# Supplementary Information

## Comprehensive variation discovery in single human genomes

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 & David B. Jaffe

## Table of contents

**Supplementary Data Set 1** provides a tab-delimited list of the variants called in the Fosmid regions, showing the Fosmid identifier, position, reference allele, alternate allele, and a comma-separated list of notes. The notes are either 'Fosmid', indicating that the variant is present in in the Fosmid reference set, or of the form caller=state, where caller is Platinum-100, GATK-250, DISCOVAR or CORTEX, and state is het, hom or hom+. Here het means that both the variant and reference were called, hom means that the variant but not the reference was called, and hom+ means that the variant was called as homozygous. **Supplementary Data Set 1** was automatically generated. See also the manually identified defects noted in **Supplementary Table 1a**.

## Supplementary Table 1a. Mis-accounting of variant calls

| Fosmid id | position on genome | call set | ΔFN | ΔFP (hom) |
|---|---|---|---|---|
| 2 | 1:54782327-54782350 | GATK-250 | 2 | 0 |
| 4 | 2:106813906-106813929 | GATK-250 | 5 | 0 |
| 9 | 5:177729238-177729392 | GATK-250 | 4 | 0 |
| 11 | 5:179256683-179256705 | Cortex | 3 | 0 |
| 11 | 5:179256683-179256705 | DISCOVAR | 3 | 0 |
| 11 | 5:179256683-179256705 | GATK-250 | 3 | 0 |
| 24 | 11:64997532-64997579 | GATK-250 | 2 | 2 |
| 28 | 12:3160304-3160316 | GATK-250 | 3 | 0 |
| 32 | 12:113990053-113990079 | GATK-250 | 2 | 1 |
| 41 | 16:24324276-24324280 | GATK-250 | 2 | 0 |
| 44 | 17:20382902-20382948 | GATK-250 | 3 | 1 |
| 44 | 17:20382903-20382947 | DISCOVAR | 1 | 1 |
| 49 | 19:18996324-18996331 | Platinum-100 | 1 | 1 |
| 59 | 1:78498169-78498254 | DISCOVAR | 1 | 4 |
| 66 | 2:106507988-106508299 | GATK-250 | 6 | 1 |
| 69 | 3:12864779-12864786 | GATK-250 | 2 | 0 |
| 69 | 3:12864779-12864786 | Platinum-100 | 2 | 2 |
| 73 | 3:56240208-56240234 | Platinum-100 | 2 | 1 |
| 73 | 3:56240208-56240234 | GATK-250 | 2 | 1 |
| 75 | 4:6100447-6100452 | GATK-250 | 2 | 0 |
| 76 | 4:57809041-57809063 | GATK-250 | 2 | 0 |
| 77 | 5:34846772-34846812 | DISCOVAR | 1 | 2 |
| 82 | 7:148977164-148977175 | GATK-250 | 2 | 0 |
| 83 | 9:83608432-83608508 | GATK-250 | 5 | 0 |
| 85 | 9:130950395-130950396 | DISCOVAR | 1 | 2 |
| 92 | 12:97617113-97617190 | GATK-250 | 1 | 0 |
| 96 | 17:11030088-11030107 | GATK-250 | 4 | 1 |
| 101 | 19:36714587-36714592 | Platinum-100 | 2 | 2 |
| 105 | 22:45817169-45817215 | Platinum-100 | 3 | 2 |
| 105 | 22:45817169-45817215 | Cortex | 2 | 1 |
| 105 | 22:45837495-45837512 | GATK-250 | 2 | 1 |

**Supplementary Table 1a. Mis-accounting of variant calls.** All variant calls were examined to identify instances where an alternate representation of a complex variant resulted in an incorrect increase in false negatives or homozygous false positives. This table contains all such instances. Fosmid id: identifier of Fosmid. Position on genome : position of event on hg19 reference sequence. Call set: the call set in which the mis-accounting occurs. ΔFN: the amount by which the false negative count should be reduced, and ΔFP (hom) the amount by which the homozygous false positive rate should be reduced, to remedy the accounting defect. Note that we automatically identified instances where a set of homozygous variants made by a caller were equivalent to a set of Fosmid reference variants, and in such cases translated the first set into the second. Such instances are not shown in the table.

