCAA	present	present	yes	-	MNP	TGTTAGTG	AATTTAATCAAATTTAAAT	-11	TP
chr3	16093871	GGGAGGGCTGCACATGGCTTGAACTTGTAAGTTCAAGGCCAGCCTGG	present	present	yes	SNP	SNP	C	T	011	TP
chr3	158138958	TCTTGATCCACAATCAACGGTTTGGCTAGAAACATGAAGGATAACTT	present	present	no	SNP	-	A	C	110	FN
chr3	175043577	TAAGCTTCACCTACAGAAATTCAGTGAGACAGAAATAGGAATTCAGT	present	present	yes	SNP	SNP	A	G	110	TP
chr3	189709073	GTGCTTGTAGTCCCAGTTATTGAGGCGGCTGAGGTGGGAGGATTGCT	present	present	no	SNP	-	A	C	110	FN
chr4	14340935	TCAGTGCTAGAAAATGTTTGGGTGTTGACCAGCCTGAGTAACAAGTA	present	present	yes	SNP	SNP	C	T	111	TP
chr4	26839240	GTGTATGACTAGTGCTTTCTTTTATAATTCTATTTGTAAATGTTATG	present	present	yes	SNP	SNP	G	A	111	TP
chr4	76952784	AACTGTGGGTTTGCATGATTCATTCAAAAACATCATTTGGGGGTCGT	present	present	yes	SNP	SNP	G	A	111	TP
chr4	84732820	AATTTCCAGAGGAGGAAGAAAGAGACGGAATTCTAAGCACAGCGGGC	present	present	yes	SNP	SNP	T	C	111	TP
chr4	98504146	CTTCAAACAGCTAGGAATTAATCAGACAGCTTGGCTTCAAGCTCGGT	missing	present	no	SNP	-	C	A	00-	FP
chr4	119767460	TAAGCCACCACTCCTGGCCTTGGAACGATTTTAACAAGACAAATGTG	present	present	yes	SNP	SNP	T	C	011	TP
chr4	182152047	GGCATGATTTGGCAGCGGCTGGTACTGCTTGTTCCTTTCCATGTTTA	present	present	no	SNP	-	C	T	010	FN
chr4	182283692	CAACTTTCAGATATATTTGTTATACTGTTTTCACATAGCTCTATATT	present	present	yes	SNP	SNP	C	T	111	TP
chr5	30381378	TGTATTTACCCAATGCCTGTGCCCCTGTTGTAGACAAGAAGTAACTA	present	present	yes	SNP	SNP	C	T	011	TP
chr5	31583344	AGTGTTGGGGCAGGGGAGGGGGGAGGAAAAGGGAGTTTACAAAAAGA	present	present	yes	SNP	SNP	A	G	111	TP
chr5	113309277	GAAGTGAATCTGATAGATCAGAAAGGCCTGTATTAGTACGCCAGTCA	present	present	no	SNP	-	A	G	110	FN
chr5	116781586	GTTAAAAAAGCTTCAAATAATCATACTCAAAACCTCTTAACTTGCTG	present	present	yes	SNP	SNP	T	C	111	TP
chr5	129805166	TGAAAAGAGTCTTATTTCCCTTTCATTTATGAAGCTTAGTTTGGCTA	present	present	yes	SNP	SNP	C	T	111	TP
chr6	14157189	CCCACAAAAATAACTAGGGGATAATATGAGGGTGGACTTAGTTCTAG	present	present	yes	SNP	SNP	T	A	111	TP
chr6	42794716	TAAGCCCTGGCATGTCAAGTGTGCATGCATACACACACACCTGCTGC	present	present	yes	SNP	SNP	C	T	111	TP
chr6	66924441	ATGTTAGTAGCAGCAGCATTTCTTTGTTCTCAAAACATCACAAAAAC	present	present	yes	-	SNP	T	C	-11	TP
chr7	66260437	GCGTATTGCACGGGAATGTGTTTAACTGTCAGCAGAGGTGGCCTGGA	present	present	yes	SNP	SNP	G	C	111	TP
chr7	82509177	CATTTAACTTAGCAAAGAAAATCTATATAATTGAGACTAAAATACAT	missing	missing	no	SNP	-	G	T	00-	FP
chr7	137493791	ATAAAATAAAATAATAACAGAGTACATGTAACTATATATAGAAATAC	present	present	yes	SNP	SNP	G	A	011	TP
chr7	144765721	CCATGGTGGTTAATAGAGCTCATTAATACATCTAAGTATTACTAAAT	missing	present	no	SNP	-	C	A	11-	FN
chr8	43737875	AGAACGGTTTTGAAACACTCTTTTTGTAGAATGTGTAAGTGAAAACT	present	present	yes	-	SNP	A	G	-11	TP
chr8	43738841	AGAGTGTTTCAAAACTGCTGTATGAAAAGGAATGTTCAACGCTGTGA	present	present	yes	-	SNP	G	A	-11	TP
chr8	43738843	AGAGTGTTTCAAAACTGCTGTATGAAAAGGAATGTTCAACGCTGTGA	present	present	yes	-	SNP	T	G	-11	TP
chr9	9965284	TTGGGAACAAAAAAGAGAGACTAGATGTCAGTATTAATTATAAAACT	present	present	yes	SNP	SNP	C	T	111	TP
chr9	110484253	CCAACCAACTTGCTGTATACACCTTAGACAAGTCATTTGACCTTTCA	present	present	yes	SNP	SNP	G	A	111	TP
chr10	57652821	TTCAAGGTCATCATATGAGACTAGCCTTATTATATCTATTTCACCTA	present	present	no	SNP	-	A	C	110	FN
chr10	80306438	TGTTGACCGGAAGCCTTACCAATAATATATAGTCGATTAACACATAT	present	present	yes	SNP	SNP	C	T	111	TP
chr10	104106966	ATGCCTCTAATTGTTTATATTTATAAATATACTTATCAATACATTTA	present	present	yes	-	SNP	A	T	-11	TP
chr10	120057414	TCAGTCACATAAAAATGTAGAGCAGACTGTTTGCAGTCTCCTTTAGG	present	present	yes	SNP	SNP	G	A	111	TP
chr11	24359728	TGAAATATCCATCTACAACTGTATTTGTTTTTGTTATTTTAACGGTC	missing	missing	no	SNP	-	C	T	11-	FN
chr11	95890315	ATGTCCTACAATCACATAGATTTTGCAATATAAACTTATATTGAATC	present	present	yes	SNP	SNP	G	A	-11	TP
chr13	95270772	GCCCTGCAGCATGCCATTTTCTGGGCGTACAGAGACGAATATAAAAT	missing	present	no	SNP	-	G	C	11-	FN
chr14	18496753	ACAGATAAATAAATTTTAGAGACTGAAGAACCCACTTACATTCTCAA	present	present	yes	-	SNP	A	G	-11	TP
chr14	39651670	ACACTCTAAGCATTAGACTTTGCATTGATAATATATATAAAATATAT	present	present	no	SNP	-	C	T	010	FN
chr14	65139282	ACCACACTGCCTCCAAGGGAACTGATGCATACATTGCAGGCTTTGTG	present	present	yes	SNP	SNP	C	T	111	TP
chr15	91802628	CCTTTCTTGGAGAAGGAAAAGTGTTTTCACAGTCCAGCTATAAGATA	missing	missing	no	SNP	-	C	T	11-	FN
chr16	23062470	CCTTCTCATAGGTATGCTTTATGTTTCTGTAGAATTTTGGGATGAAG	present	present	yes	-	SNP	C	T	-11	TP
chr16	50735389	AATGAGCCTCAGCACGCAGCAGCCCTGACATCAAACGCACTTGTGTT	present	present	yes	SNP	SNP	C	T	111	TP
chr19	3715608	GAGCCTTCCCTGAAATACTGCTCAGAGTCAAAACCGGGATCCCCGTT	missing	present	no	SNP	-	G	A	11-	FN
chr21	16410630	CAATGGATAGCATTTCATTAGATGGGAATGTGGTTCAAAGCTACCCA	present	present	yes	-	SNP	G	T	-11	TP
chrX	9149290	CAAATATCCATCCAACTGAATACTCTCAAGGGATAGAAAGAAAAGAA	missing	present	no	SNP	-	C	T	01-	FN
chrX	126947832	GTCTAAGGAAGTCAGGCGCCAATTTCAATCATAAACATTTAAGAAAT	present	present	yes	SNP	SNP	T	C	011	TP

8 Discussion

8.1 Introduction

In this dissertation, we have demonstrated a graphical method for the detection of *de novo* mutations. We have attempted to address six major concerns regarding detection of DNMs in genomes. First, false-positive rates can be difficult to control, as DNMs are often singletons in the population and in many circumstances may be confused for sequencing error. Second, it is not typically known how much *de novo* variation is present in a sample, and thus difficult to ascertain whether all present variation has been captured by variant calling analyses or if some has escaped detection. Without whole-genome assemblies of all samples (a prohibitively expensive prospect), such sensitivity measures cannot be produced. Third, reference-based analyses cannot provide access to accessory regions of the genome - regions that are clinically interesting and often highly divergent between different members of the population. Fourth, a new method should scale beyond the genomes of small parasites, all the way to sizes typical of mammalian genomes and beyond. Fifth, in keeping with the need to scale beyond parasite genomes, it must also be capable of handling non-haploid data. Finally, the method must be robust to low read coverage. All six elements are addressed by the implementation and application of our graphical variant caller, designed to operate without the need for a reference sequence and on the novel kmers produced by the mutational processes that generate *de novo* mutations.

8.2 Reference versus graph

8.2.1 Indirect comparison: reference-based analysis

Canonical reference sequences have been tremendously beneficial for analyses of genetic variation in sequenced samples. Reference genomes are expensive and laborious to produce, requiring the use of sample preparation and sequencing strategies that do not scale to many samples (hundreds, if not thousands). Aligning short, lower-quality reads from inexpensive second-generation sequencing technologies to a reference genome enables

the comparison of many samples. Nearly all general purpose variant-calling software packages are implemented as reference-based methods, reliant on pileups of short reads aligned to correct locations in the reference in order to discover a variant.

However, given the limitations of current technology, this procedure has a tacit reference bias. Second-generation sequencing reads tend to be short ($75 - 150$ bp or so), and alignment algorithms require a significant amount of each read to align to the reference without error (mismatches, insertions, or deletions). If the region in question has low diversity in the population, this protocol works very well. If the sampled haplotypes deviate substantially from the reference haplotype, two types of error may occur. First, reads may fail to map correctly, leading to false-positive calls. Alternatively, reads may fail to map at all, leading to false negative calls.

Thus, one find themselves in an ironic position for the discovery of *de novo* mutations: despite having data for the parental and child haplotypes, discovering mutations in the child with respect to its closest relatives first requires an *indirect* analysis: a reference-mediated comparison, where the reference is usually unrelated to the samples in question. If the reference haplotype is substantially different than the sampled haplotypes, one is likely to encounter mistakes that compromise the accuracy of the callset.

Aligning reads to a reference genome and calling variants is essentially taking a reference genome to be an initial hypothesis of the genome sequence, then refining that hypothesis with the variant calls. However, the ability to refine that hypothesis is dependent on the properties of the data used to refine it. With second-generation sequencing, our reads are too short to refine this initial hypothesis - the reference genome - when the true genome deviates substantially from it. As we have seen in *P. falciparum*, the initial hypothesis serves the core regions of the genome well, but the accessory regions very poorly, as these regions harbor antigenic genes and are thus under intense selection by immune systems. In any species, regions under such diversifying selective pressure will typically be a struggle to treat in reference-based studies.

8.2.2 Direct comparision: graph-based analysis

Our work demonstrates an alternative approach: a hypothesis-free method based on *de novo* assembly of genomes. Constructing de Bruijn graphs for each member of a trio (mother, father, and child) is reasonably straightforward, and combining them into a multi-color de Bruijn graph facilitates their comparison. Given that our samples are deeply covered (recall the Lander-Waterman statistics presented in 1.2), we expect the entire genome to be represented in these graphs. Thus, we remove the need for a canonical reference sequence to mediate the comparison, instead relying on the samples themselves

for the variant discovery process. This graphical approach is a more direct framework for DNM discovery as we are comparing the child's haplotypes to the haplotypes from which they are immediately derived, without the need to first compare them to an unrelated reference haplotype.

8.3 Implementation notes

8.3.1 Merits of the graphical approach

In implementing a graphical approach to DNM discovery, many merits to the approach over read alignment and contig alignment become apparent. First, discovering where a novel kmer exists in a graph is very straightforward. A graph can be thought of as a hashtable, in which case lookup time is $\mathcal{O}(1)$, but the entire graph needs to be stored in memory. Our sorted-graph-on-disk implementation, wherein we perform a binary search over the sorted kmers in the graph, avoids the need for storing the entire graph in memory and has average lookup time of $\mathcal{O}(\log n)$, which is still very fast. Additionally, as *de novo* mutations are rare, the number of kmers in the graph that need be accessed is much smaller than the total number of kmers in the graph. In contrast, it is less straightforward to search for novel kmers in the reads and contigs. Without an index and without storing every read or contig in memory, one must resort to more primitive methods. For example, one could find all reads and contigs with an $\mathcal{O}(nm)$ lookup: a linear traversal in which n reads or contigs are examined, all m kmers in the read are examined for presence in a novel kmer hash, and the read or contig is stored for later use if the kmer is present in the hash.

