Chapter 5

Assembly-Based Variant Calling Results

5.1 Introduction

In the previous chapter, I described methods for finding sequence variation by comparing sets of reads using a de Bruijn graph or string graph. In this chapter, I explore this approach to variant calling.

In section 4.2.2 I proposed that k-mers unique to a particular genome (or set of reads) can be used to discover candidate variant sequences. In section 5.2 I test this idea by simulating random mutations in the human reference genome. In sections 5.3 and 5.4 I test the full variant calling pipeline by simulating variants and sequence reads. The benefit of using simulated data is that the true set of variants is known, which allows direct calculation of the sensitivity and precision of the variant calling procedure. However, these simulations do not model some complications found in real data, like biased sequencing coverage, systematic sequencing errors or large structural variants. The remainder of the chapter uses real sequencing data from the Illumina platform. In section 5.5 I make variant calls for an individual genome compared to the human reference sequence. The variant calls for this individual are compared to previously published variants to assess the performance of my method. In 5.6 I explore the false positive rate of my variant caller.

In sections 5.7, 5.8 and 5.9 I apply my variant caller to three key variant calling problems. In section 5.7 I call mutations that occur in the child of two parents (*de novo* mutations) where all three individuals have been sequenced. Section 5.8 explores finding somatic mutations that occur during progression of cancer. Section 5.9 describes the use of assembly-based variant calling for a population of individuals, where each member of the population is sequenced at low coverage. The data used in this section is part of the 1000 Genomes Project. The results in this chapter demonstrate the performance of our algorithms and their software implementation in a wide variety of contexts. In the last section of this chapter the results are discussed in a broader context including future areas of work.

5.1.1 Implementation Note

The algorithms from Chapter 4 are implemented within my FM-Index assembler, SGA. Version 0.9.30 of SGA was used for this chapter. The source code is freely available online at `www.github.com/jts/sga`.

5.2 The power to detect variants using unique k-mers

In section 4.2.2 I described an algorithm to find candidate variants between two genomes, G_v and G_c, by finding k-mers unique to G_v. To assess the power of detecting candidate variants using this approach, I performed a simulation by introducing point mutations randomly into the human reference genome (build GRC 37, preprocessed to remove sequence gaps). If any k-mer containing the introduced point mutation is not found in the human reference genome (it is a unique k-mer), I call the mutation *detectable*. If all k-mers containing the point mutation are unique, I say that the mutation forms a *clean bubble*. I performed this simulation for all k from 16 to 71. In each simulation, $10,000,000$ random mutations were introduced.

The results of this simulation are plotted in figure 5.1. When $k = 21$, 93.8% of variants are detectable but only 63.6% of variants form clean bubbles. When $k = 51$, 99.6% of variants are detectable and 93.7% form clean bubbles. These

results highlight the power of using unique k-mers for finding potential variants - even for relatively small k most changes generate unique k-mers which we can use to start the haplotype assembly process described in the previous chapter.

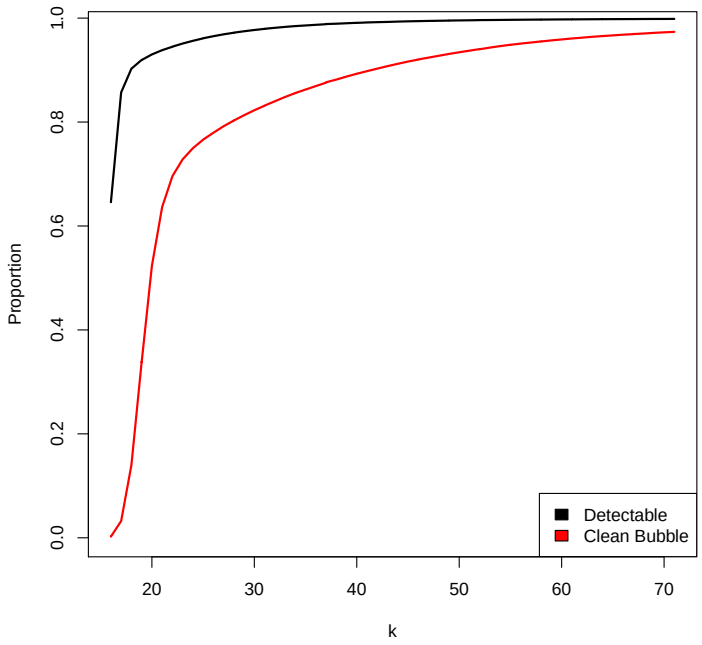

Figure 5.1: The k-mer detectability of point mutations introduced into the human reference genome. The black line indicates the proportion of introduced variants that are detectable at a given k. The red line indicates the proportion of variants that form clean bubbles.

5.3 Simulated single-genome variants calls

As an initial test of our complete variant calling algorithm, I generated two new chromosomes derived from human chromosome 20. First, the human chromosome 20 sequence was pre-processed to remove "N"s from the reference without changing the coordinate system by choosing a random base for each "N" symbol. I then generated two sets of mutations - one set of homozygous changes and one

set of heterozygous changes. Each set contained random substitution mutations at a frequency of 1 in 2,000bp and random indel mutations at frequency 1 in 20,000bp. The indel size was generated by starting at 1bp and extending the event with probability 0.3. In total, 34,701 homozygous and 34,670 heterozygous events were created. The heterozygous events were randomly partitioned into two subsets. To derive the new chromosomes from chromosome 20, all homozygous events were applied to chromosome 20 and one of the two heterozygous subsets using the tool `FastaAlternateReferenceMaker` from the Genome Analysis Toolkit [DePristo et al., 2011]. I will refer to the derived chromosomes as G_1 and G_2.

I sampled 20X read coverage from each of G_1 and G_2 (100bp reads, uniform 1% error rate) using DWGSIM[1]. The 20X read sets were mixed together into one 40X read set, which simulates random shotgun coverage of a diploid genome. I built an FM-index from the 40X reads using the `sga-bcr` algorithm. Variant calls were made by comparing these reads against the chromosome 20 reference sequence. This is the *reference-based* calling mode of our program - the reference genome serves as the set of control sequences. To assess the performance of the de Bruijn graph haplotype generator (4.2.3) and the string graph haplotype generator (4.2.4), calls were made using both modes. I will refer to these modes as the de Bruijn graph caller and the string graph caller, respectively. To assess the effect of the k-mer parameter, I ran multiple trials with each caller, using k from 33 to 75 in increments of 3. A minimum of 5 variant k-mer occurrences were required to trigger haplotype assembly.