## Supplementary Table 1b. Mis-accounting of variant calls (summary)

| call set | ΔFN | ΔFN% | ΔFP (hom) | ΔFP (hom)% |
|---|---|---|---|---|
| Platinum-100 | 10 | 0.22 | 8 | 0.38 |
| GATK-250 | 54 | 1.20 | 8 | 0.33 |
| Cortex | 5 | 0.11 | 1 | 0.01 |
| DISCOVAR | 7 | 0.16 | 9 | 0.35 |

**Supplementary Table 1b. Mis-accounting of variant calls (summary).** This table provides totals corresponding to **Supplementary Table 1a**. ΔFN: sum of ΔFN entries. ΔFN%: same, as percent of total Fosmid calls (from **Supplementary Table 6**). ΔFP (hom): sum of ΔFP (hom) entries. ΔFP (hom)%: same, as percent of homozygous Fosmid calls for the call set (from **Supplementary Table 6**).

## Supplementary Table 2. Effect on GATK sensitivity and specificity of changing variant calling filter

| caller | read length | filter | %FN | %FP | |
|---|---|---|---|---|---|
| | | | | heterozygous variants | homozygous variants |
| GATK | 100 | PASS | 25.2 ± 2.5 | 0.83 ± 0.07 | 1.23 ± 0.26 |
| | | VQSR ≥ 99.9% | 21.0 ± 2.1 | 6.31 ± 1.76 | 1.27 ± 0.25 |
| | | VQSR ≥ 99% | 15.2 ± 2.1 | 10.9 ± 1.69 | 1.42 ± 0.26 |
| GATK | 250 | PASS | 13.5 ± 1.8 | 1.82 ± 0.45 | 1.07 ± 0.72 |
| | | VQSR ≥ 99.9% | 13.4 ± 1.8 | 2.66 ± 0.76 | 1.07 ± 0.72 |
| | | VQSR ≥ 99% | 13.4 ± 1.8 | 2.68 ± 0.76 | 1.07 ± 0.72 |

**Supplementary Table 2. Effect on GATK sensitivity and specificity of changing variant calling filter.** This table mirrors **Table 2**, however we use three different filters here. PASS: same as in **Table 2**, only calls labeled PASS are included. VQSR ≥ 99.9%: calls labeled either PASS or flagged in this way are included. VQSR ≥ 99%: calls labeled either PASS or flagged in this way are included. Calls labeled Low… were excluded. Note that the values in this table do not include the *manual* corrections of **Supplementary Tables 1a,b**, because such corrections would have to be computed separately for each of the six rows in this table.

## Supplementary Table 3. HapMap3 sensitivity of call sets

| **call set** | **FN%** |
|---|---|
| Platinum-100 | 2.5 |
| GATK-250 | 1.8 |
| Cortex | 7.9 |
| DISCOVAR | 1.5 |

**Supplementary Table 3. HapMap3 sensitivity of call sets.** We consider the set of SNPs in HapMap3 release 27 that are genotyped in HapMap3 as present in NA12878, exclusive of sites reported as triallelic. For each of four call sets, we report sensitivity relative to this set. FN%: percent of this set that is absent in the given call set (regardless of whether the SNPs are reported as heterozygous or homozygous). We note that some of the false negative SNPs appear to adjacent to indels, and thus may be represent artifacts arising from the HapMap3 methodology.

## Supplementary Table 4. Percent of false positive variant calls by sequence type

| variant type | sequence type | Platinum-100 | GATK-250 | Cortex-250 | DISCOVAR-250 |
|---|---|---|---|---|---|
| homozygous | low complexity | 94 | 89 | 67 | 80 |
| | segmental duplication | 11 | 6 | 0 | 8 |
| heterozygous | low complexity | 57 | 15 | 4 | 24 |
| | segmental duplication | 4 | 8 | 1 | 34 |

**Supplementary Table 4. Percent of false positive variant calls by sequence type.** For each variant type (homozygous, heterozygous) and each of the four call sets of this work, we determine the percent of false positive variant calls that lie in low complexity sequence or in segmental duplications. Both categories are defined in **Table 1**. They overlap slightly.