Smarter methods to identify all reads or contigs containing novel kmers certainly exist (e.g. aligning novel kmers to the reads or contigs themselves), but we quickly run into the second benefit of the graphical method over others: the de Bruijn graph imposes a restriction that kmers may only appear once in the data structure. Thus, finding the kmer yields a single region of the graph to examine. If implemented as a lookup on reads or contigs, we would discover many reads with many potential homes around the genome (some correct, some incorrect). At best, each novel kmer could identify "active regions" around the genome - windows on the genome that may harbor DNMs. However, other reads harboring the novel kmer and a mismatch later in the read sequence might align to a different region of the genome, yielding two active regions and no convenient way of combining them or eliminating regions from consideration.

Third, the novel kmers track progress in discovering DNMs in a way that only makes sense for the graphical method. As we have shown, DNMs typically generate runs of

novel kmers rather than isolated novel kmers. These novel kmers span the variant in question. In the ideal case where there are no sequencing errors, the reference-based approach could yield novel kmers that are all aligned to successive positions, easily revealing the narrow window in which the variant occurs. However, in the non-ideal case, misaligned reads may not reveal the active region so precisely.

Fourth, true-but-polymorphic DNMs are much more easily identified in the graphical method. As mentioned in the previous chapter, these mutations can arise in *P. falciparum* during culture. They are not false-positives per se, but depending on the researcher's question, they may or may not want to be considered as part of the DNM callset. It is convenient to ignore such sites in the reference-based analysis of haploid samples; doing so focuses one's effort on monomorphic sites which are less likely to suffer from false-positives owing to the amount of evidence required to identify them. If working with contigs rather than the graph, the assembler's attempt to linearize contigs by popping low-coverage branches of bubbles may erase evidence of a variant. In diploid samples, such filters are hugely problematic. Most DNMs would be heterozygous, and ignoring them is not an option. With the reference-based analysis, the variant calling protocols must be different depending on whether one is working with haploid or diploid samples. For our graphical approach, they are one and the same: if a novel kmer associates with a bubble motif, it is considered *de novo*. The genotype of the variant is not considered until after the variant is identified, and is not used as a filter.

Finally, our random-access approach to graph traversal allows us to process genomes of any size without needing to load an entire graph into memory. As *de novo* variants are rare, the number of vertices we need to visit in the graph is far lower that the number of vertices in the graph. This enables our method to scale to much larger (e.g. mammalian) genomes. Analysis of the *Pan troglodytes verus* pedigree in Chapter 7, wherein each sample required < 8 hours of processing time, directly demonstrates this point. Low coverage ($\approx 20x$) in the $F1$ generation of this dataset, along with the need to detect heterozygotes in diploid data, poses additional challenges to achieving callset specificity. However, with additional filters designed to reexamine dirty data for evidence that refutes DNM calls, we were able to produce a DNM callset comparable (if not more sensitive and specific) than the reference-based analysis.

8.3.2 Challenges of the graphical approach

There are several challenges that must be met in developing a graphical approach for DNM discovery. First and foremost, very little shared infrastructure around genome graphs exists. Reference-based analyses benefit from standardized file formats (FASTA,

BAM, VCF, BED, GFF, etc.). The GATK, Platypus, Samtools, Picard, SOAPsnp, and many others, provide dozens of data manipulation tools for next-generation sequencing data. The htslib C library and the htsjdk Java library enable rapid and random access to compressed sequence data formats. While many tools for *de novo* assembly exist, there is no standard file format for describing the underlying graph data structure, and there is no readily available library for interrogating the file formats that do exist. Assemblers are generally written such that the user is only ever intended to see the final contig output; the particulars of the graph are private to the software, not for user consumption.

Very little software exists for visualizing graphs so that an analyst can inspect various aspects of a graph to convince themselves of the credibility of a call or callset. Reference-based methods benefit from the existence of several visualization tools, all free for public use (IGV, Tablet[90]). Visualization tools for genome graphs are few and far between, require very specific file formats, can be incredibly demanding of memory resources, do not understand multiple samples, and typically only display graphs at the contig level, rather than the kmer level.[117,118] General-purpose graph visualization algorithms can be used to visualize genome graphs at the kmer level, but result in static layouts that attempt to maximize some arbitrary aesthetic properties (e.g. preserve node hierarchy, minimize edge crossings, avoid sharp bends in edges, keep edges short, maximize symmetry, etc.[119]).

Graphs are far more susceptible to data contamination issues than alignment-based methods. With alignment, the reference genome acts as a sort of sieve, retaining only reads that are similar in sequence to some region of the reference. The same failing that prevents us from accessing the subtelomeric regions of the genome protects us from processing sequence contamination as if it were part of the genome. In the graphical method, this protection does not exist; all kmers are added to the graph, no matter the source. If a contaminant happens to share a single low-complexity kmer with the samples in question, the genomes become linked through the graph. As we have seen, tremendous effort must be applied to mitigate issues involving data contamination.

Localizing variants can be problematic. Although our method does not rely on the presence of a reference genome, for post-calling analysis, it is of course desirable to place the variant in its genomic context. Determining this location can be tricky. A lookup table for every kmer in the reference genome and all of its possible homes is helpful, but often a kmer (47 bp in our case) is still rather short and aligns ambiguously. Instead of aligning single kmers, we can align the parental branch spanning a bubble rather than the child's branch. This reduces the amount of variation that the aligner needs to contend with. However, errors or other variants nearby can prematurely truncate the contig, and the

truncated contigs may not align as readily. Thus, the graphical approach can sometimes leave us with variants whose location is unclear.

Determining the haplotype background upon which a variant has occurred is not always straightforward. The background for a variant situated in a genomic region where one parent is exceedingly different than the other's is easy to determine. For a variant that occurs in a region that is perfectly homologous between the two parents, inspection of the graph is less revealing. For a variant that occurs near another variant (perhaps a polymorphic variant), the event may appear on one background when it actually occurs on the other. In theory, one should be able to walk the graph in either direction of a DNM until one sees inherited variation that tags one background or the other. In practice, however, the ability to traverse past errors and repetitive regions is limited with such short reads, and its unlikely that one can establish enough genomic context to determine the background accurately and consistently.

8.3.3 Addressing the implementation challenges

On the issue of reusable software, we have provided tools for manipulating graphs and an object-oriented API for accessing the contents of the Cortex binary graph format so that other developers can do the same. Our toolkit, INDIANA, is a framework for graph manipulation, graph-based variant calling, and visualization. The framework is written in Java, ensuring that it can be run on any platform. Embedded within it is our `CortexGraph` API is capable of iterating through the graph one record at a time, or (if the graph is sorted), providing random access to the records within the file. We employ memory-mapped access to the file contents using the Java NIO (new I/O) API to achieve high-performance reads and writes. The library can be compiled as a single Java archive (.jar) file so that it can be included some of the most popular Java-based toolkits for manipulating sequence data, including the GATK, Picard, and IGV.

To address the problem of visualization, we wrote `VisualCortex`. `VisualCortex` is a lightweight web-based visualizer for Cortex graphs, built on top of our aforementioned Cortex API and the force-directed graph layout engine provided by the d3.js JavaScript visualization library.[120] VisualCortex makes use of the JDK's built-in `HttpServer` object to expose a RESTful API for accessing graph data from a web browser, allowing us to fetch and display data without the need for page reloads after every request. Our client provides a search interface for a user to specify a single kmer in the graph, which the server uses to fetch the local subgraph and associated metadata. The traversals applied to the child and parent colors are precisely the same as used during variant calling, so the server can be used to evaluate the caller's decision-making process on traversing or

ignoring branches in the graph. Vertices and edges are drawn as scalable vector graphics (SVG) elements for later lossless export to PDF. Child edges are drawn as straight lines, while parental edges are drawn as curves with opposite bends so that all three edges can be seen connecting two adjacent vertices simultaneously. Edges involved in a variant call are drawn with thicker lines so as to be more visually apparent. Vertices are color-coded to reveal novelty (red) or lack thereof (grey), and shape-coded to reveal contaminants (crosses) or normal data (circles). Vertices can be repositioned by drag-and-drop gestures, permitting the user to choose a more aesthetically pleasing layout and/or one that calls attention to particular details of the graph. Low-level graph data (i.e. the kmer sequence, coverage, and edge data from the underlying Cortex graph, and the kmer's location in the parental reference genomes if available) are displayed on vertex mouse-over events. To enable exploration, double-clicking a vertex triggers an additional traversal which continues graph navigation for a short distance (50 vertices, or more than one junction, or until there are no more edges to explore - whichever comes first). In theory, because our implementation follows the model-view-controller (MVC) design paradigm wherein the external representation of information is wholly separate from the internal representation, any graphical file format could be accomodated so long as it implemented the CortexGraph API.

Our decontamination procedure also makes use of our customizable traversal engine to find and mark likely contaminants so they can be ignored by the caller downstream. We note that our procedure is implemented on the graph rather than the reads, despite it theoretically being easier to remove contaminating reads before graph construction than to remove kmers from the graph after the fact. Our more complicated procedure is motivated by the pathological case that in two overlapping reads from a contaminant, only the first may have a kmer in the BLAST database. If we had only removed reads with possible contaminants, the overlapping reads would remain in the graph and would still have a number of novel kmers that confuses our analysis. Furthermore, removing the first read with contaminating kmers would discard the signal later needed to remove the second.

To map variants back to a canonical home in the genome, we employ multiple methods. We keep an index for each parental genome specifying the location(s) of every kmer in the genome (a sorted list of a kmer and its location, with multiple records for the same kmer if it has multiple homes, all conveniently implemented via the Java-based MapDB library for on-disk storage of typical `java.util` collections[1]). We've also implemented access to two external aligners, BWA and LASTZ, both of which return records in SAM

[1] `http://www.mapdb.org/`

format and latter of which returns better results when aligning long contigs. The index allows us to quickly determine if a contig found in the child might be chimeric, while the external aligners are robust to sequencing errors and allow us to place variants in their genomic context.

Finally, unable to rely on graph traversals to consistently reveal the haplotypic background of an event, we instead used haplotype transmission tracks generated by the reference-based method. These tracks are produced by simply calling variants in mother, father, and child along the reference sequence (ignoring the masked regions from the MalariaGen project), and for variants not shared by both parents, determining the parent of origin. Successive variants with the same parent of origin are merged into large intervals and added to an interval tree, a tree data structure that allows one to find intervals contained by or overlapping with other intervals in $\mathcal{O}(\log n + m)$ time. If a background determination cannot be made using the graph traversal, we use the localization procedure discussed above to determine a locus in the reference sequence and an interval tree lookup to determine the background.

8.3.4 Novel kmers and their effect on design considerations

The novel kmers concept is important to the design of our calling software. In our implementation, they serve three functions: as beacons (where to look), as signposts (which direction to walk), and as mile markers (how much further to go).

As beacons, novel kmers indicate regions in the graph to examine for DNM activity, allowing us to process only the relevant parts of the graph rather than exploring every possible location. Our code is written to iterate through the set of novel kmers and attempt to call a variant at each one. We recognize that not all novel kmers tag *unique* variants; many are found adjacent to one another as part of the same mutation. Thus, we keep track of every novel kmer we jump to and every novel kmer we see during a variant-calling operation so as not to perform redundant work. Because of this, any future improvements pursuing parallelization would have to be carefully considered. Simply parallelizing the variant-calling operations by splitting up the list of novel kmers among processing threads would lead to a lot of redundant effort. Instead, a better strategy might be to identify novel contigs associated with novel kmers, then parallelize on the contigs.

As signposts, novel kmers allow us to continue graph traversals that would otherwise be truncated due to an inability to navigate junctions at errors. Here it is important to understand that we have made two explicit assumptions: adjacent novel kmers belong to the same variant, and *de novo* mutations are rare. The former assumption allows us

to choose a reasonable branch when we encounter a junction with novel kmers on one output and not the other. The latter assumption allows us to act on the first assumption without fear that we will accidentally traverse to an irrelevant region of the genome and output a nonsensical variant call.

Finally, as mile markers, novel kmers inform us as to how much of the graph remains to be explored. While many other variant callers certainly have progress meters, they are typically measuring how many bases remain in the genome to be processed. Our accounting is not simply a matter of tracking progress, but rather about reaching full sensitivity. Novel kmers do not simply tell us where to look, they tell us when to stop looking.