Figure 5.2 plots the sensitivity (true positives / (true positives + false negatives)) and precision ((true positives / (true positives + false positives)) for each caller as a function of k. Here k refers to the variant detection k-mer for the string graph caller and both the variant detection and haplotype assembly k-mer for the de Bruijn graph caller.

[1] https://github.com/nh13/DWGSIM

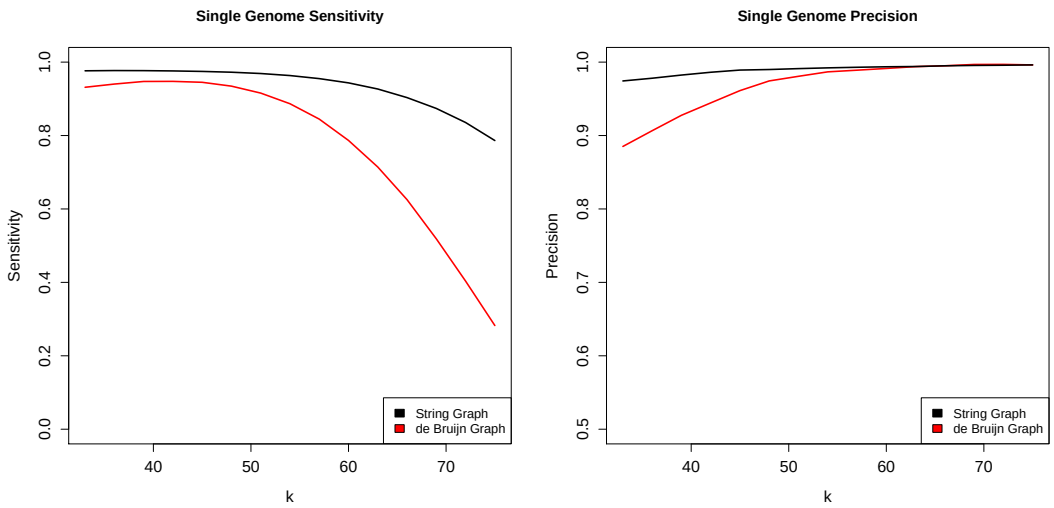

Figure 5.2: Sensitivity (left panel) and precision (right panel) of reference-based calls on simulated data. Note the different range of the y-axis in each panel.

The sensitivity of the de Bruijn graph method decreased sharply with increasing k. This is due to the need to sample a k-mer for every position overlapping the point of variation. When k is large there is insufficient read coverage to ensure that each position has been sampled. This has a weaker effect on the string graph method, as k-mers are only used to detect the variant and a perfect tiling of k-mers is not necessary. Additionally, by performing error correction the string graph caller is able to recover some k-mers that are lost to the de Bruijn graph caller due to sequencing errors.

The peak sensitivity for the de Bruijn method was 0.9476 at $k = 42$. At this k, 96.16% of the homozygous variants were found and 93.34% of the heterozygous variants were found. The increased sensitivity for homozygous variants is expected as they have twice the sequence coverage as heterozygous differences. The sensitivity for indels was slightly higher than that of SNPs (0.9553 vs 0.9469).

The peak sensitivity for the string graph method was 0.9770 at $k = 36$. At $k = 36$, the string graph method recovered 98.50% of the homozygous variants and 96.88% of the heterozygous variants. Like for the de Bruijn graph method, the sensitivity for indels was slightly higher than that of SNPs (0.9821 and 0.9764,

respectively).

At the k-mer chosen to maximize sensitivity the precision of the two methods was 0.9445 (de Bruijn graph) and 0.9781 (string graph). Of the $3,865$ false positive calls for the de Bruijn graph method, 360 (9.3%) are within annotated segmental duplications of the human genome[1]. The string graph method generated $1,520$ false positives at $k = 36$, $1,289$ (85%) of which are in segmental duplications. The lower precision for the de Bruijn graph method is due to the low k-mer used to maximize sensitivity. When a low k-mer is chosen, it is much more likely that a complete bubble forms around sequencing errors. As an illustration of this principle consider the case when k is less than half the read length. When an error occurs in the middle of the read, the erroneous k-mers may be flanked by correct k-mers at the ends of the read. This sequence of k-mers will generate a complete path in the de Bruijn graph between the correct k-mers. When k is high, this situation is much less likely to occur as the sequencing error would need to occur in multiple reads for a complete bubble to form. This effect will be offset to an extent by the requirement that each variant k-mer is seen in at least 5 reads.

In practice the string graph caller with k in the range $50 - 60$ gives better precision (0.9912 to 0.9937) and good sensitivity (0.9690 to 0.9435). For this reason, the default k-mer is set to 54.

5.3.1 Computation Requirements

The de Bruijn graph caller is significantly faster than the string graph caller (10.4 CPU hours versus 24.5 CPU hours, respectively). As both algorithms use the same compressed FM-index, both modes have the same peak memory usage of 1.5GB.

5.4 Simulated genome comparison

Our variant calling model is designed to directly detect variation between two related genomes by directly comparing their sequence reads. I designed a second

[1] as annotated by the UCSC genome browser

simulation to test this method. I started from the pair of chromosomes G_1 and G_2 used in the previous simulation. I generated a new set of substitution variants at frequency 1 in $10,000$bp and indels at 1 in $100,000$bp. These variants were split into two sets and one set was applied to G_1 and one set was applied to G_2 to generate two new genomes G_3 and G_4[1]. I sampled 20X coverage from each of G_3 and G_4 and mixed the reads into one 40X set. Let $\mathcal{R}_A$ be the reads generated from G_1 and G_2 in the previous section and $\mathcal{R}_B$ be the new reads generated from G_3 and G_4 in this section. I made comparative calls using $\mathcal{R}_B$ as the variant sequences and $\mathcal{R}_A$ being the control sequences. Chromosome 20 was used as the reference genome. To trigger assembly, a unique k-mer in $\mathcal{R}_B$ must occur at least 5 times in the $\mathcal{R}_B$ reads and not be present in $\mathcal{R}_A$. Again I used both the de Bruijn graph caller and string graph caller over a range of k.

The results are presented in figure 5.3. The overall trend - that sensitivity decreases as a function k - is similar to the single-genome assessment. At peak sensitivity, the string graph method made slightly more calls (sensitivity 0.9290 at $k = 45$) than the de Bruijn graph method (sensitivity 0.9205 at $k = 36$) and was more accurate (precision 0.9954 vs 0.9752). As in the previous simulation the sensitivity to detect indels was slightly higher (string graph 0.9489 vs 0.9271, de Bruijn graph 0.9425 vs 0.9183).