## Supplementary Table 5. Representation of disease triplets in variant call sets

| | | | | Triple run multiplicity | | | | |
|---|---|---|---|---|---|---|---|---|
| Abbr | Disease | Triplet | Triplet start | Ref | Truth | Platinum-100 | GATK-250 | DISCOVAR-250 |
| DM1 | Myotonic dystrophy type 1 | CTG | 19:46273463 | 20 | 5,13 | 20,20 | 5,13 | 5,13 |
| DRPLA | Dentatorubropallidoluysian atrophy | CAG | 12:7045892 | 15 | 15,15 | 15,15 | 15,15 | 15,15 |
| FRDA | Friedreich's ataxia | GAA | 9:71652203 | 6 | 9,9 | 6,6 | 9,9 | 9,9 |
| HD | Huntington's disease | CAG | 4:3076604 | 19 | 16,18 | 19,19 | 19,19 | 16,18 |
| HDL2 | Huntington's disease-like 2 | CTG | 16:87637894 | 14 | 14,16 | 14,14 | 14,16 | 14,16 |
| SBMA | Spinal-bulbar muscular atrophy | CAG | X:66765160 | 22 | 20,24 | 22,22 | 20,24 | 20,24 |
| SCA1 | Spinocerebellar ataxia type 1 | CAG | 6:16327918 | 12 | 12,12 | 12,12 | 12,12 | 12,12 |
| SCA2 | Spinocerebellar ataxia type 2 | CAG | 12:112036785 | 13 | 8,13 | 13,13 | 8,13 | 8,13 |
| SCA3 | Spinocerebellar ataxia type 3 | CAG | 14:92537355 | 8 | 17,18 | 8,8 | 17,18 | 17,18 |
| SCA6 | Spinocerebellar ataxia type 6 | CAG | 19:13318673 | 13 | 11,12 | 13,13 | 11,13 | 11,12 |
| SCA7 | Spinocerebellar ataxia type 7 | CAG | 3:63898362 | 10 | 10,12 | 10,10 | 10,10 | 10,12 |
| SCA8 | Spinocerebellar ataxia type 8 | CTG | 13:70713516 | 15 | 15,16 | 15,15 | 15,16 | 15,16 |
| SCA12 | Spinocerebellar ataxia type 12 | CAG | 5:146258292 | 10 | 10,14 | 10,10 | 10,14 | 10,14 |
| SCA17 | Spinocerebellar ataxia type 17 | CAG | 6:170870996 | 3 | 3,3 | 3,3 | 3,3 | 3,3 |

**Supplementary Table 5. Representation of disease triplets in variant call sets.** For 14 diseases associated with base triplet expansions[19], excluding 100% GC triplets, we assayed the true genotype of the triplet run lengths in NA12878 and its representation in each of three call sets. Abbr: abbreviation for disease. Disease: name of disease. Triplet: sequence of triplet in direction of transcription of gene. Triplet start: hg19 start of triple run, from http://www.ncbi.nlm.nih.gov/clinvar. Run lengths are computed as ***the number of copies of the exact triplet starting at the given triple start***. Ref: multiplicity of triplet run in hg19 reference sequence. Truth: true multiplicity genotype of triplet run in NA12878, determined by tabulating the number of exact triplet runs present in the reads and having the correct flanking sequences, using both NA12878 datasets of this work. Platinum-100, GATK-250, DISCOVAR-250: multiplicity genotype inferred from each of three variant call sets. Erroneous genotypes are shown in gray.

## Supplementary Table 6. Raw data for false negative and false positive rates

| | Fosmid calls | | Homozygous non-Fosmid calls | Heterozygous calls | |
|---|---|---|---|---|---|
| | all | homozygous | | Male X | Female X |
| Platinum 100 base reads | 3357 | 2110 | 26 | 636 | 42811 |
| GATK 250 base reads | 3879 | 2438 | 26 | 1760 | 53688 |
| Cortex 250 base reads | 2720 | 1821 | 64 | 209 | 34736 |
| DISCOVAR 250 base reads | 4207 | 2573 | 59 | 1485 | 57296 |
| total | 4486 | | | | |

**Supplementary Table 6. Raw data for false negative and false positive rates.** This table provides the raw counts from which the rates in **Table 2** are derived (exclusive of corrections from **Supplementary Table 1ab**, which are not included here). Fosmid calls: last row (total) shows that total number of calls that are deduced by aligning the Fosmid reference sequences to the hg19 reference sequence; first four rows show the number of such calls that are present in each of the given call sets for NA12878. Fosmid calls - homozygous: these are the homozygous calls that lie in the Fosmid reference set. Homozygous non-Fosmid calls: these are the homozygous calls that do not lie in the Fosmid reference set. Heterozygous calls: the number of calls are shown for the region 10-110 Mb on chromosome X for a male (NA12878's father) and a female (NA12878).