There is an argument to be made that the amount of effort we expend to eliminate errors and contamination from the initial list of novel kmers is wasted. After all, we have not observed traversals seeded from erroneous novel kmers resulting in false positive bubble-motif events, and mistaken linear-motif events are rare. Rather, the false positives are more often unclassifiable events. If there is little danger in flooding our callset with errors by increasing the list of novel kmers that we inspect, why try so hard to narrow that list? Our answer is that neglecting to filter the novel kmer list would compromise its usefulness as a sensitivity measure. Removing errors and contaminating kmers effectively provides an explanation for how those kmers came to be part of our dataset in the first place. Had we not removed them, we might not have necessarily compromised the integrity of the final callset, but we would almost certainly be left wondering how so many unclassified events had come to be part of our data.

8.4 Further improvements

8.4.1 Towards a probabilistic treatment of assembly graphs

One interesting aspect of our approach is that it is the beginning of a probabilistic treatment of assembly graphs. Assembly graphs are generally treated in a very binary way: a kmer is either an error or not; a contig either belongs in the graph or doesn't; navigation proceeds until it hits a junction, and makes no effort to traverse the boundary. In contrast, we make a number of decisions that subvert this behavior. Although we rely on McCortex and its error-cleaning algorithm for the actual construction of the de Bruijn graph data structure, we have implemented our code in such a way to allow for a kmer to be retreived from the dirty graph if it allows us to complete a traversal that otherwise could not occur. In doing so, we recognize the fact that no error cleaning algorithm can ever be completely correct. This is akin to having a quality score attached to each kmer (albeit

one without much dynamic range). Although many kmers might deserve a low quality score indicating their likelihood of being an error, some kmers clearly represent real data despite appearing erroneous at first glance. We've implemented our system such that a dirty kmer can be incorporated into the traversal if a clean kmer is absent, but one could imagine a more probablistic treatment.

In fact, the traversal algorithm is already set up to permit a probabilistic treatment. Dijkstra's shortest path algorithm attempts to find the shortest weight path, not necessarily the shortest number of kmers. If all the edge weights are identical, these are one and the same. However, weights could be easily incorporated into the algorithm, with higher weights indicating a lower quality. If multiple paths from source to destination exist, the path with fewer dirty kmers will likely be chosen. However, if the path utilizing the dirty kmers is the only path that can be found, they'll be used. Although we intended to implement this functionality in our own code, we had difficulty finding clear examples in simulation where such traversal rules proved critical to recovering variants. Thus we simply stuck with the ad-hoc incorporation of dirty kmers when necessary. In general, the mutation rate of *P. falciparum* appears to be too low and the read coverage too high to make adequate use of this feature. In lower coverage data and/or data with a higher mutation rate, it may very well be a useful feature.

8.4.2 Removing graphical tangles

As we have seen, filters on the graph are necessary to remove artifactual novel kmers. One graphical motif that we did not develop a filter for is the so-called "tangle", a bundle of low-complexity kmers with promiscuous connectivity. Novel kmers likely arise in these regions as the result of recurrent sequencing errors promoted by low complxity sequence. We speculate that these tangles can be treated as a "community detection" problem. Formally, community detection aims to partition graphs such that each subset has many more edges connecting members within groups than between groups.[121] A popular algorithm that attempts to optimize a modularity score (a scalar measure of intra-community vs inter-community links) could likely be applied.[122]

8.4.3 Detecting inversions

Our work did not reveal any inversions in the *P. falciparum* or chimpanzee datasets. However, we note that our detection apparatus is currently only set up for small events (substantially shorter than a read length) that look no different than typical bubble-motif events. This was a conscious decision, as there is Large inversions (substantially longer than a read length) will not necessarily exhibit this pattern. Instead, we would expect

to see novel kmers at the breakpoints, and inherited kmers between them with edges running in opposing directions between child and parent along the inverted segment.[123] Detecting inversions is a desirable future feature of our graphical variant caller.

8.4.4 Other miscellaneous improvements

Other than being used during graph construction, cleaning, and novel kmer filtering, coverage is not inspected in the execution of our variant-calling process. While we are able to detect insertions and deletions, we have made no attempt to detect copy number changes. Coverage across the parental versus alternate branch can inform on this point. In addition, potential recurrent sequencing artifacts in diploid data could be removed by requiring roughly equal coverage on each branch (though we note that, in theory, McCortex's graph cleaning algorithm should already make similar considerations).

8.5 Conclusion

We have shown our graphical variant calling method works well with simulated data (both idealized and realistic), real data from small genomes, and real data from large genomes. Our approach enables datasets to be processed in a reference-free manner, eliminating the inherent reference bias of such approaches. We thus have tremendous flexibility in how we search for *de novo* mutations in terms of the types of variants we search for, dataset size, optional reference genome availability, and reference genome quality. With the advent of third-generation sequencing, our graphical variant calling method should enable the study of genome biology in an era where long-read sequencing platforms are used to generate a draft-quality assembly for use as merely a coordinate system, while cheaper short-read sequencing platforms are used for refinements of the genome.

References

[1] Thomas S Rask, Daniel A Hansen, Thor G Theander, Anders Gorm Pedersen, and Thomas Lavstsen. Plasmodium falciparum Erythrocyte Membrane Protein 1 Diversity in Seven Genomes – Divide and Conquer. *PLoS computational biology*, 6(9):e1000933, September 2010.

[2] C Wang, K Takeuchi, L H Pinto, and R A Lamb. Ion channel activity of influenza A virus M2 protein: characterization of the amantadine block. *Journal of virology*, 67(9):5585–5594, September 1993.

[3] Martha I Nelson, Lone Simonsen, Cécile Viboud, Mark A Miller, and Edward C Holmes. The origin and global emergence of adamantane resistant A/H3N2 influenza viruses. *Virology*, 388(2):270–278, June 2009.

[4] Vegard Eldholm, Gunnstein Norheim, Bent von der Lippe, Wibeke Kinander, Ulf R Dahle, Dominique A Caugant, Turid Mannsåker, Anne T Mengshoel, Anne M Dyrhol-Riise, and Francois Balloux. Evolution of extensively drug-resistant Mycobacterium tuberculosis from a susceptible ancestor in a single patient. *Genome Biology*, 15(11):490, November 2014.

[5] Matthew T G Holden, Edward J Feil, Jodi A Lindsay, Sharon J Peacock, Nicholas P J Day, Mark C Enright, Tim J Foster, Catrin E Moore, Laurence Hurst, Rebecca Atkin, Andrew Barron, Nathalie Bason, Stephen D Bentley, Carol Chillingworth, Tracey Chillingworth, Carol Churcher, Louise Clark, Craig Corton, Ann Cronin, Jon Doggett, Linda Dowd, Theresa Feltwell, Zahra Hance, Barbara Harris, Heidi Hauser, Simon Holroyd, Kay Jagels, Keith D James, Nicola Lennard, Alexandra Line, Rebecca Mayes, Sharon Moule, Karen Mungall, Douglas Ormond, Michael A Quail, Ester Rabbinowitsch, Kim Rutherford, Mandy Sanders, Sarah Sharp, Mark Simmonds, Kim Stevens, Sally Whitehead, Bart G Barrell, Brian G Spratt, and Julian Parkhill. Complete genomes of two clinical Staphylococcus aureus strains: evidence for the rapid evolution of virulence and drug resistance. *Proceedings of the National Academy of Sciences of the United States of America*, 101(26):9786–9791, June 2004.

[6] Donald F Conrad, Jonathan E M Keebler, Mark A DePristo, Sarah J Lindsay, Yujun Zhang, Ferran Casals, Youssef Idaghdour, Chris L Hartl, Carlos Torroja, Kiran V Garimella, Martine Zilversmit, Reed Cartwright, Guy A Rouleau, Mark Daly, Eric A Stone, Matthew E Hurles, Philip Awadalla, and 1000 Genomes Project. Variation in genome-wide mutation rates within and between human families. *Nature genetics*, 43(7):712–714, July 2011.

[7] Augustine Kong, Michael L Frigge, Gisli Masson, Soren Besenbacher, Patrick Sulem, Gisli Magnusson, Sigurjon A Gudjonsson, Asgeir Sigurdsson, Aslaug Jonasdottir, Adalbjorg Jonasdottir, Wendy S W Wong, Gunnar Sigurdsson, G Bragi Walters, Stacy Steinberg, Hannes Helgason, Gudmar Thorleifsson, Daniel F Gudbjartsson, Agnar Helgason, Olafur Th Magnusson, Unnur Thorsteinsdottir, and Kari Stefansson. Rate of de novo mutations and the importance of father's age to disease risk. *Nature*, 488(7412):471–475, August 2012.

[8] Benjamin M Neale, Yan Kou, Li Liu, Avi Ma'ayan, Kaitlin E Samocha, Aniko Sabo, Chiao-Feng Lin, Christine Stevens, Li-San Wang, Vladimir Makarov, Paz Polak, Seungtai Yoon, Jared Maguire, Emily L Crawford, Nicholas G Campbell, Evan T Geller, Otto Valladares, Chad Schafer, Han Liu, Tuo Zhao, Guiqing Cai, Jayon Lihm, Ruth Dannenfelser, Omar Jabado, Zuleyma Peralta, Uma Nagaswamy, Donna Muzny, Jeffrey G Reid, Irene Newsham, Yuanqing Wu, Lora Lewis, Yi Han, Benjamin F Voight, Elaine Lim, Elizabeth Rossin, Andrew Kirby, Jason Flannick, Menachem Fromer, Khalid Shakir, Tim Fennell, Kiran Garimella, Eric Banks, Ryan Poplin, Stacey Gabriel, Mark Depristo, Jack R Wimbish, Braden E Boone, Shawn E Levy, Catalina Betancur, Shamil Sunyaev, Eric Boerwinkle, Joseph D Buxbaum, Edwin H Cook, Bernie Devlin, Richard A Gibbs, Kathryn Roeder, Gerard D Schellenberg, James S Sutcliffe, and Mark J Daly. Patterns and rates of exonic de novo mutations in autism spectrum disorders. 485(7397):242–245, May 2012.

[9] Oliver Venn, Isaac Turner, Iain Mathieson, Natasja de Groot, Ronald Bontrop, and Gil McVean. Strong male bias drives germline mutation in chimpanzees. 344(6189):1272–1275, June 2014.

[10] Antoine Claessens, William L Hamilton, Mihir Kekre, Thomas D Otto, Adnan Faizullabhoy, Julian C Rayner, and Dominic Kwiatkowski. Generation of Antigenic Diversity in Plasmodium falciparum by Structured Rearrangement of Var Genes During Mitosis. *PLoS genetics*, 10(12):e1004812, December 2014.

[11] Wigard P Kloosterman, Laurent C Francioli, Fereydoun Hormozdiari, Tobias Marschall, Jayne Y Hehir-Kwa, Abdel Abdellaoui, Eric-Wubbo Lameijer, Matthijs H Moed, Vyacheslav Koval, Ivo Renkens, Markus J van Roosmalen, Pascal Arp, Lennart C Karssen, Bradley P Coe, Robert E Handsaker, Eka D Suchiman, Edwin Cuppen, Djie T Thung, Mitch McVey, Michael C Wendl, Andre Uitterlinden, Cornelia M van Duijn, Morris Swertz, Cisca Wijmenga, Gertjan van Ommen, P Eline Slagboom, Dorret I Boomsma, Alexander Schönhuth, Evan E Eichler, Paul I W de Bakker, Kai Ye, and Victor Guryev. Characteristics of de novo structural changes in the human genome. page gr.185041.114, 2015.

[12] Laurent C Francioli, Paz P Polak, Amnon Koren, Androniki Menelaou, Sung Chun, Ivo Renkens, Cornelia M van Duijn, Morris Swertz, Cisca Wijmenga, Gertjan van Ommen, P Eline Slagboom, Dorret I Boomsma, Kai Ye, Victor Guryev, Peter F Arndt, Wigard P Kloosterman, Paul I W de Bakker, and Shamil R Sunyaev. Genome-wide patterns and properties of de novo mutations in humans. *Nature genetics*, 47(7):822–826, May 2015.

[13] Satoru Fukiya, Hiroshi Mizoguchi, Toru Tobe, and Hideo Mori. Extensive genomic diversity in pathogenic Escherichia coli and Shigella Strains revealed by comparative genomic hybridization microarray. *Journal of bacteriology*, 186(12):3911–3921, June 2004.