[1]Note this implies all variants are heterozygous in this simulation

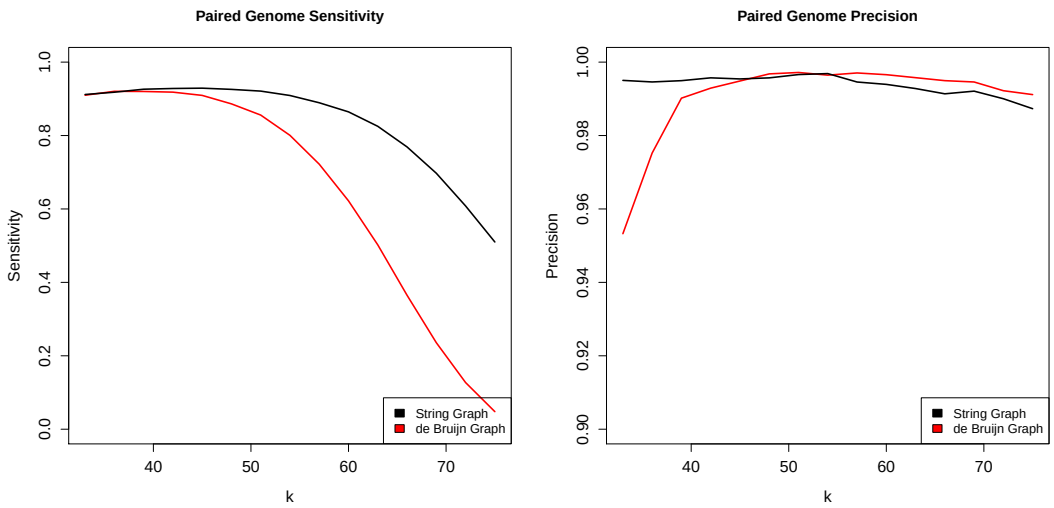

Figure 5.3: Sensitivity (left panel) and precision (right panel) of the simulated genome comparison. Note the different range of the y-axis in each panel.

5.4.1 Computation Requirements

The de Bruijn graph caller required 10.3 CPU hours to make calls at $k = 36$. The string graph caller required 11.4 CPU hours at $k = 45$. Despite having the same number of reads as the reference-based simulation in the previous section both programs were faster in a comparative calling framework. This is particularly true for the string graph caller, which required less than half the time. These results highlight that the number of variants is a crucial determinant of the runtime of the program. The memory usage for both modes was 2.4GB.

5.5 Reference-based Substitution Calls

The results presented above validates that our assembly-based variant calling method can recover the vast majority of simulated SNPs and indels, while retaining high accuracy. Real sequencing data is more challenging however as sequence bias, systematic errors and large structural variants complicate variant calling. To explore the application of our approach to real data, I made reference-based

calls for an individual genome. I used 100bp Illumina sequence data from the NA12878 individual of the CEU population, which has been extensively studied before [Conrad et al., 2011; DePristo et al., 2011; Simpson and Durbin, 2012]. The input data set is available online at `ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/working/20120117_ceu_trio_b37_decoy/`. I used human chromosome 20 as a test case. Reads for this chromosome were extracted from the whole genome BAM file. An FM-index of this subset of reads was created using the `sga-bcr` algorithm. When aligning haplotypes to the reference, I did not limit the alignment to chromosome 20 but rather used the entire reference genome as I found that this helped to reduce the number of false positive variants due to the input reads being mapped to the wrong chromosome. I made two call sets, one using the full set of reads (over 80X coverage) and one using half of the reads. As before, calls were made using both the de Bruijn graph caller and string graph caller.

As the true differences between NA12878 and the reference genome are unknown, I cannot directly evaluate the sensitivity and precision of my variant calls. Instead, I assessed the completeness of my call set by calculating the proportion of mapping-based calls that were found by the assembly callers. The mapping-based calls are from the publication of the GATK variant caller [DePristo et al., 2011]. As a measure of the accuracy of my calls, I compared the calls to variants present in dbSNP v1.32. This build of dbSNP contains variants found by the pilot project of the 1000 Genomes Project which includes NA12878 as a sample. Both of these assessments are restricted to SNP calls. The results are summarized in figure 5.4.

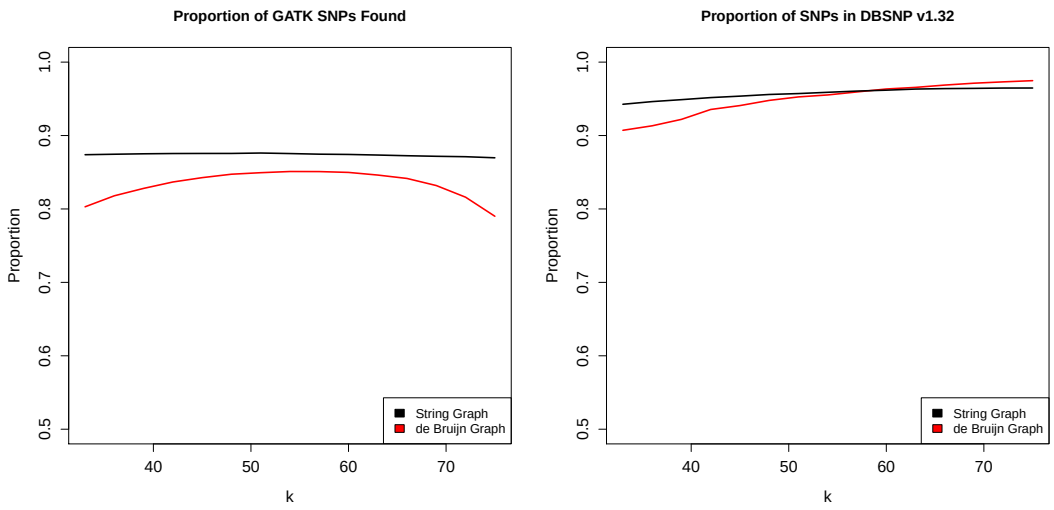

Figure 5.4: The left panel plots the proportion of mapping-based SNP calls using GATK that were found by the de Bruijn graph and string graph callers as a function of k. In the right panel, the proportion of SNP calls that are found in dbSNP v1.32 is plotted.