## Supplementary Table 7. Indel differences between HGAP+iCorn assemblies and Fosmid reference sequences

| Fosmid id | x/y/z | # x/y/z | # x/z | conclusion |
|---|---|---|---|---|
| 1 | TATTAAGACATCAC/T/TACTTGTTCTGCTG | 0 | 614 (ref) | ref right |
| 1 | TCTCATTGAGTTCCC/A/GTGCAAAGC | 753 (ref) | 0 | ref right |
| 1 | GTGCAAAGC/A/AACACCTGTTAAATGT | 0 | 786 (ref) | ref right |
| 1 | GGCAAAATAGTACGTAGGAGA/T/TTTAAT | 0 | 1248 (ref) | ref right |
| 1 | TTTAAT/C/CTGAAATTATGCTCCCAGA | 0 | 1321 (ref) | ref right |
| 1 | GTTATAGTTAACTGTG/T/TTTTTAACAGAGCATGAAA | 0 | 783 (ref) | ref right |
| 1 | TACTATGCAGCCAT/A/AAAAAATGATGAGTTCAT | 0 | 804 (ref) | ref right |
| 1 | GTGGGGTGGGGGGA/G/GGGGAAGAGATAGCATT | 745 (ref) | 0 | ref right |
| 1 | AGAATATTGGTTGCT/G/GGGGGGGTGGGGAAAAT | 597 (ref) | 5 | ref right |
| 3 | GTCTC/A/AAAAAAAAAAAAAAAAAAAAAAAAAAAAGAAGG | 0 | 15 (ref) | ref right |
| 6 | CAGCTAATTATTTTA/T/TTTTTTGTAGAGATAG | 0 | 245 (ref) | ref right |
| 6 | CATGATTACA/T/TTTCTATAGAGCCCAT | 0 | 232 (ref) | ref right |
| 6 | TTTCAGAGATTGGTGG/C/CCCTTAT | 0 | 195 (ref) | ref right |
| 19 | AGTGC/AA/AAAAAAAAAAAAAAAAAAAAAAAAAATATAT | 3 | 45 (ref) | ref right |
| 20 | TTTTTG/TTTTTTT/TTTTTTTTTTTTTTTTTTTTTTT<br>TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTGAGAC | 1 (ref) | 0 | ref right[1] |
| 34 | GCTCACTGCAACCTCCACCTCCCGGGTTCAAGCAATC/<br>CTCCTGCCTCAGCCTCCTGAGTAGCTGGGACTACAGGTGTGTGCTACCACACCAAGCTAATTTTTTTTGTATT<br>TTTAGTAGAGATGAGGTCTCCCATGTTGGCCAGGATGGTCTCAATCTCTTCACCTCAGATCCACCCACCTCAG<br>CCTCCCAAAGTGCTGGGATTACAGGCGTGAGCCACTATCCTTGGCCTGTTGTTTTTTTTTTTTTTTTGAGACG<br>GAGTTTCATTCTTGTTGCCCAGGCTGGAGTGCAGTGGCGCAATCTCAGCTCACTGCAACCTCTGCCTCCTTGG<br>TTCAAGGAATT/<br>CTCCTGCCTCAGCCTCCTGAGTAGCTGGGACTACAGGTGTGTGCTACCACACCAAGCTAATTTTTTTTGTATT<br>TTTAGTAGAGATGAGGTCTCCCATGTTGGCCAGGATGGTCTCAATCTCTTCACCTCAGATCCACCCACCTCAG<br>CCTCCCAAAGTGCTGGGATTACAGGCGTGAGCCACTATCCTTGGCCTGTTGTTTTTTTTTTTTTTTTGAGACG<br>GAGTTTCATTCTTGTTGCCCAGGCTGGAGTGCAGTGGCGCAATCTCAGCTCACTGCAACCTCTGCCTCCTTGG<br>TTCAAGGAATT<br>CTCCTGCCTCAGCCTCCTGAGTAGCTGGCATTATA | 0 | 0 (ref) | ref right[2] |