[14] Betsy A Read, Jessica Kegel, Mary J Klute, Alan Kuo, Stephane C Lefebvre, Florian Maumus, Christoph Mayer, John Miller, Adam Monier, Asaf Salamov, Jeremy Young, Maria Aguilar, Jean-Michel Claverie, Stephan Frickenhaus, Karina Gonzalez, Emily K Herman, Yao-Cheng Lin, Johnathan Napier, Hiroyuki Ogata, Analissa F Sarno, Jeremy Shmutz, Declan Schroeder, Colomban de Vargas, Frederic Verret, Peter von Dassow, Klaus Valentin, Yves Van de Peer, Glen Wheeler, Andrew E Allen, Kay Bidle, Mark Borodovsky, Chris Bowler, Colin Brownlee, J Mark Cock, Marek Elias, Vadim N Gladyshev, Marco Groth, Chittibabu Guda, Ahmad Hadaegh, Maria Debora Iglesias-Rodriguez, Jerry Jenkins, Bethan M Jones, Tracy Lawson, Florian Leese, Erika Lindquist, Alexei Lobanov, Alexandre Lomsadze, Shehre-Banoo Malik, Mary E Marsh, Luke Mackinder, Thomas Mock, Bernd Mueller-Roeber, António Pagarete, Micaela Parker, Ian Probert, Hadi Quesneville, Christine Raines, Stefan A Rensing, Diego Mauricio Riaño-Pachón, Sophie Richier, Sebastian Rokitta, Yoshihiro Shiraiwa, Darren M Soanes, Mark van der Giezen, Thomas M Wahlund, Bryony Williams, Willie Wilson, Gordon Wolfe, Louie L Wurch, Joel B Dacks, Charles F Delwiche, Sonya T Dyhrman, Gernot Glöckner, Uwe

John, Thomas Richards, Alexandra Z Worden, Xiaoyu Zhang, and Igor V Grigoriev. Pan genome of the phytoplankton Emiliania underpins its global distribution. *Nature*, 499(7457):209–213, June 2013.

[15] A J Mungall, S A Palmer, S K Sims, C A Edwards, J L Ashurst, L Wilming, M C Jones, R Horton, S E Hunt, C E Scott, J G R Gilbert, M E Clamp, G Bethel, S Milne, R Ainscough, J P Almeida, K D Ambrose, T D Andrews, R I S Ashwell, A K Babbage, C L Bagguley, J Bailey, R Banerjee, D J Barker, K F Barlow, K Bates, D M Beare, H Beasley, O Beasley, C P Bird, S Blakey, S Bray-Allen, J Brook, A J Brown, J Y Brown, D C Burford, W Burrill, J Burton, C Carder, N P Carter, J C Chapman, S Y Clark, G Clark, C M Clee, S Clegg, V Cobley, R E Collier, J E Collins, L K Colman, N R Corby, G J Coville, K M Culley, P Dhami, J Davies, M Dunn, M E Earthrowl, A E Ellington, K A Evans, L Faulkner, M D Francis, A Frankish, J Frankland, L French, P Garner, J Garnett, M J R Ghori, L M Gilby, C J Gillson, R J Glithero, D V Grafham, M Grant, S Gribble, C Griffiths, M Griffiths, R Hall, K S Halls, S Hammond, J L Harley, E A Hart, P D Heath, R Heathcott, S J Holmes, P J Howden, K L Howe, G R Howell, E Huckle, S J Humphray, M D Humphries, A R Hunt, C M Johnson, A A Joy, M Kay, S J Keenan, A M Kimberley, A King, G K Laird, C Langford, S Lawlor, D A Leongamornlert, M Leversha, C R Lloyd, D M Lloyd, J E Loveland, J Lovell, S Martin, M Mashreghi-Mohammadi, G L Maslen, L Matthews, O T McCann, S J McLaren, K McLay, A McMurray, M J F Moore, J C Mullikin, D Niblett, T Nickerson, K L Novik, K Oliver, E K Overton-Larty, A Parker, R Patel, A V Pearce, A I Peck, B Phillimore, S Phillips, R W Plumb, K M Porter, Y Ramsey, S A Ranby, C M Rice, M T Ross, S M Searle, H K Sehra, E Sheridan, C D Skuce, S Smith, M Smith, L Spraggon, S L Squares, C A Steward, N Sycamore, G Tamlyn-Hall, J Tester, A J Theaker, D W Thomas, A Thorpe, A Tracey, A Tromans, B Tubby, M Wall, J M Wallis, A P West, S S White, S L Whitehead, H Whittaker, A Wild, D J Willey, T E Wilmer, J M Wood, P W Wray, J C Wyatt, L Young, R M Younger, D R Bentley, A Coulson, R Durbin, T Hubbard, J E Sulston, I Dunham, J Rogers, and S Beck. The DNA sequence and analysis of human chromosome 6. *Nature*, 425(6960):805–811, October 2003.

[16] Takashi Shiina, Kazuyoshi Hosomichi, Hidetoshi Inoko, and Jerzy K Kulski. The HLA genomic loci map: expression, interaction, diversity and disease. *Journal of Human Genetics*, 54(1):15–39, January 2009.

[17] Faviel F González-Galarza, Louise Y C Takeshita, Eduardo J M Santos, Felicity Kempson, Maria Helena Thomaz Maia, Andrea Luciana Soares da Silva, André

Luiz Teles e Silva, Gurpreet S Ghattaoraya, Ana Alfirevic, Andrew R Jones, and Derek Middleton. Allele frequency net 2015 update: new features for HLA epitopes, KIR and disease and HLA adverse drug reaction associations. *Nucleic acids research*, 43(D1):gku1166–D788, November 2014.

[18] 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, April 2015.

[19] Alistair Miles, Zamin Iqbal, Paul Vauterin, Richard Pearson, Susana Campino, Michel Theron, Kelda Gould, Daniel Mead, Eleanor Drury, John O'Brien, Valentin Ruano Rubio, Bronwyn MacInnis, Jonathan Mwangi, Upeka Samarakoon, Lisa Ranford-Cartwright, Michael Ferdig, Karen Hayton, Xinzhuan Su, Thomas Wellems, Julian Rayner, Gil McVean, and Dominic Kwiatkowski. Genome variation and meiotic recombination in Plasmodium falciparum: insights from deep sequencing of genetic crosses. *bioRxiv*, page 024182, August 2015.

[20] D Walliker, I Quakyi, T Wellems, T McCutchan, A Szarfman, W London, L Corcoran, T Burkot, and R Carter. Genetic analysis of the human malaria parasite Plasmodium falciparum. 236(4809):1661–1666, June 1987.

[21] D S Peterson, D Walliker, and T E Wellems. Evidence that a point mutation in dihydrofolate reductase-thymidylate synthase confers resistance to pyrimethamine in falciparum malaria. *Proceedings of the National Academy of Sciences of the United States of America*, 85(23):9114–9118, December 1988.

[22] T E Wellems, L J Panton, I Y Gluzman, V E do Rosario, R W Gwadz, A Walker-Jonah, and D J Krogstad. Chloroquine resistance not linked to mdr-like genes in a Plasmodium falciparum cross. *Nature*, 345(6272):253–255, May 1990.

[23] A B Vaidya, O Muratova, F Guinet, D Keister, T E Wellems, and D C Kaslow. A genetic locus on Plasmodium falciparum chromosome 12 linked to a defect in mosquito-infectivity and male gametogenesis. *Molecular and biochemical parasitology*, 69(1):65–71, January 1995.

[24] T Furuya, J Mu, K Hayton, A Liu, J Duan, L Nkrumah, D A Joy, D A Fidock, H Fujioka, A B Vaidya, T E Wellems, and X z Su. Disruption of a Plasmodium falciparum gene linked to male sexual development causes early arrest in gametocytogenesis. *Proceedings of the National Academy of Sciences of the United States of America*, 102(46):16813–16818, November 2005.

[25] L H Freitas-Junior, E Bottius, L A Pirrit, K W Deitsch, C Scheidig, F Guinet, U Nehrbass, T E Wellems, and A Scherf. Frequent ectopic recombination of virulence factor genes in telomeric chromosome clusters of P. falciparum. *Nature*, 407(6807):1018–1022, October 2000.

[26] Michael F Duffy, Timothy J Byrne, Celine Carret, Alasdair Ivens, and Graham V Brown. Ectopic recombination of a malaria var gene during mitosis associated with an altered var switch rate. *Journal of molecular biology*, 389(3):453–469, June 2009.

[27] 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, Aaron McKenna, Tim J Fennell, Andrew M Kernytsky, Andrey Y Sivachenko, Kristian Cibulskis, Stacey B Gabriel, David Altshuler, and Mark J Daly. A framework for variation discovery and genotyping using next-generation DNA sequencing data. *Nature genetics*, 43(5):491–498, May 2011.

[28] Avinash Ramu, Michiel J Noordam, Rachel S Schwartz, Arthur Wuster, Matthew E Hurles, Reed A Cartwright, and Donald F Conrad. DeNovoGear: de novo indel and point mutation discovery and phasing. *Nature methods*, 10(10):985–987, October 2013.

[29] Andy Rimmer, Hang Phan, Iain Mathieson, Zamin Iqbal, Stephen R F Twigg, Andrew O M Wilkie, Gil McVean, and Gerton Lunter. Integrating mapping-, assembly- and haplotype-based approaches for calling variants in clinical sequencing applications. *Nature genetics*, 46(8):912–918, July 2014.

[30] G Miller. Immortalization of human lymphocytes by Epstein-Barr virus. *The Yale Journal of Biology and Medicine*, 55(3-4):305–310, May 1982.

[31] N Woodford and M J Ellington. The emergence of antibiotic resistance by mutation. *Clinical Microbiology and Infection*, 13(1):5–18, 2007.

[32] Eric S Lander and Michael S Waterman. Genomic mapping by fingerprinting random clones: A mathematical analysis. *Genomics*, 2(3):231–239, April 1988.

[33] 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, February 2012.

[34] Malcolm J Gardner, Neil Hall, Eula Fung, Owen White, Matthew Berriman, Richard W Hyman, Jane M Carlton, Arnab Pain, Karen E Nelson, Sharen Bowman, Ian T Paulsen, Keith James, Jonathan A Eisen, Kim Rutherford, Steven L Salzberg, Alister Craig, Sue Kyes, Man-Suen Chan, Vishvanath Nene, Shamira J Shallom, Bernard Suh, Jeremy Peterson, Sam Angiuoli, Mihaela Pertea, Jonathan Allen, Jeremy Selengut, Daniel Haft, Michael W Mather, Akhil B Vaidya, David M A Martin, Alan H Fairlamb, Martin J Fraunholz, David S Roos, Stuart A Ralph, Geoffrey I McFadden, Leda M Cummings, G Mani Subramanian, Chris Mungall, J Craig Venter, Daniel J Carucci, Stephen L Hoffman, Chris Newbold, Ronald W Davis, Claire M Fraser, and Bart Barrell. Genome sequence of the human malaria parasite Plasmodium falciparum. 419(6906):498–511, October 2002.

[35] C Aurrecoechea, J Brestelli, B P Brunk, J Dommer, S Fischer, B Gajria, X Gao, A Gingle, G Grant, O S Harb, M Heiges, F Innamorato, J Iodice, J C Kissinger, E Kraemer, W Li, J A Miller, V Nayak, C Pennington, D F Pinney, D S Roos, C Ross, C J Stoeckert, C Treatman, and H Wang. PlasmoDB: a functional genomic database for malaria parasites. *Nucleic acids research*, 37(Database):D539–D543, January 2009.

[36] Heng Li. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. March 2013.

[37] Aaron McKenna, Matthew Hanna, Eric Banks, Andrey Sivachenko, Kristian Cibulskis, Andrew Kernytsky, Kiran Garimella, David Altshuler, Stacey Gabriel, Mark Daly, and Mark A DePristo. The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. 20(9):1297–1303, September 2010.

[38] Mark J P Chaisson, Richard K Wilson, and Evan E Eichler. Genetic variation and the de novo assembly of human genomes. *Nature Reviews Genetics*, 16(11):627–640, November 2015.

[39] Ronald Ross. On some Peculiar Pigmented Cells Found in Two Mosquitos Fed on Malarial Blood. *Br Med J*, 2(1929):1786–1788, December 1897.

[40] Rogerio Amino, Sabine Thiberge, Béatrice Martin, Susanna Celli, Spencer Shorte, Friedrich Frischknecht, and Robert Ménard. Quantitative imaging of Plasmodium transmission from mosquito to mammal. *Nature medicine*, 12(2):220–224, January 2006.

[41] Donato Torre, Filippo Speranza, Massimo Giola, Alberto Matteelli, Roberto Tambini, and Gilberto Biondi. Role of Th1 and Th2 Cytokines in Immune Response to Uncomplicated Plasmodium falciparum Malaria. *Clinical and Diagnostic Laboratory Immunology*, 9(2):348–351, March 2002.

[42] K Lingelbach and K A Joiner. The parasitophorous vacuole membrane surrounding Plasmodium and Toxoplasma: an unusual compartment in infected cells. *Journal of cell science*, 111 (Pt 11):1467–1475, June 1998.

[43] Teun Bousema, Lucy Okell, Ingrid Felger, and Chris Drakeley. Asymptomatic malaria infections: detectability, transmissibility and public health relevance. *Nature reviews. Microbiology*, 12(12):833–840, December 2014.