The peak proportion of mapping SNP calls recovered by the string graph method was 0.8764 at $k = 51$. For the de Bruijn graph method, the peak was 0.8512 at $k = 54$. The performance of the string graph caller was more consistent across the range of k. The proportion of variants found by the de Bruijn graph caller dropped at low and high k-mer values, highlighting the importance of carefully choosing this parameter. Both methods were accurate when assessed by the number of variant calls that are already present in dbSNP v1.32 - 96.88% for the string graph calls ($k = 51$) and 96.74% for the de Bruijn graph calls ($k = 54$) were in dbSNP.

When downsampling the coverage to 42X, the differences between the algorithms become more apparent (figure 5.5). The peak proportion of mapping calls dropped from 0.8764 to 0.8498 for the string graph method ($k = 39$) and 0.8512 to 0.8248 for the de Bruijn graph method ($k = 45$). The profile of the de Bruijn graph caller was similar to the results for simulated data in section 5.3 - there was a steep drop in sensitivity for large k due to lack of coverage. The accuracy was largely unaffected by the decrease in coverage. The proportion of SNPs found

in dbSNP v1.32 for the string graph method at $k = 39$ was 97.25%. For the de Bruijn graph method the dbSNP proportion was 96.87% at $k = 45$.

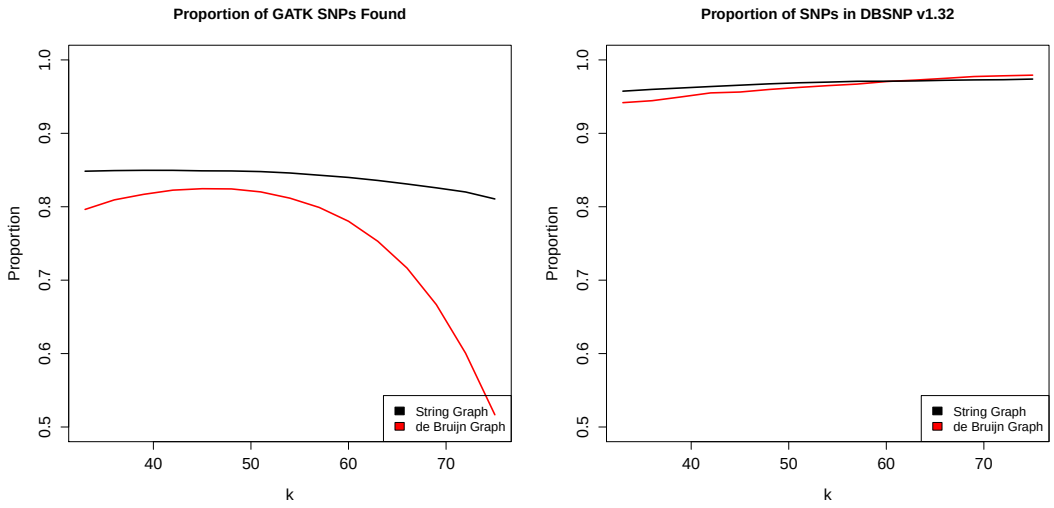

Figure 5.5: The proportion of mapping-based SNPs found (left) and the proportion of our SNP calls contained in dbSNP v1.32 (right) for the downsampled data set.

To further characterize the performance of my assembly based variant caller, I investigated the variant calls that were found by the mapping-based caller (GATK) but not called by the assembly-based callers. In total, GATK called 75,838 substitutions on chromosome 20. Of these, 42,686 (56.3%) are in known segmental duplications or repetitive elements masked by RepeatMasker[1]. The string graph caller ($k = 51$) did not find 9,391 GATK calls in the high depth (>80X) data set. Of these 8,093 (86.2%) lie within annotated repeats. Similarly the de Bruijn graph caller ($k = 54$) missed 11,300 variants, 9,714 (86.0%) of which are in known repeats. This suggests that our ability to make assembly calls is higher in non-repetitive regions of the genome. This is expected as repetitive regions will lead to a more complicated assembly graph making it less likely that clean haplotypes can be assembled. It is also possible that the false positive rate of the mapping caller is higher in these difficult regions of the genome.

[1] Annotations were downloaded from the UCSC genome browser

5.6 Estimating the background error rate for comparative variant calling

I used the NA12878 sequence data to estimate the false positive rate of our comparative variant caller. I split the NA12878 chromosome 20 reads into two subsets of approximately 40X each by randomly assigning each read pair to one of two files. I will refer to these two halves as $\mathcal{H}_1$ and $\mathcal{H}_2$. I made comparative calls by using $\mathcal{H}_1$ as the variant sequences and $\mathcal{H}_2$ as the control sequences. As all the reads were drawn from the same individual no variants should be called - all variants found by this procedure are false positives either due to sequencing errors, assembly errors or incorrectly aligning the assembled haplotypes to the reference genome. As before, I ran the caller in both the string graph mode and de Bruijn graph mode over a range of k. I required 5 occurrences of a k-mer to trigger variant assembly. I classified the errors into three categories - substitutions, indels in homopolymer sequences (a string of ≥ 7 or more occurrences of the same base) or indels outside of homopolymers. The results are summarized in table 5.1. The number of false positive calls drops sharply with increasing k. In all cases, the majority of false positive calls are due to mis-calling the length of a long homopolymer run. This is likely due to the increased sequencing error rate associated with these regions [Albers et al., 2011; Li, 2012].

Table 5.1: False positive variant calls found by splitting the NA12878 chromosome 20 data into two halves. The variants are classified into substitutions (Subs), indels outside of homopolymer runs (non-HP indels) and indels within homopolymer runs (HP indels).

	de Bruijn Graph Calls			String Graph Calls		
k	Subs	Non-HP indels	HP indels	Subs	Non-HP indels	HP indels
33	41	6	118	15	2	48
36	24	6	112	13	4	50
39	20	7	97	15	3	48
42	18	6	83	16	2	45
45	8	4	71	16	2	46
48	5	3	55	16	1	47
51	11	2	39	12	1	44
54	3	2	31	14	1	39
57	1	2	15	10	1	30
60	2	2	6	10	1	22
63	2	0	4	7	1	18
66	0	1	2	6	1	17
69	0	1	2	7	2	9
72	0	0	0	2	2	5
75	0	0	0	2	3	3

5.7 Calling *de novo* mutations in a trio

I will now describe the application of our comparative variant caller to real sequencing problems. The first problem I will address is the discovery of *de novo* mutations. These are mutations that occur in the germline of an individual's parents and are subsequently passed along to the child. *De novo* mutations have been implicated in a number of human diseases including schizophrenia [Girard et al., 2011] and autism [Sanders et al., 2012]. To find *de novo* mutations the genome of a child is sequenced along with the genome of both of its parents (this is commonly referred to as a sequencing a "trio"). Conrad et al. [2011] developed an algorithm to call *de novo* mutations using reads mapped to a reference genome. Their framework considers the three individuals jointly in a Bayesian

framework[1].