| 39 | CCATC/A/AAAAAAAAAAAAAAAAAAAAGATTT | 213 | 7 (ref) | ref wrong |
| 40 | AAGACACCTT/CCGAGCGTCTGCTCTATCCCCTTCCACCCTCAGCGGATGATAATCTCAAGACACCTC/CCGA<br>GTGTCT | 106 | 1 (ref) | ref wrong |
| 41 | TCAAGATGGGGAACA/G/GCAGCTCCCAGGGC | 0 | 55 (ref) | ref right |
| 41 | GCAGCTCCCAGGGC/G/TATATTCTGTTTTGTT | 0 | 83 (ref) | ref right |
| 43 | GTCTC/AA/AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAGATTT | 0 | 1 (ref) | ref right[3] |
| 49 | ATTC/TTTT/TTTTTTTTTTTTTTTTTTTTTTTTGAGA | 18 | 1 (ref) | both wrong[4] |
| 52 | ATCTCTGAGT/<br>TTTTCTCTGAGGAGTGTGACTTGGGATGGGGGCAGGGGGGCCTGTCCTCACCAGCCTTGTCATCTCTGAGC/<br>TTTTCTCTGAGGAGTGTGACTTGGGATGGGGGCAGGGGGGCCTGTCCTCACCAGCCTTGTCATCTCTGAGC<br>TTTTCTCTGAGGAGTGTGACTTGGGATGGGGGCAGGGGGGCCTGTCCTCACCAGCCTTGTCATCTCTGAGC<br>TTTTCTCTGAGGAGTGTGACTTGGGATGGGGGCAGGGGGGCCTGTCCTCACCAGCCCTGTCATCTCTGAGT | 0 | 0 (ref) | ref wrong[5] |
| 55 | GAAAGAA/<br>AAAGAAAGAAAGAAAGAAAGAAAG/<br>AAAGAAAGAAAGAAAGAAAGAAAG<br>AAAGAAAGAAAGAGAGGAA | 0 | 0 (ref) | both wrong[6] |
| 61 | ATGGC/T/TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTGAGAC | 0 | 9 (ref) | ref right |
| 76 | AAC/T/TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTACC | 13 (ref) | 2 | ref right |
| 78 | ATGGC/AT/ATATATATATATATATATATATATATATATATATATATATATATATATAAAAG | 0 | 118 (ref) | ref right |
| 79 | GATAC/T/TTTTTTTTTTTTTTTTTTTTTTTTAAGTA | 249 | 13 (ref) | ref wrong |
| 79 | CCCTC/T/TTTTTTTTTTTTTTTTTTTTTTTTTTTTGAGAC | 5 | 110 (ref) | ref right |
| 85 | AGCAAAAAAAAAAAAAAAAAAAAA/AAAACCCAAAAC/AAAACAAAACAAAAAATT | 4 | 0 (ref) | ref wrong |
| 91 | ACC/A/AAAAAAAAAAAAAAAAAAAAAGGC | 1 (ref) | 101 | ref wrong |
| 94 | ATCTC/A/AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAGAT | 1 | 50 (ref) | ref right |
| 94 | GATTT/A/AAAAAAAAAAAAAACAATC | 1 (ref) | 787 | ref right[7] |
| 96 | CAGAGAAAGAAGGAG/AGAA/AGAAAGAAAGAAAGAAAGAAAGAAAGAAAGAAAGAAAGAAAGAAAGAAA<br>GAAAGAAAGAAAGAAAGAGA | 2 | 133 (ref) | ref right |
| 104 | ATATA/TTTT/TTTTTTTTTTTTTTTCTATT | 2 | 373 (ref) | ref right |