[44] Miguel Prudêncio, Ana Rodriguez, and Maria M Mota. The silent path to thousands of merozoites: the Plasmodium liver stage. *Nature reviews. Microbiology*, 4(11):849–856, November 2006.

[45] Kenneth Murphy, Paul Travers, and Mark Walport. *Janeway's Immunobiology*. Garland Science, October 2011.

[46] Alan F Cowman and Brendan S Crabb. Invasion of Red Blood Cells by Malaria Parasites. *Cell*, 124(4):755–766, February 2006.

[47] Gavin J Wright and Julian C Rayner. Plasmodium falciparum Erythrocyte Invasion: Combining Function with Immune Evasion. *PLoS pathogens*, 10(3):e1003943, March 2014.

[48] Gabrielle A Josling and Manuel Llinás. Sexual development in Plasmodium parasites: knowing when it's time to commit. *Nature reviews. Microbiology*, 13(9):573–587, August 2015.

[49] Kirsten Flick and Qijun Chen. var genes, PfEMP1 and the human host. *Molecular and biochemical parasitology*, 134(1):3–9, March 2004.

[50] Victor Fernandez, Carl Johan Treutiger, Gerard B Nash, and Mats Wahlgren. Multiple Adhesive Phenotypes Linked to Rosetting Binding of Erythrocytes in Plasmodium falciparum Malaria. *Infection and Immunity*, 66(6):2969–2975, June 1998.

[51] Martine M Zilversmit, Ella K Chase, Donald S Chen, Philip Awadalla, Karen P Day, and Gil McVean. Hypervariable antigen genes in malaria have ancient roots. *BMC Evolutionary Biology*, 13(1):110, May 2013.

[52] Lubin Jiang, Jianbing Mu, Qingfeng Zhang, Ting Ni, Prakash Srinivasan, Kempaiah Rayavara, Wenjing Yang, Louise Turner, Thomas Lavstsen, Thor G Theander, Weiqun Peng, Guiying Wei, Qingqing Jing, Yoshiyuki Wakabayashi, Abhisheka Bansal, Yan Luo, José M C Ribeiro, Artur Scherf, L Aravind, Jun Zhu, Keji Zhao, and Louis H Miller. PfSETvs methylation of histone H3K36 represses virulence genes in Plasmodium falciparum. 499(7457):223–227, July 2013.

[53] Lorenzo Zammarchi Alessandro Bartoloni. Clinical Aspects of Uncomplicated and Severe Malaria. *Mediterranean Journal of Hematology and Infectious Diseases*, 4(1), 2012.

[54] R E Sinden. The cell biology of sexual development in plasmodium. *Parasitology*, 86 (Pt 4):7–28, April 1983.

[55] T S MacFie, E Nerrienet, R E Bontrop, and N I Mundy. The action of falciparum malaria on the human and chimpanzee genomes compared: Absence of evidence for a genomic signature of malaria at HBB and G6PD in three subspecies of chimpanzee. *Infection, Genetics and Evolution*, 9(6):1248–1252, December 2009.

[56] Lisa C Ranford-Cartwright and Jonathan M Mwangi. Analysis of malaria parasite phenotypes using experimental genetic crosses of Plasmodium falciparum. *International Journal for Parasitology*, 42(6):529–534, May 2012.

[57] Xinzhuan Su, Karen Hayton, and Thomas E Wellems. Genetic linkage and association analyses for trait mapping in Plasmodium falciparum. *Nature Reviews Genetics*, 8(7):497–506, July 2007.

[58] F L Schuster. Cultivation of Plasmodium spp. *Clinical Microbiology Reviews*, 15(3):355–364, July 2002.

[59] P Alano, L Roca, D Smith, D Read, R Carter, and K Day. Plasmodium falciparum: parasites defective in early stages of gametocytogenesis. *Experimental Parasitology*, 81(2):227–235, September 1995.

[60] Mathieu Gissot, Philippe Refour, Sylvie Briquet, Charlotte Boschet, Stéphane Coupé, Dominique Mazier, and Catherine Vaquero. Transcriptome of 3D7 and its gametocyte-less derivative F12 Plasmodium falciparum clones during erythrocytic development using a gene-specific microarray assigned to gene regulation, cell cycle and transcription factors. *Gene*, 341:267–277, October 2004.

[61] Francesco Silvestrini, Zbynek Bozdech, Alessandra Lanfrancotti, Eugenia Di Giulio, Emanuele Bultrini, Leonardo Picci, Joseph L Derisi, Elisabetta Pizzi, and Pietro Alano. Genome-wide identification of genes upregulated at the onset of gametocytogenesis in Plasmodium falciparum. *Molecular and biochemical parasitology*, 143(1):100–110, September 2005.

[62] Anthony J F Griffiths, Jeffrey H Miller, David T Suzuki, Richard C Lewontin, and William M Gelbart. *An Introduction to Genetic Analysis*. W H Freeman & Company, April 2003.

[63] Seongman Kang, Adam Jaworski, Keiichi Ohshima, and Robert D Wells. Expansion and deletion of CTG repeats from human disease genes are determined by the direction of replication in E. coli. *Nature genetics*, 10(2):213–218, June 1995.

[64] Michael R Lieber. The mechanism of human nonhomologous DNA end joining. *The Journal of biological chemistry*, 283(1):1–5, January 2008.

[65] Anat Haviv-Chesner, Yoshifumi Kobayashi, Abram Gabriel, and Martin Kupiec. Capture of linear fragments at a double-strand break in yeast. *Nucleic acids research*, 35(15):5192–5202, 2007.

[66] Mitch McVey and Sang Eun Lee. MMEJ repair of double-strand breaks (director's cut): deleted sequences and alternative endings. *Trends Genet*, 24(11):529–538, November 2008.

[67] Tatiana Kent, Gurushankar Chandramouly, Shane Michael McDevitt, Ahmet Y Ozdemir, and Richard T Pomerantz. Mechanism of microhomology-mediated end-joining promoted by human DNA polymerase θ. *Nature Structural & Molecular Biology*, 22(3):230–237, February 2015.

[68] F L Lin, K Sperle, and N Sternberg. Model for homologous recombination during transfer of DNA into mouse L cells: role for DNA ends in the recombination process. *Molecular and cellular biology*, 4(6):1020–1034, June 1984.

[69] Rita Rebollo, Mark T Romanish, and Dixie L Mager. Transposable Elements: An Abundant and Natural Source of Regulatory Sequences for Host Genes. *Annual review of genetics*, 46(1):21–42, December 2012.

[70] Christian Biémont and Cristina Vieira. Genetics: Junk DNA as an evolutionary force. *Nature*, 443(7111):521–524, October 2006.

[71] Patrick Sung and Hannah Klein. Mechanism of homologous recombination: mediators and helicases take on regulatory functions. *Nature reviews. Molecular cell biology*, 7(10):739–750, August 2006.

[72] Jian-Min Chen, David N Cooper, Nadia Chuzhanova, Claude Férec, and George P Patrinos. Gene conversion: mechanisms, evolution and human disease. *Nature Reviews Genetics*, 8(10):762–775, October 2007.

[73] Wenli Gu, Feng Zhang, and James R Lupski. Mechanisms for human genomic rearrangements. *PathoGenetics*, 1(1):4, November 2008.

[74] Mariko Sasaki, Julian Lange, and Scott Keeney. Genome destabilization by homologous recombination in the germ line. *Nature reviews. Molecular cell biology*, 11(3):182–195, March 2010.

[75] Angelika C Roehl, Julia Vogt, Tanja Mussotter, Antje N Zickler, Helene Spöti, Josef Högel, Nadia A Chuzhanova, Katharina Wimmer, Lan Kluwe, Victor Felix Mautner, David N Cooper, and Hildegard Kehrer Sawatzki. Intrachromosomal mitotic nonallelic homologous recombination is the major molecular mechanism underlying type-2 NF1 deletions. *Human Mutation*, 31(10):1163–1173, September 2010.

[76] Matthew M Parks, Charles E Lawrence, and Benjamin J Raphael. Detecting nonallelic homologous recombination from high-throughput sequencing data. *Genome Biology*, 16(1):704, 2015.

[77] Bobo W Mok, Ulf Ribacke, Ellen Sherwood, and Mats Wahlgren. A highly conserved segmental duplication in the subtelomeres of Plasmodium falciparum chromosomes varies in copy number. *Malaria journal*, 7(1):1–9, 2008.

[78] Selina E R Bopp, Micah J Manary, A Taylor Bright, Geoffrey L Johnston, Neekesh V Dharia, Fabio L Luna, Susan McCormack, David Plouffe, Case W McNamara, John R Walker, David A Fidock, Eros Lazzerini Denchi, and Elizabeth A Winzeler. Mitotic Evolution of Plasmodium falciparum Shows a Stable Core Genome but Recombination in Antigen Families. *PLoS genetics*, 9(2):e1003293, February 2013.

[79] J F Crow. The origins, patterns and implications of human spontaneous mutation. *Nature Reviews Genetics*, 1(1):40–47, October 2000.

[80] Rivka L Glaser, Karl W Broman, Rebecca L Schulman, Brenda Eskenazi, Andrew J Wyrobek, and Ethylin Wang Jabs. The paternal-age effect in Apert syndrome is

due, in part, to the increased frequency of mutations in sperm. *American journal of human genetics*, 73(4):939–947, October 2003.

[81] Anne Goriely and Andrew O M Wilkie. Paternal Age Effect Mutations and Selfish Spermatogonial Selection: Causes and Consequences for Human Disease. *The American Journal of Human Genetics*, 90(2):175–200, February 2012.

[82] Joris A Veltman and Han G Brunner. De novo mutations in human genetic disease. *Nature Reviews Genetics*, 13(8):565–575, August 2012.

[83] Petr Danecek, Adam Auton, Goncalo Abecasis, Cornelis A Albers, Eric Banks, Mark A DePristo, Robert Handsaker, Gerton Lunter, Gabor Marth, Stephen T Sherry, Gilean McVean, Richard Durbin, and 1000 Genomes Project Analysis Group. The Variant Call Format and VCFtools. *Bioinformatics (Oxford, England)*, June 2011.

[84] Luc Devroye. *Non-Uniform Random Variate Generation*. Springer Science & Business Media, New York, NY, November 2013.

[85] Thomas S Rask, Daniel A Hansen, Thor G Theander, Anders Gorm Pedersen, and Thomas Lavstsen. Plasmodium falciparum erythrocyte membrane protein 1 diversity in seven genomes–divide and conquer. *PLoS computational biology*, 6(9), 2010.

[86] Susan M Kraemer and Joseph D Smith. A family affair: var genes, PfEMP1 binding, and malaria disease. *Current Opinion in Microbiology*, 9(4):374–380, August 2006.

[87] Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor Marth, Goncalo Abecasis, Richard Durbin, and 1000 Subgroup. The Sequence Alignment/Map format and SAMtools. *Bioinformatics (Oxford, England)*, 25(16):2078–2079, August 2009.

[88] H Bao, H Guo, J Wang, R Zhou, X Lu, and S Shi. MapView: visualization of short reads alignment on a desktop computer. *Bioinformatics (Oxford, England)*, 25(12):1554–1555, May 2009.

[89] James T Robinson, Helga Thorvaldsdóttir, Wendy Winckler, Mitchell Guttman, Eric S Lander, Gad Getz, and Jill P Mesirov. Integrative genomics viewer. *Nature biotechnology*, 29(1):24–26, January 2011.

[90] Iain Milne, Gordon Stephen, Micha Bayer, Peter J A Cock, Leighton Pritchard, Linda Cardle, Paul D Shaw, and David Marshall. Using Tablet for visual exploration of second-generation sequencing data. *Briefings in bioinformatics*, 14(2):193–202, March 2013.

[91] Helga Thorvaldsdóttir, James T Robinson, and Jill P Mesirov. Integrative Genomics Viewer (IGV): high-performance genomics data visualization and exploration. *Briefings in bioinformatics*, 14(2):178–192, March 2013.

[92] Joshua N Burton, Andrew Adey, Rupali P Patwardhan, Ruolan Qiu, Jacob O Kitzman, and Jay Shendure. Chromosome-scale scaffolding of de novo genome assemblies based on chromatin interactions. *Nature biotechnology*, 31(12):1119–1125, December 2013.

[93] Isaac Turner. McCortex Workflow Examples. November 2015.

[94] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models*. Principles and Techniques. MIT Press, 2009.

[95] N G Bruijn. *A combinatorial problem*. ... Nederlandse Akademie van Wetenschappen. Series A, 1946.

[96] John Hopcroft and Robert Tarjan. Efficient algorithms for graph manipulation, 1971.

[97] S F Altschul, W Gish, W Miller, E W Myers, and D J Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, October 1990.

[98] E W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[99] Stefan Kurtz, Adam Phillippy, Arthur L Delcher, Michael Smoot, Martin Shumway, Corina Antonescu, and Steven L Salzberg. Versatile and open software for comparing large genomes. *Genome Biology*, 5(2):R12, January 2004.