I used our assembly-based approach to call *de novo* mutations in a trio from the CEU population, which was also studied by Conrad et al. [2011] using lymphoblastoid cell line DNA. It is known there are many somatic cell line mutations in this sample. The individual NA12878 used in section 5.5 is the child in this trio. Conrad et al. used early Illumina sequencing data from the pilot of the 1000 Genomes Project. In this section, I use more recent data consisting of 101bp reads[2]. In this data set, NA12878 was sequenced to 81X depth. The parents were sequenced to 71X (identifier NA12891) and 70X (NA12892). I used the read set of the child, $\mathcal{R}_c$, as the variant sequences and the union of the read sets from the two parents, $\mathcal{R}_p$, as the control set. For computational convenience, I made calls chromosome-by-chromosome by taking reads mapped to the reference genome and separating them into subsets based on the chromosome the reads mapped to. While this introduces a weak bias towards the reference genome, I believe this drawback is offset by the reduction in run time and memory usage. I used $k = 54$ when making calls and required 5 occurrences of a variant k-mer to trigger assembly. Candidate haplotypes were mapped to the full reference genome.

In Conrad et al.'s original paper, they selected a large number of calls for experimental validation. The selected calls were validated by PCR amplification followed by Illumina sequencing or target enrichment followed by SOLiD sequencing. As the first measure of the performance of my software on calling *de novo* mutations, I compared our calls to the successfully validated mapping calls. The results are presented in table 5.2. In total, the de Bruijn graph caller found 908 of the 936 (97.0%) of the validated subset of calls. The string graph caller found 898 of 936 (95.6%). While these results are encouraging, it is worth noting that the calls selected for validation were filtered to avoid difficult regions of the genome. Mapping-based calls in simple repeats, known copy number variants, segmental duplications or in dbSNP v1.29 were excluded from this validation set. Additionally, sites without read coverage in all three individuals or near a short

[1] Details can be found in [Conrad et al., 2011]

[2] ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/working/20120117_ceu_trio_b37_decoy/

polymorphic indel were removed. In total, these filters excluded 468 Mbp of the genome.

Table 5.2: Chromosome by chromosome breakdown of the number of de novo substitution calls in the validation set and for the de Bruijn (DBG) and string graph (SG) callers.

Chr.	Validated	DBG Calls	Found by DBG (%)	SG Calls	Found by SG (%)
1	51	184	51 (100.0%)	199	50 (98.0%)
2	63	235	63 (100.0%)	267	61 (96.8%)
3	71	201	71 (100.0%)	219	70 (98.6%)
4	100	252	97 (97.0%)	276	99 (99.0%)
5	92	187	92 (100.0%)	213	92 (100.0%)
6	64	176	60 (93.8%)	190	58 (90.6%)
7	64	115	60 (93.8%)	118	59 (92.2%)
8	75	155	73 (97.3%)	165	72 (96.0%)
9	58	128	56 (96.6%)	146	57 (98.3%)
10	47	113	45 (95.7%)	131	45 (95.7%)
11	27	128	26 (96.3%)	144	26 (96.3%)
12	17	89	16 (94.1%)	95	15 (88.2%)
13	31	119	30 (96.8%)	128	30 (96.8%)
14	31	121	29 (93.5%)	129	28 (90.3%)
15	28	94	28 (100.0%)	104	28 (100.0%)
16	23	63	21 (91.3%)	73	18 (78.3%)
17	20	49	20 (100.0%)	64	20 (100.0%)
18	33	80	30 (90.9%)	89	30 (90.9%)
19	8	97	7 (87.5%)	114	7 (87.5%)
20	22	62	22 (100.0%)	69	22 (100.0%)
21	8	24	8 (100.0%)	38	8 (100.0%)
22	3	27	3 (100.0%)	31	3 (100.0%)
total	936	2699	908 (97.0%)	3002	898 (95.9%)

The assembly-based callers found thousands of substitutions that are not present in the validation set. To get a more complete measure of the performance of the assembly caller, I compared the assembly calls to the raw output of

Conrad's caller, DeNovoGears. Here I did not use the same DeNovoGears callset as in [Conrad et al., 2011] but rather I used an updated call set using the most recent version of the program (v0.3) on the recent 101bp sequencing data [1]. I parsed the raw DeNovoGear output to remove SNP calls that had a posterior probability of being a *de novo* mutation less than 0.75. After this filter, 4488 calls remained. 2006 of the 2699 de Bruijn graph SNP calls (74.3%) are found in this set of DeNovoGears calls. For the string graph caller, 2003 of 3002 (66.7%) are found in the DeNovoGear set. This suggests that the majority of the assembly substitution calls are true *de novo* mutations, not false positives.

The de Bruijn graph caller made 2,157 indel calls. After filtering out indels that occur in homopolymers of length 7 or greater, 245 indels remain. The String Graph caller made 2,795 indel calls, 321 of which are not in homopolymer runs. In both call sets the non-homopolymer events are biased towards deletions. In the de Bruijn graph call set the ratio of deletions to insertions is 4.3:1. In the string graph call set the ratio is 3.2:1.

5.8 Cancer mutations

As a second test of our comparative assembly algorithm, I called mutations in a human breast cancer. A typical cancer sequencing experiment sequences a tumor along with the individual's matched normal genome. Variants found only in the tumor are putative *somatic* mutations. In this test, I used a breast cancer sample sequenced at the Sanger Institute as part of the Cancer Genome Project. The tumor read set consists of 1.65 billion 100bp reads (55X). The matched normal genome has 1.32 billion reads (44X). This data set was recently used as part of a large project to catalog mutations [Nik-Zainal et al., 2012a] and mutational history [Nik-Zainal et al., 2012b] in 21 breast cancers. Finding cancer mutations is a more difficult use case than other applications as tumors typically exhibit subclonal structure and some mutations are found only in a subset of tumor cells. In addition the tumor is typically not an entirely pure sample and is contaminated with normal tissue. For this reason, some mutations will be covered by few reads.

[1] These calls are provided by Art Wuster of the Hurles lab

To account for this I used a lower number of required k-mer occurrences to trigger assembly, 3. As in the trio variant calling, I used $k = 54$.