**Supplementary Table 7. Indel differences between HGAP+iCorn assemblies and Fosmid reference sequences.** For each indel between an HGAP+iCorn assembly and the Fosmid reference sequence, we manually assayed the locus to determine the true sequence. Fosmid id: the identifier of the Fosmid. x/y/z: three sequences whose concatenation was present in either the HGAP+iCorn assembly or the Fosmid reference sequence. The sequence y is deleted in one. # x/y/z: number of times that x/y/z appears in the Illumina Fosmid pool reads, plus the annotation (ref) if x/y/z represents the Fosmid reference version. # x/z: number of times that x/z appears in the Illumina Fosmid pool reads, plus an annotation (ref) if x/z represents the Fosmid reference version. Conclusion: asserts that either the reference is right (and if so the HGAP+iCorn assembly is wrong), or that the reference is wrong (and if so that the HGAP+iCorn assembly is right), or that both are wrong.

[1]Long form supported by NA12878 WGS reads, 2 to 0.

[2]Long form shows perfect tandem duplication of 303 base sequence, which cannot be correct, because ATCTCAGCTCACTGCAACCTCTGCCTCCTTGGTTCAAGGAATTCTCCTGCCTCAGCCTCCTGAGTAGCTGG appears 251 times in the Fosmid pool reads whereas ATCTCAGCTCACTGCAACCTCTGCCTCCTTGGTTCAAGGAATTCTCCTGCCTCAGCCTCCTGAGTAGCTGGG (one base added on right) appears zero times.

[3]Short form supported by NA12878 WGS reads, 1 to 0.

[4]Most likely true answer is y = TT (count = 45 in Fosmid pool).

[5]Insertion of copy in tandem duplication. It seems unlikely that the HGAP assembly would be wrong.

[6]Sequence GAAAGAAAAAGAAAGAAAGAAAGAAAGAAAGAAAGAAAGAAAGAAAGAAAGAAAGAAAGAAAGA is present 111 times in Fosmid pool reads, and not in the reference. However next base in reads is clearly G but next base in HGAP+iCorn assembly is A. Since the truth was uncertain, we did not correct the reference sequence in this case.

[7]Fosmid pool reads support HGAP+iCorn assembly, but NA12878 WGS reads support reference, 53 to 0, suggesting defect in Fosmid clone.

## Supplementary Table 8. Other differences between HGAP+iCorn assemblies and Fosmid reference sequences

| Fosmid id | assembly (HGAP+iCorn) | reference | # assembly | # reference | conclusion |
|---|---|---|---|---|---|
| 3 | GAGGCAGGAGAATGGCATGAACCCG | GAGGCAGGAGAATGGCGTGAACCCG | 264 | 29 | ref wrong[1] |
| 26 | TCCCCTCCCCTCTCCTCCGTTCCTCTCCCC TCTCCTCTGTTCCTCTGCCCTCTCCTCCGT TCCTCTCCCCTCTCCTCTCC | TCCCCTCCCCTCTCCTCCGTTCCTCTCCCC TCTCCTCCGTTCCTCTGCCCTCTCCTCCGT TCCTCTCCCCTCTCCTCTCC | 94 | 0 | ref wrong |
| 79 | TTTGGGAGGCTGAGGCGGGGGGCGGGGGG GGGGCGGAAGAGAGGGGAGGAGGTGGGG GAGGAGGGGGAGGGGGGAGGGGGGGGGG GGGATGGGGGGGGTGCATCAT | TTTGGGAGGCTGAGGAGGGGGGATGGGGG GGGTGCATCAT | 0 | 0 | ref wrong[2] |
| 106 | AGGGCTGTGTTATAGGGGGGTGGGGGGGG GGGGGGCCTG | AGGGCTGTGTTATAGGGGGGTGGGGGGGT GGGGGGCCTG | 8 | 37 | ref right[3] |
| 106 | AAGCCGGGGCTCCCCCCCCGCCCCTCCCTC CTCCCT | AAGCCGGGGCTCCCCACCCGCCCCTCCCTC CTCCCT | 24 | 43 | ref right[3] |

**Supplementary Table 8. Indel differences between HGAP+iCorn assemblies and Fosmid reference sequences.** For each non-indel difference between an HGAP+iCorn assembly and the Fosmid reference sequence, we manually assayed the locus to determine the true sequence. Fosmid id: the identifier of the Fosmid. Assembly (HGAP+iCorn): the sequence found in the HGAP+iCorn assembly. Reference: the sequence found in the Fosmid reference. Bases that differ between the assembly and reference are labeled in red. # assembly: number of times that the assembly sequence occurs in the Illumina Fosmid pool reads. # reference: number of times that the Fosmid reference sequence appears in the Illumina Fosmid pool reads. Conclusion: same as in **Supplementary Table 7**.

[1]The Fosmid allele was inherited from NA12878's mother. Reads from mother vote 7:0 for assembly.
[2]Accepting HGAP+iCorn sequence as most likely the best approximation to the true sequence.
[3]Assembly sequence has common systematic error for Illumina, and the sequence context is also difficult for Pacific Biosciences.