[100] Adrian J Gibbs and George A Mcintyre. The Diagram, a Method for Comparing Sequences. *European Journal of Biochemistry*, 16(1):1–11, September 1970.

[101] Karen Hayton, Deepak Gaur, Anna Liu, Jonathan Takahashi, Bruce Henschen, Subhash Singh, Lynn Lambert, Tetsuya Furuya, Rachel Bouttenot, Michelle Doll, Fatima Nawaz, Jianbing Mu, Lubin Jiang, Louis H Miller, and Thomas E Wellems. Erythrocyte Binding Protein PfRH5 Polymorphisms Determine Species-Specific Pathways of Plasmodium falciparum Invasion. *Cell Host & Microbe*, 4(1):40–51, January 2008.

[102] William Brockman, Pablo Alvarez, Sarah Young, Manuel Garber, Georgia Giannoukos, William L Lee, Carsten Russ, Eric S Lander, Chad Nusbaum, and David B Jaffe. Quality scores and SNP detection in sequencing-by-synthesis systems. 18(5):763–770, May 2008.

[103] Chen-Shan Chin, David H Alexander, Patrick Marks, Aaron A Klammer, James Drake, Cheryl Heiner, Alicia Clum, Alex Copeland, John Huddleston, Evan E Eichler, Stephen W Turner, and Jonas Korlach. Nonhybrid, finished microbial genome assemblies from long-read SMRT sequencing data. *Nature methods*, 10(6):563–569, June 2013.

[104] E W Myers. Toward simplifying and accurately formulating fragment assembly. *Journal of computational biology : a journal of computational molecular cell biology*, 2(2):275–290, June 1995.

[105] J T Simpson and R Durbin. Efficient construction of an assembly string graph using the FM-index. *Bioinformatics (Oxford, England)*, 26(12):i367–i373, June 2010.

[106] Martin I Krzywinski, Jacqueline E Schein, Inanc Birol, Joseph Connors, Randy Gascoyne, Doug Horsman, Steven J Jones, and Marco A Marra. Circos: An information aesthetic for comparative genomics. 19(9):1639–1645, June 2009.

[107] C E Shannon. A mathematical theory of communication. *Bell System Technical Journal, The*, 27(3):379–423, July 1948.

[108] E N Trifonov. Making sense of the human genome. *Structure and methods : proceedings of the Sixth Conversation in the Discipline Biomolecular Stereodynamics held at the State University of New York at Albany, June 6-10, 1989 / edited by R.H. Sarma & M.H. Sarma*, 1990.

[109] J C Wootton and S Federhen. Analysis of compositionally biased regions in sequence databases. *Methods in enzymology*, 266:554–571, 1996.

[110] C K Stover, X Q Pham, A L Erwin, S D Mizoguchi, P Warrener, M J Hickey, F S L Brinkman, W O Hufnagle, D J Kowalik, M Lagrou, R L Garber, L Goltry, E Tolentino, S Westbrock-Wadman, Y Yuan, L L Brody, S N Coulter, K R Folger, A Kas, K Larbig, R Lim, K Smith, D Spencer, G K S Wong, Z Wu, I T Paulsen, J Reizer, M H Saier, R E W Hancock, S Lory, and M V Olson. Complete genome sequence of Pseudomonas aeruginosa PAO1, an opportunistic pathogen. *Nature*, 406(6799):959–964, August 2000.

[111] Geoffrey L Winsor, Emma J Griffiths, Raymond Lo, Bhavjinder K Dhillon, Julie A Shay, and Fiona S L Brinkman. Enhanced annotations and features for comparing thousands of Pseudomonas genomes in the Pseudomonas genome database. *Nucleic acids research*, 44(D1):D646–53, January 2016.

[112] Goo Jun, Matthew Flickinger, Kurt N Hetrick, Jane M Romm, Kimberly F Doheny, Gonçalo R Abecasis, Michael Boehnke, and Hyun Min Kang. Detecting and estimating contamination of human DNA samples in sequencing and array-based genotype data. *American journal of human genetics*, 91(5):839–848, November 2012.

[113] Adam F Sander, Thomas Lavstsen, Thomas S Rask, Michael Lisby, Ali Salanti, Sarah L Fordyce, Jakob S Jespersen, Richard Carter, Kirk W Deitsch, Thor G Theander, Anders Gorm Pedersen, and David E Arnot. DNA secondary structures are associated with recombination in major Plasmodium falciparum variable surface antigen gene families. *Nucleic acids research*, November 2013.

[114] Olivo Miotto, Jacob Almagro-Garcia, Magnus Manske, Bronwyn MacInnis, Susana Campino, Kirk A Rockett, Chanaki Amaratunga, Pharath Lim, Seila Suon, Sokunthea Sreng, Jennifer M Anderson, Socheat Duong, Chea Nguon, Char Meng Chuor, David Saunders, Youry Se, Chantap Lon, Mark M Fukuda, Lucas Amenga-Etego, Abraham V O Hodgson, Victor Asoala, Mallika Imwong, Shannon Takala-Harrison, Francois Nosten, Xin-zhuan Su, Pascal Ringwald, Frédéric Ariey, Christiane Dolecek, Tran Tinh Hien, Maciej F Boni, Cao Quang Thai, Alfred Amambua-Ngwa, David J Conway, Abdoulaye A Djimdé, Ogobara K Doumbo, Issaka Zongo, Jean-Bosco Ouedraogo, Daniel Alcock, Eleanor Drury, Sarah Auburn, Oliver Koch, Mandy Sanders, Christina Hubbart, Gareth Maslen, Valentin Ruano-Rubio, Dushyanth Jyothi, Alistair Miles, John O'Brien, Chris Gamble, Samuel O Oyola, Julian C Rayner, Chris I Newbold, Matthew Berriman, Chris C A Spencer, Gilean McVean, Nicholas P Day, Nicholas J White, Delia Bethell, Arjen M Dondorp, Christopher V Plowe, Rick M Fairhurst, and Dominic P Kwiatkowski. Multiple populations of artemisinin-resistant Plasmodium falciparum in Cambodia. *Nature genetics*, 45(6):648–655, April 2013.

[115] S Mallick, S Gnerre, P Muller, and D Reich. The difficulty of avoiding false positives in genome scans for natural selection. 19(5):922–933, May 2009.

[116] The Chimpanzee Sequencing and Analysis Consortium and Chimpanzee Sequencing and Analysis Consortium. Initial sequence of the chimpanzee genome and comparison with the human genome. *Nature*, 437(7055):69–87, August 2005.

[117] Ryan R Wick, Mark B Schultz, Justin Zobel, and Kathryn E Holt. Bandage: interactive visualization of de novo genome assemblies. 31(20):3350–3352, October 2015.

[118] Neil I Weisenfeld, Shuangye Yin, Ted Sharpe, Bayo Lau, Ryan Hegarty, Laurie Holmes, Brian Sogoloff, Diana Tabbaa, Louise Williams, Carsten Russ, Chad Nusbaum, Eric S Lander, Iain MacCallum, and David B Jaffe. Comprehensive variation discovery in single human genomes. *Nature genetics*, October 2014.

[119] Emden R Gansner, Eleftherios Koutsofios, Stephen C North, and Kiem-Phong Vo. A Technique for Drawing Directed Graphs. *IEEE Trans. Software Eng. ()*, 19(3):214–230, 1993.

[120] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D^3 Data-Driven Documents. *IEEE Trans. Vis. Comput. Graph. ()*, 17(12):2301–2309, 2011.

[121] Santo Fortunato. Community detection in graphs. *arXiv.org*, (3):75–174, June 2009.

[122] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, October 2008.

[123] Claire Lemaitre, Liviu Ciortuz, and Pierre Peterlongo. Mapping-Free and Assembly-Free Discovery of Inversion Breakpoints from Raw NGS Reads. In *Algorithms for Computational Biology*, pages 119–130. Springer International Publishing, Cham, July 2014.

Appendices

A Manual examination of all DNMs in a single P. falciparum sample

A.1 Introduction

To establish the veracity of our calls in the *P. falciparum* crosses, we manually examined all events in a single sample, the 3D7xHB3 progeny PG0063-C. This appendix presents several screenshots from IGV and our custom graph viewing software used to establish the visual veracity of the calls. The calls depicted are taken from Table 6.4.

A.2 Screenshots

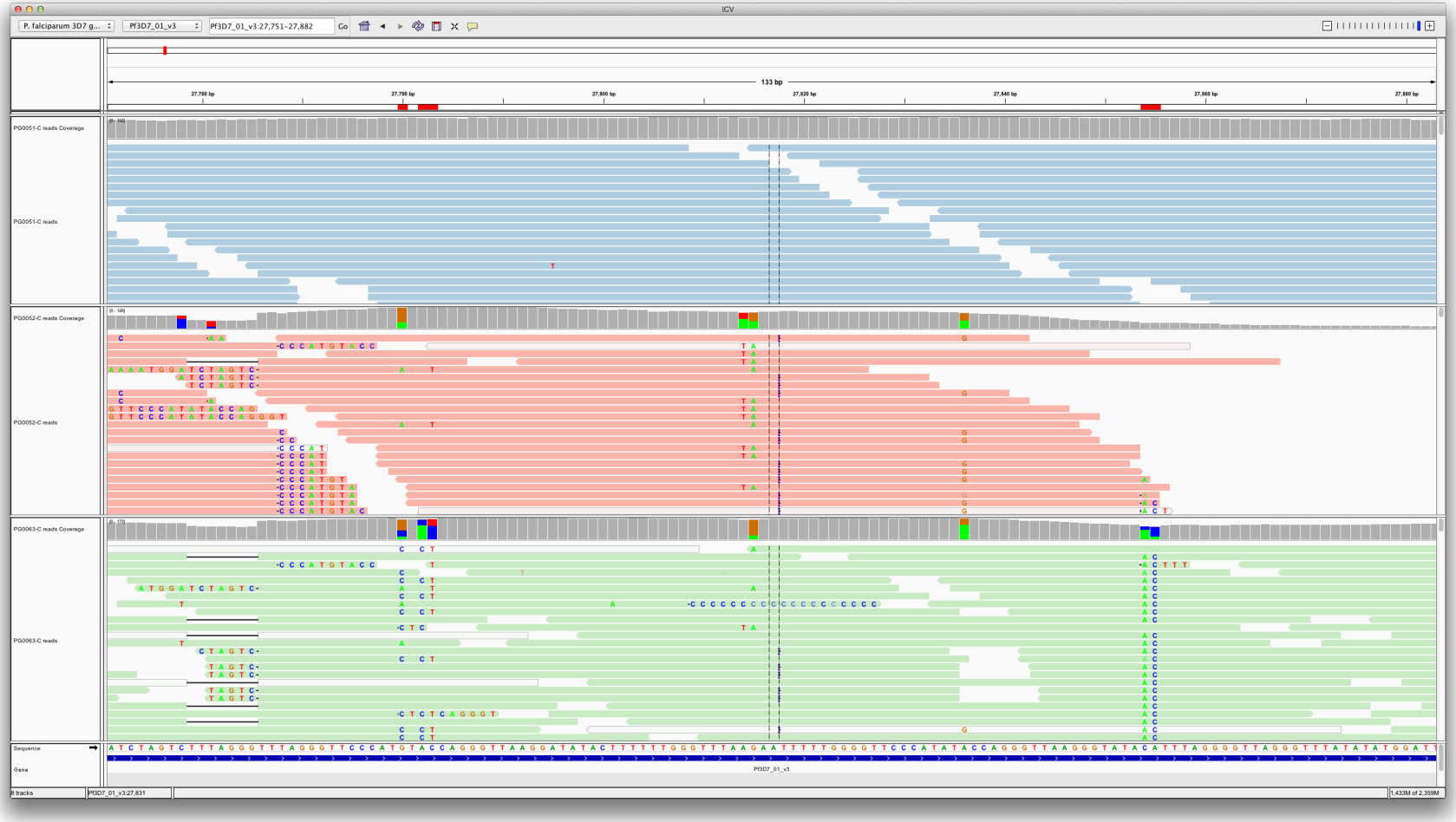


Figure A.1: IGV screenshot of PG0063-C event 1 (a-c), SNP and two MNPs: three apparent *de novo* mutations within a much larger NAHR event. Top panel: 3D7 parent. Middle panel: HB3 parent. Lower panel: PG0063-C child. Variant positions are shown as red blocks at the top of the three panels below the genome position track. Stacked barplots above variant positions indicate the proportion of bases supporting an alternate allele to the reference.