In the framework of my comparative variant caller the reads from the tumor, $\mathcal{R}_T$, are the variant reads, and the reads from the normal, $\mathcal{R}_N$ are the control set. As in the trio experiment I made calls chromosome-by-chromosome. As part of the Cancer Genome Project's standard pipeline they call somatic mutations from the mapped reads using in-house software. In table 5.3 I compare CGP's mapping calls to the assembly calls. Of the $10,381$ calls found by CGP's mapping based caller, the de Bruijn graph caller found 8,035 (77.4%). The string graph caller found $8,593$ (82.8%). There is a noteworthy excess of substitutions on chromosome 6. Closer inspection revealed a dense cluster of C>T transitions on this chromosome. These events occurred primarily in a TpC context (TC>TT substitution). As these C>T events are in close proximity, they often assemble into a single haplotype. Using the string graph calls as an example, the most divergent chromosome 6 haplotype assembled had 35 C>T mutations. Nik-Zainal et al. studied this hypermutation phenomenon in detail in [Nik-Zainal et al., 2012a].

Table 5.3: Chromosome-by-chromosome breakdown of the substitutions called in the breast cancer tumor.

Chr.	Mapping Calls	DBG Calls	Found by DBG (%)	SG Calls	Found by SG (%)
1	783	807	612 (78.2%)	973	647 (82.6%)
2	849	827	660 (77.7%)	946	699 (82.3%)
3	632	590	489 (77.4%)	678	537 (85.0%)
4	645	613	502 (77.8%)	678	536 (83.1%)
5	480	438	366 (76.2%)	482	393 (81.9%)
6	1286	1262	1050 (81.6%)	1313	1082 (84.1%)
7	837	921	661 (79.0%)	1096	717 (85.7%)
8	459	457	352 (76.7%)	519	377 (82.1%)
9	296	309	222 (75.0%)	357	241 (81.4%)
10	519	499	399 (76.9%)	592	422 (81.3%)
11	429	431	329 (76.7%)	491	349 (81.4%)
12	391	419	311 (79.5%)	490	330 (84.4%)
13	247	224	186 (75.3%)	249	204 (82.6%)
14	186	178	143 (76.9%)	215	157 (84.4%)
15	192	185	146 (76.0%)	220	157 (81.8%)
16	260	282	194 (74.6%)	379	211 (81.2%)
17	181	205	113 (62.4%)	241	136 (75.1%)
18	379	396	316 (83.4%)	463	328 (86.5%)
19	128	144	81 (63.3%)	213	93 (72.7%)
20	228	318	176 (77.2%)	387	178 (78.1%)
21	198	197	151 (76.3%)	240	163 (82.3%)
22	95	107	66 (69.5%)	169	76 (80.0%)
X	681	661	510 (74.9%)	701	560 (82.2%)
total	10381	10470	8035 (77.4%)	12092	8593 (82.8%)

The Cancer Genome Project validated 309 of the substitution calls made by their mapping-based caller. Of these 309, 255 were found by the de Bruijn graph caller (82.5%) and 270 (87.4%) were found by the string graph caller.

The de Bruijn graph caller made $4,499$ indel calls, 974 of which are not in a homopolymer run. For the string graph caller $4,510$ indel calls were made, $1,104$ outside of homopolymers. The Cancer Genome Project called indels on

this sample using Pindel [Ye et al., 2009], 333 of which were validated. Of the 333 validated indels, 297 (89.2%) were found by the de Bruijn graph caller and 293 (88.0%) were found by the string graph caller[1]. A number of the assembly indel calls are near a CGP validated indel but did not have the exact same sequence. If I relax the matching criteria to only require the assembly call to be within 20bp of the CGP event, the number of matching events increases to 320 for the de Bruijn graph caller (96.1%) and 319 for the string graph caller (95.8%)[2]. The reasons that the breakpoint sequences differ in these cases remains to be investigated.

Our probabilistic realignment method estimates the allele frequency of variants. In the context of cancer sequencing, this is an estimation of the proportion of chromosomes present in the entire tumor sample (including non-cancerous contaminating tissue) that harbors a particular mutation. Figure 5.6 plots the distribution of allele frequencies for the substitutions called by our string graph method. If mutations occurred on one of the two chromosomes at random and the mutated chromosome was present in all cells of the tumor, we would expect the allele frequencies to be distributed around 0.5. However as cancers continuously accumulate mutations as they evolve, all cells will not contain every mutation. Additionally, contamination by normal tissue will shift the allele frequency distribution towards lower frequency. These effects are shown in figure 5.6 as the median allele frequency is 0.225.

[1] Homopolymer indels were included in this analysis
[2] Calculated with BEDTools' intersectBED program

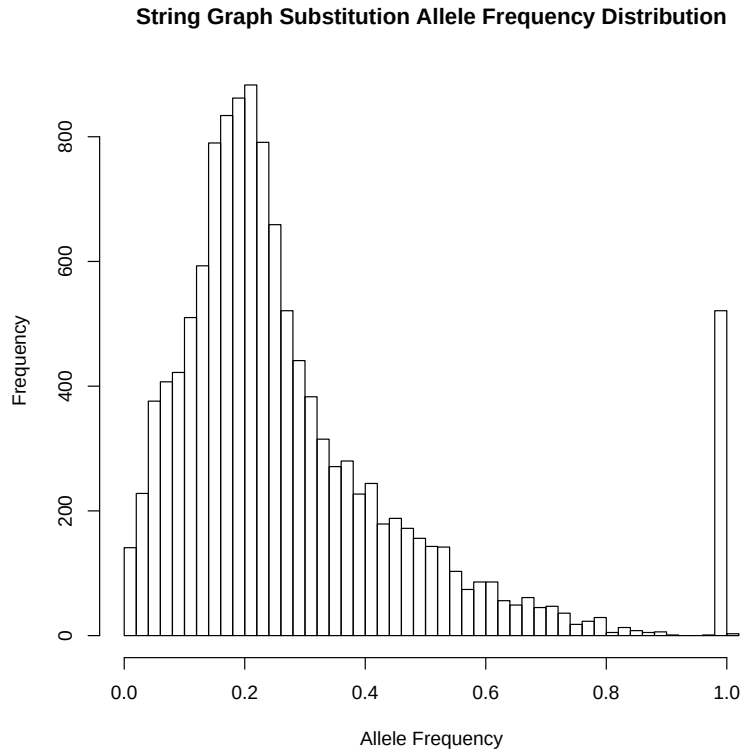

Figure 5.6: The allele frequency distribution for substitution calls made by the string graph caller

To assess the accuracy of our allele frequency estimates, I compared our estimates to those made by CGP's mapping-based pipeline. The allele frequency estimates for variants common to both call sets is plotted in figure 5.7. Our allele frequency estimates are well-correlated to those of the mapping based caller ($r = 0.957$).