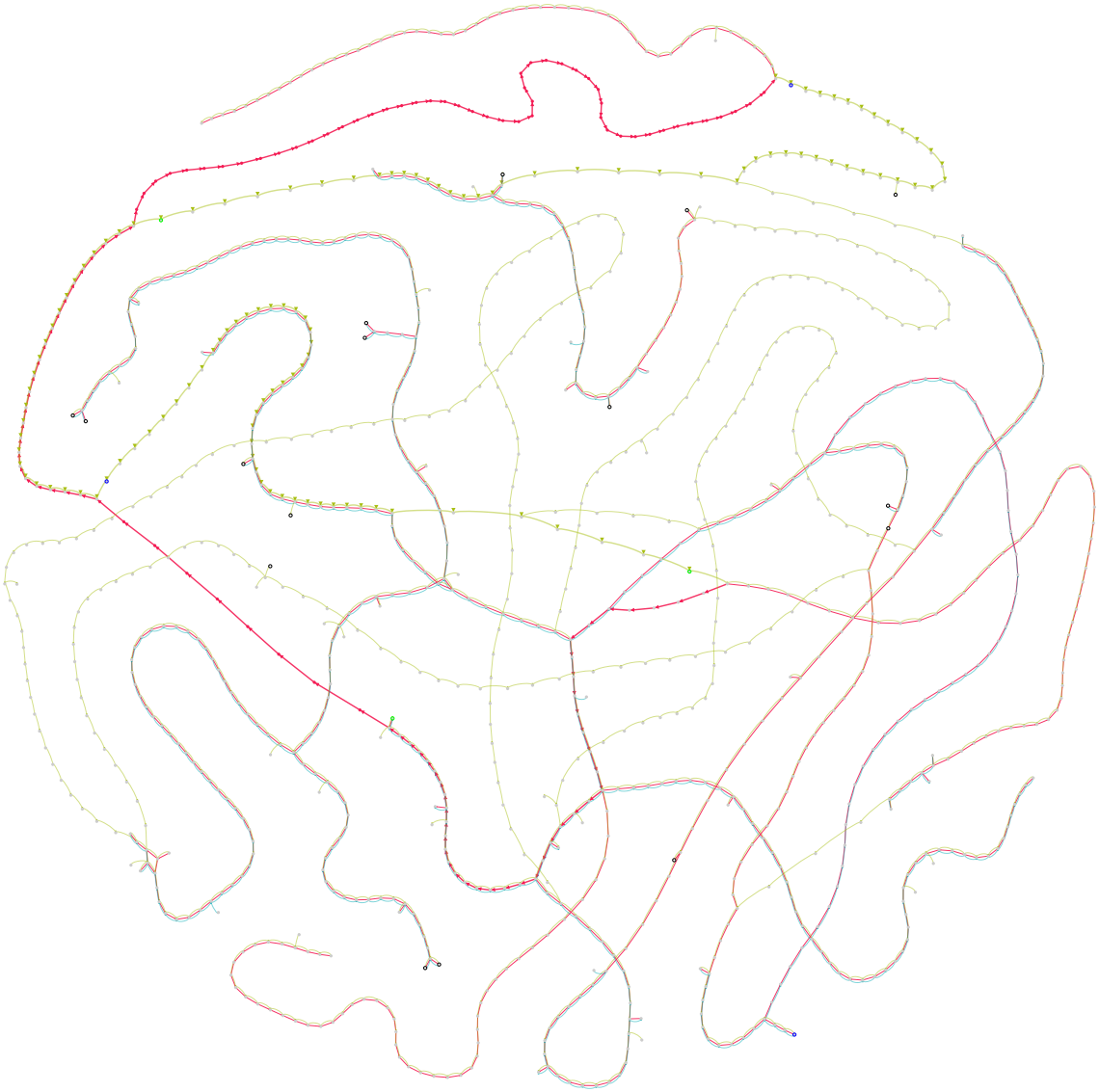


Figure A.2: Graph screenshot of event 1 (a-c), SNP and two MNPs: three apparent *de novo* mutations within a much larger NAHR event shown as two bubbles with novel kmer involvement in the upper and middle-left sections of the plot.

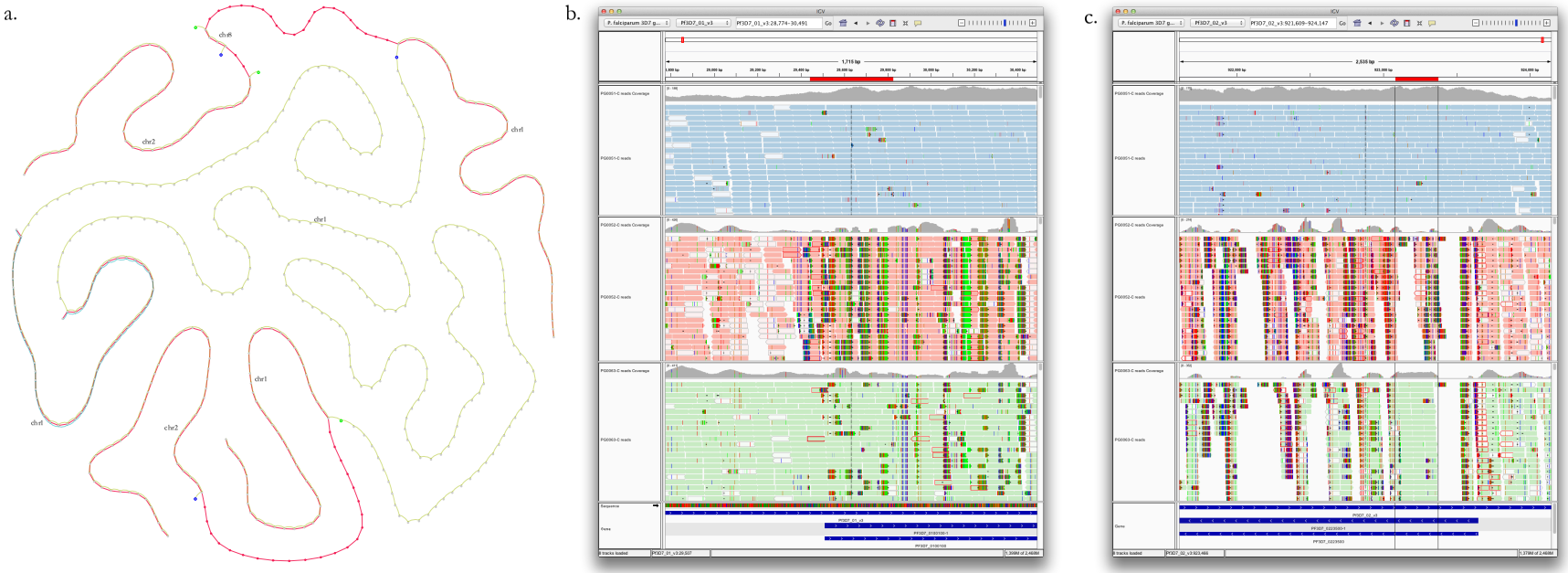


Figure A.3: IGV and graph screenshots of event 1 (d-h), a NAHR event. a. Graphical view of the exchange; chromosome of origin specified for various regions. b. IGV screenshot for 3D7 (top), HB3 (middle), and PG0063-C (child) on the chromosome 1 region of the event. c. IGV screenshot of the chromosome 2 region of the event.

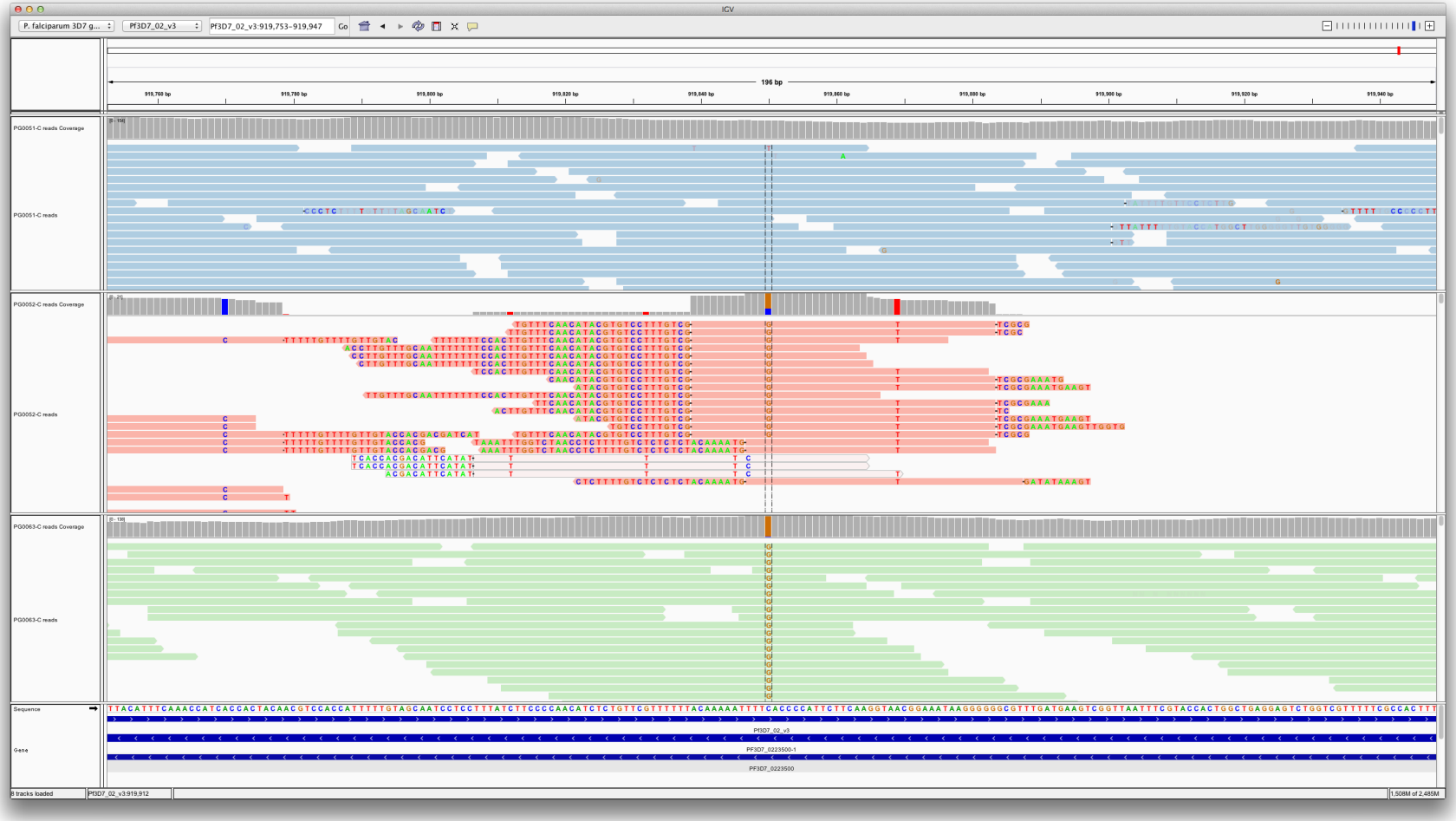


Figure A.4: Event 1 h: a *de novo* SNP in a *var* on the 3D7 haplotypic background. Top panel: 3D7. Middle: HB3. Lower: PG0063-C child.

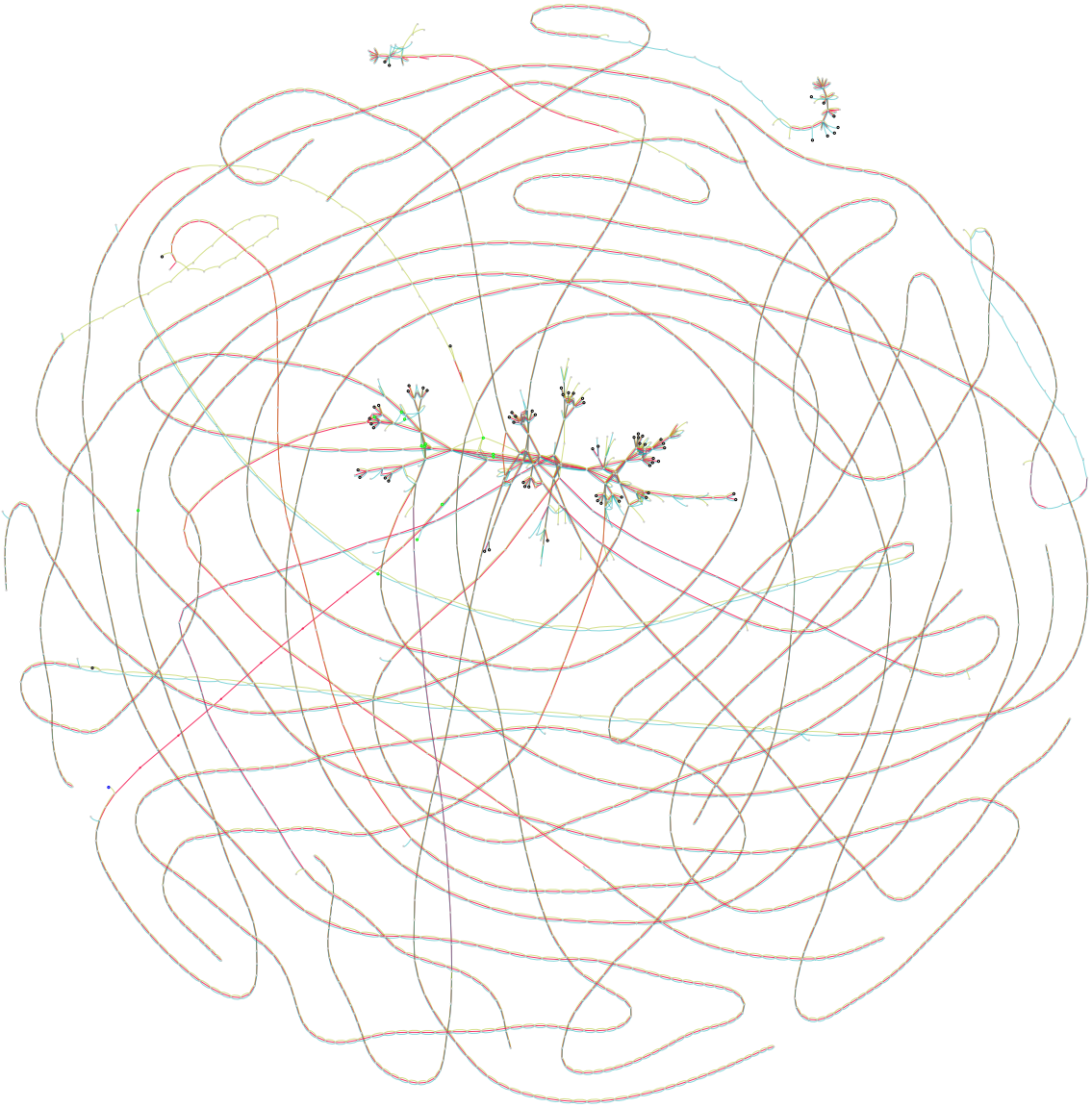


Figure A.5: Event 2, a graph "tangle": a region of the graph where the in/out degree of vertices within a small radius sharply increases, resulting from attempting to traverse a low-complexity region of the genome.

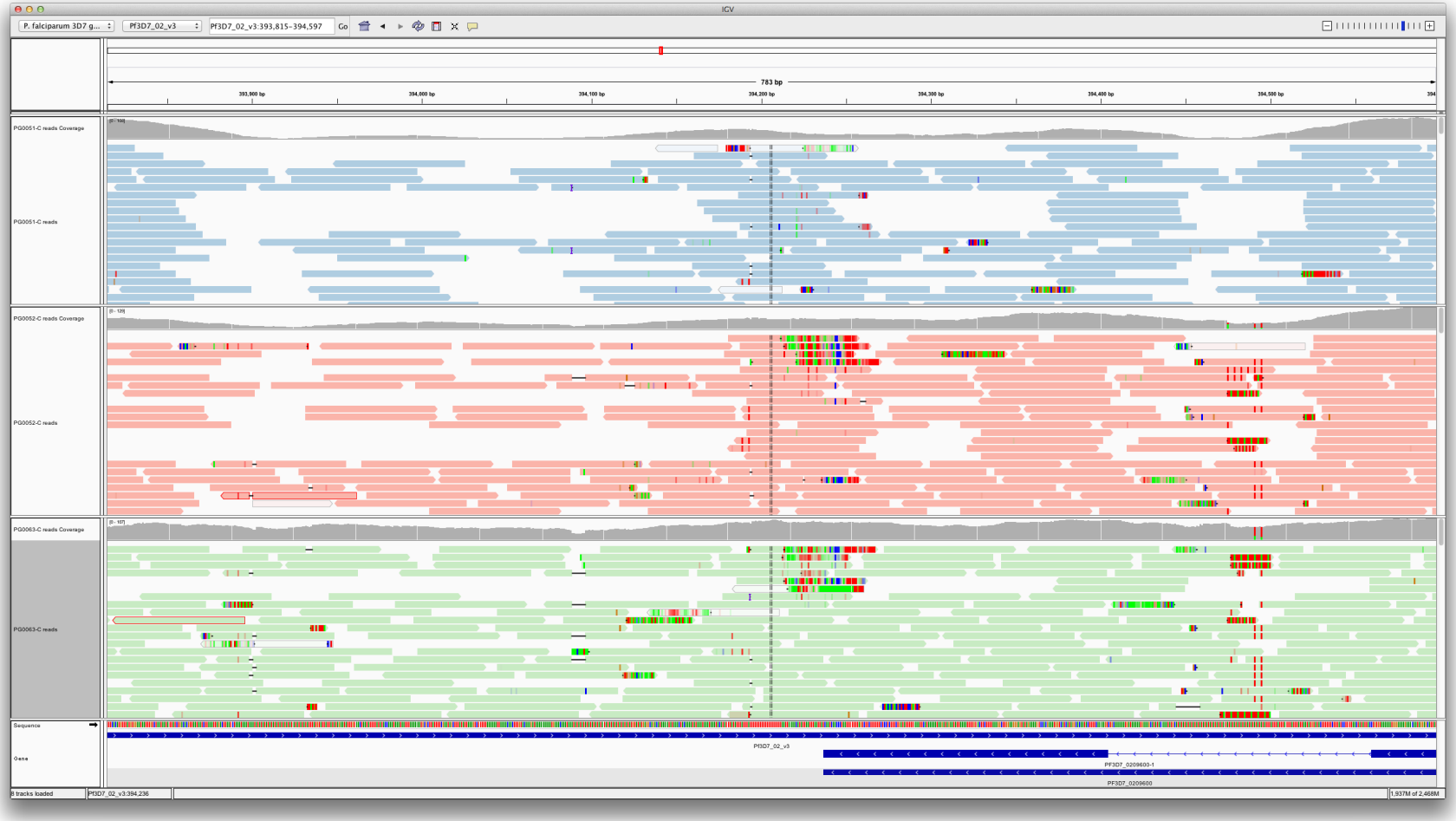


Figure A.6: IGV screenshot of event 2, a low-complexity region of the genome containing many reads with many apparent errors.

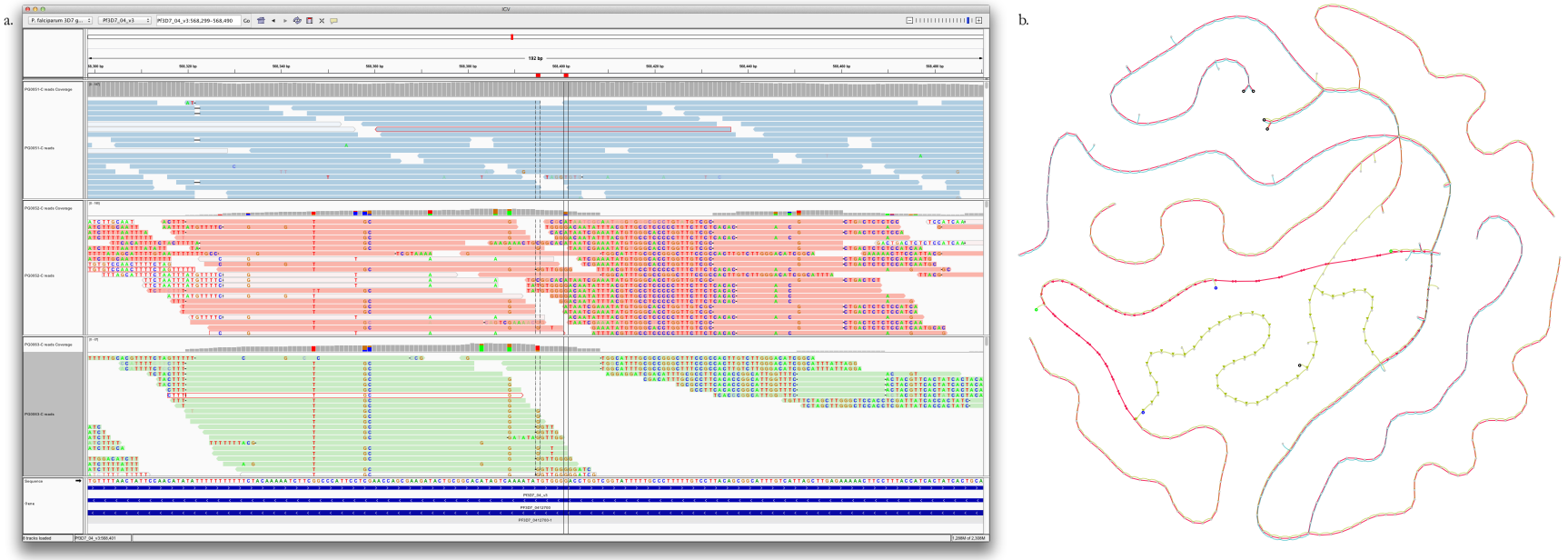


Figure A.7: Events 3 and 4, two SNPs, non-obvious from the alignments, but apparent in graph. a. IGV screenshot of two *de novo* SNPs in a *var* gene. b. Graphical view of the two SNPs.

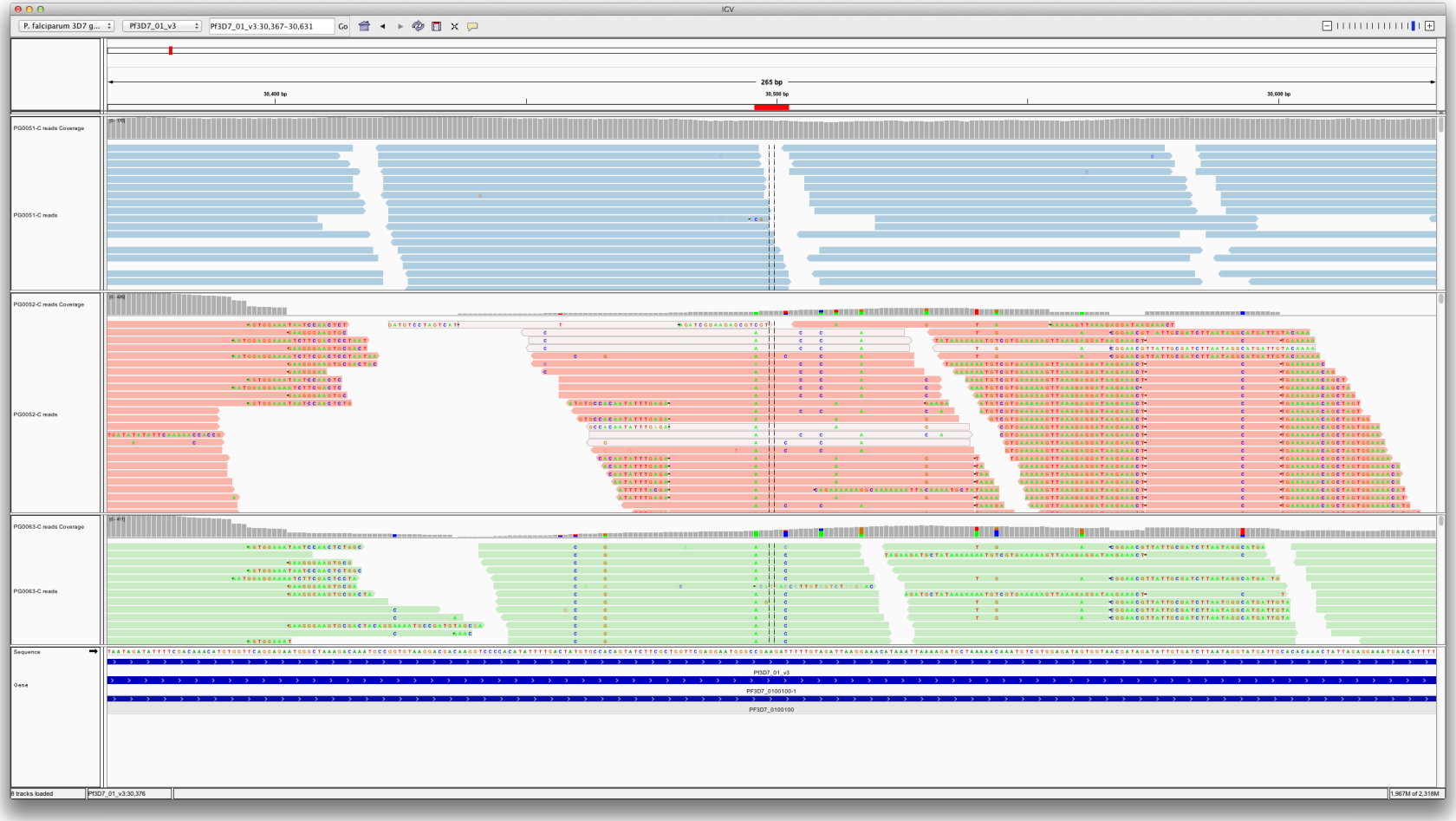


Figure A.8: The true home of the events 3 and 4, two SNPs misplaced during alignment.