The string graph substitution call set contains 447 substitutions with estimated allele frequency 1.0. Few of these calls are also contained in the CGP call set (figure 5.7). Of the 447 high allele frequency substitution calls, 376 are found in dbSNP v1.32. This suggests these calls are probably homozygous SNPs that are incorrectly called as somatic mutations. The likely cause for these false positives is poor sequence coverage of the sites in the matched normal sample.

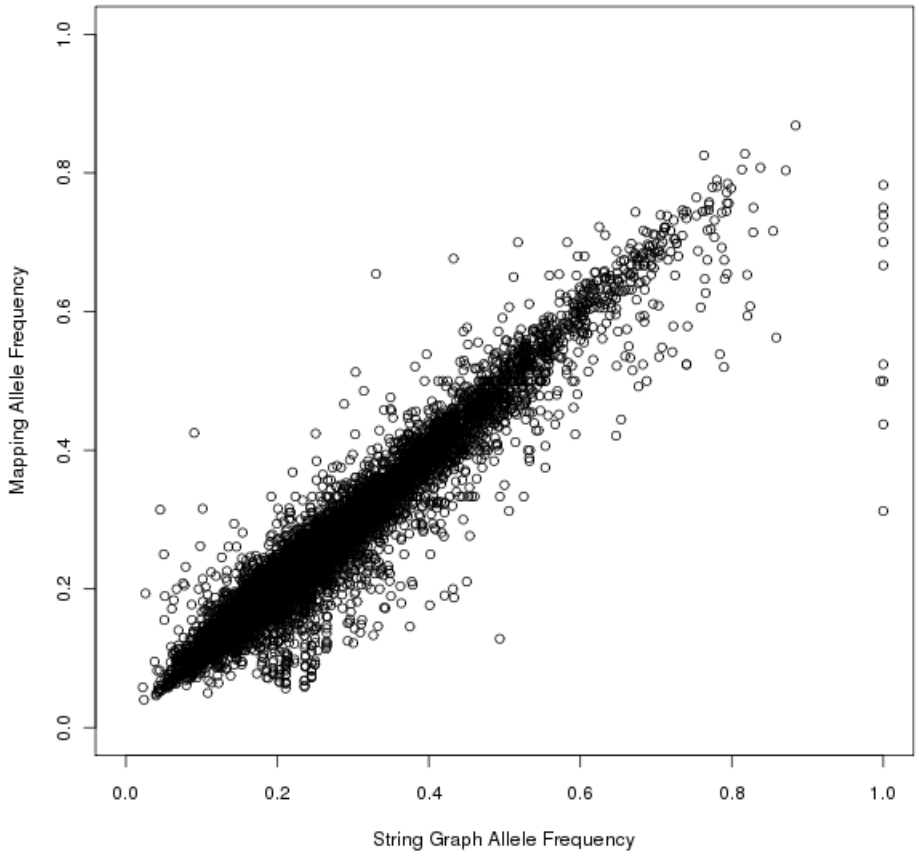

Figure 5.7: The allele frequency calculated by the string graph caller (x-axis) and CGP (y-axis) for calls made a common sites

Finally, I assessed the functional consequences of the string graph calls. Using the Variant Effect Predictor provided by Ensembl [Flicek et al., 2012; McLaren et al., 2010], I predicted the effect of all substitution mutations and all non-homopolymer indels. Ensembl release version 66 was used. My mutation call set had 3 frameshift mutations, including a single base deletion in the important tumor suppressor *TP53*. This variant was also found by the CGP and experimentally validated. Two substitution mutations generated new stop codons. There are 47 non-synonymous coding mutations and 32 synonymous changes.

5.8.1 Analysis Notes

The mapping-based variants, their estimated allele frequencies and validation status were provided by Serena Nik-Zainal of the Cancer Genome Project. The CGP variant calls were made by Caveman, an in-house caller.

5.9 Low-Coverage Population Calls

Finally, I used the assembly-based variant caller on low-coverage human population sequencing. The data used is from Phase 2 of the 1000 Genomes Project [1000 Genomes Project Consortium, 2010]. I used all reads mapping to chromosome 20 for the African continental group (LWK, YRI, ASW, ACB populations). Only individuals that had 75bp reads or greater were included. This subset of the data contains 191 individuals. I used the de Bruijn graph caller for this data set with a k-mer size of 61. Five occurrences of a k-mer were required to trigger assembly and five occurrences of a k-mer were required to use it in the de Bruijn graph (m parameter in `generateDeBruijnHaplotypes`).

The de Bruijn graph caller found 218,852 single nucleotide polymorphisms, 35,846 indels and 2,246 multi-nucleotide polymorphisms[1]. To assess the accuracy of my call set, I calculated the transition/transversion ratio of the SNP variants and the proportion of variants that were previously found. For completely random mutations in random sequence the transition/transversion ratio (Ti/Tv) would be 1:2. In actual sequence however transitions are more likely to occur [Wakeley, 1996]. The transition/transversion ratio of the chromosome 20 calls for African samples in phase 1 of the 1000 Genomes Project is 2.37. The transition/transversion ratio of my call set is 2.20:1. To assess the novelty of my calls, I compared the SNP calls to dbSNP v1.32, which contains calls for the pilot data of the 1000 Genomes Project. 89.68% of my SNP calls are known variants.

In addition to SNPs and MNPs, I called 35,846 indels. To assess the accuracy of my indel calls, I calculated the ratio of in-frame indels (those that do not change the reading frame of protein translation) versus the number of frameshift mutation. As it is expected that frameshift mutations are significantly damaging

[1] Block substitutions of length > 1

to protein function, very few frameshift mutations are expected. My call set contains 14 in-frame and 14 frameshift indels.

Our population caller estimates genotype likelihoods for each sequenced individual and uses these likelihoods to estimate allele frequencies in the population. The allele frequency distribution for the de Bruijn graph SNP and indel calls is presented in figure 5.8. As the assembly based caller requires significant read coverage of each variant sequence to successfully assemble it into a haplotype, we have reduced power to detect low-frequency variants (allele frequency $< 5\%$).

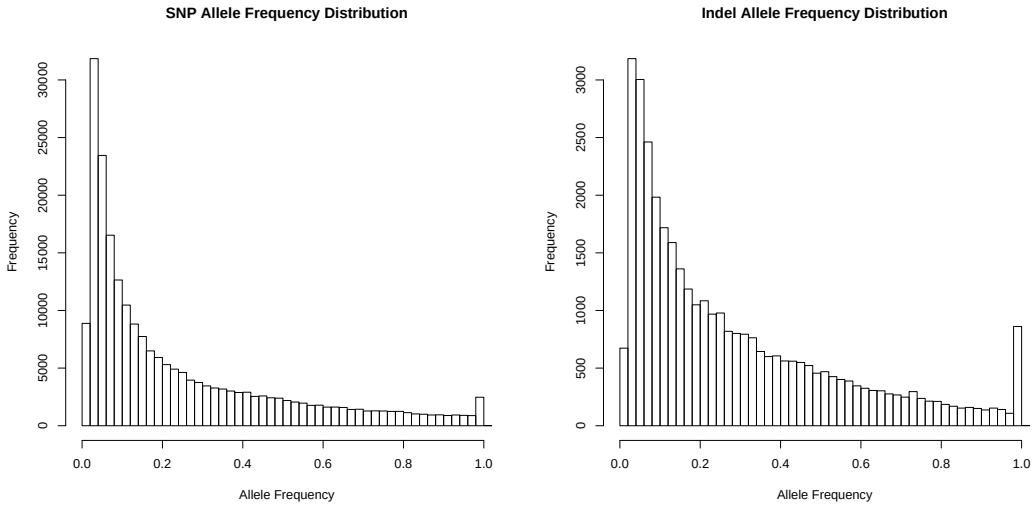

Figure 5.8: The allele frequency distribution for SNP and Indel calls on the AFR continental group of the 1000 Genomes Project

I also compared my indel calls to the mapping-based indel calls from phase 1 of the 1000 Genomes Project[1]. The mapping-based calls were made from 1,094 individuals. I made a subset of the phase 1 calls consisting of calls on chromosome 20 that are not contained in the "excluded" calls file[2]. The mapping-based indel

[1] The calls were downloaded from ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/phase1/analysis_results/consensus_call_sets/indels/ALL.wgs.VQSR_V2_GLs_polarized.20101123.indels.low_coverage.sites.vcf.gz

[2] ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/phase1/analysis_results/supporting/excluded_indel_sites/ALL.wgs.excluded_sites_20120312.20101123.indels.sites.vcf.gz

call set contains few calls of length greater than 20bp. Despite using far fewer samples, the assembly call set contains many more large indels, demonstrating the benefit of assembly approaches for finding complex variation (figure 5.9).

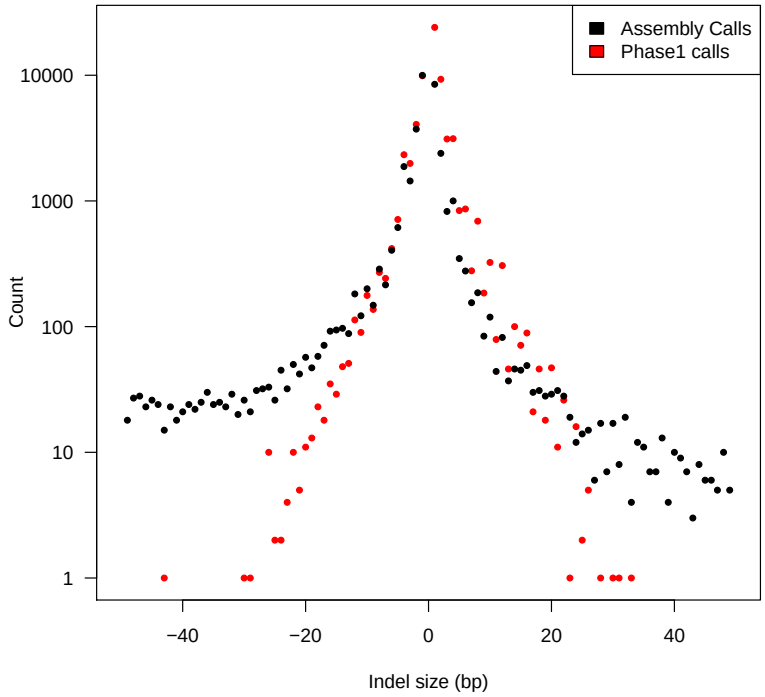

Figure 5.9: The distribution of insertion (positive) and deletion (negative) lengths for the 1000 Genomes data set. The data set consists of 35,846 *assembly indel calls (black points) and* 64,319 *mapping calls from Phase 1 of the 1000 Genomes Project (red points). Events larger than 50bp were excluded from this plot.*

5.9.0.1 Computation Requirements

Constructing the FM-index for the population required 162 CPU hours (59 hours elapsed time). The peak memory usage during index construction was 26GB. Variant calling required 359 CPU hours. Variant calling was run using 16 computation threads, which allowed the task to complete in 29 wall-cloock hours.

The peak memory usage during variant calling was 39GB.

5.10 Discussion

In this chapter, I explored the properties of our assembly-based approach to variant calling. On simulated data, the assembly-based caller recovered the majority of variants while retaining high accuracy. For real data, some power is clearly lost when compared to mapping-based approaches. It is an open question of how many of the "missed" mapping-based variants are true SNPs or indels and how many are false positives. Assembly-based calling requires higher coverage than mapping so it is expected that some true variants will be missed due to insufficient sequence depth. Likewise, we do not yet use read pairs in our haplotype generation functions. This may lead to a loss of power in difficult to assemble regions, which is reflected by the fact that most of the GATK SNPs that we missed in NA12878 are found in annotated repeats. Despite these limitations, the assembly-based approach is promising. The assembly-based caller found most of the validated *de novo* mutations in the trio and validated indels in the cancer sample. The indel size distribution on the 1000 Genomes data suggests the assembly caller has better representation of large events when compared to mapping-based approaches.

Assembly-based variant calling is a new technique. Cortex [Iqbal et al., 2012] and Fermi [Li, 2012] were published this year - the algorithms described in this work were developed in parallel. I did not directly compare to Cortex and Fermi due to the practicalities of running these programs on the range of data sets presented here. A comparison and assessment including Cortex, SGA and state of the art mapping and local reassembly methods is underway for phase 2 of the 1000 Genomes Project. This upcoming assessment should help demonstrate the pros and cons of assembly based approaches.