## Supplementary Table 9. Locations of Fosmids in hg19 and GenBank

| id | chr | start | stop | GenBank |
|---|---|---|---|---|
| 0* | 1 | 405014 | 436839 | KC951366.1 |
| 1 | 1 | 24846099 | 24885028 | KC951367.1 |
| 2 | 1 | 54770796 | 54815430 | KC951368.1 |
| 3 | 1 | 164908215 | 164944914 | KC951369.2 |
| 4 | 2 | 106774655 | 106814780 | KC951370.1 |
| 5 | 2 | 239359354 | 239395827 | KC951371.1 |
| 6 | 3 | 11057753 | 11093946 | KC951372.1 |
| 7 | 3 | 61537377 | 61574622 | KC951373.1 |
| 8 | 5 | 111033226 | 111071672 | KC951374.1 |
| 9 | 5 | 177690366 | 177732716 | KC951375.1 |
| 10 | 5 | 179300456 | 179343713 | KC951376.1 |
| 11 | 5 | 179230770 | 179265317 | KC951377.2 |
| 12 | 6 | 19607928 | 19643253 | KC951378.1 |
| 13 | 6 | 92591410 | 92635929 | KC951379.2 |
| 14 | 7 | 3890242 | 3929023 | KC951380.1 |
| 15 | 7 | 38718049 | 38755677 | KC951381.1 |
| 17 | 8 | 23203282 | 23242439 | KC951382.1 |
| 18 | 8 | 30787215 | 30822871 | KC951383.1 |
| 19 | 8 | 72792736 | 72837890 | KC951384.1 |
| 20 | 8 | 128773536 | 128809830 | KC951385.1 |
| 21 | 10 | 30893379 | 30932772 | KC951386.1 |
| 22 | 11 | 44933001 | 44966967 | KC951387.1 |
| 23 | 11 | 45516649 | 45559188 | KC951388.1 |
| 24 | 11 | 64973568 | 65010825 | KC951389.1 |
| 25 | 11 | 67764493 | 67794848 | KC951390.1 |
| 26 | 11 | 75490858 | 75534967 | KC951391.2 |
| 27 | 11 | 111825164 | 111865892 | KC951392.1 |
| 28 | 12 | 3154380 | 3184001 | KC951393.1 |
| 29 | 12 | 7028568 | 7066173 | KC951394.1 |
| 30 | 12 | 14859162 | 14896823 | KC951395.1 |
| 31 | 12 | 57585789 | 57666917 | KC951396.1 |
| 32 | 12 | 113988564 | 114030742 | KC951397.1 |
| 34 | 14 | 104039549 | 104081406 | KC951398.1 |
| 35* | 15 | 21334868 | 21375963 | KC951399.2 |
| 37 | 15 | 30434722 | 30473940 | KC951400.2 |
| 39 | 15 | 74963396 | 74996690 | KC951401.2 |
| 40 | 16 | 15032318 | 15067606 | KC951402.2 |
| 41 | 16 | 24292447 | 24331252 | KC951403.1 |
| 42 | 16 | 52733566 | 52765907 | KC951404.1 |
| 43 | 16 | 61336985 | 61375446 | KC951405.1 |
| 44 | 17 | 20363625 | 20405498 | KC951406.1 |
| 45 | 17 | 48615799 | 48632029 | KC951407.1 |
| 46 | 17 | 72429061 | 72449823 | KC951408.1 |
| 47 | 18 | 43380830 | 43426587 | KC951409.1 |
| 48 | 18 | 7359307 | 7390782 | KC951410.1 |
| 49 | 19 | 18994603 | 19031125 | KC951411.2 |
| 50 | 19 | 50071090 | 50109758 | KC951412.1 |
| 51 | 19 | 50180784 | 50217291 | KC951413.1 |
| 52 | 22 | 50807079 | 50839730 | KC951414.2 |
| 53 | X | 55290022 | 55330437 | KC951415.1 |
| 54 | X | 103565209 | 103602970 | KC951416.1 |
| 55 | X | 137402949 | 137439452 | KC951417.1 |
| 56 | 1 | 34656948 | 34694712 | KC951418.1 |
| 57 | 1 | 37070197 | 37107529 | KC951419.1 |
| 58 | 1 | 78024986 | 78062378 | KC951420.1 |
| 59 | 1 | 78462072 | 78498676 | KC951421.1 |
| 60 | 1 | 94073186 | 94106033 | KC951422.1 |
| 61 | 1 | 243215939 | 243250704 | KC951423.1 |
| 62 | 2 | 10103954 | 10143530 | KC951424.2 |
| 63 | 2 | 50311571 | 50354951 | KC951425.1 |
| 64 | 2 | 74222369 | 74267555 | KC951426.1 |
| 65 | 2 | 75064950 | 75104088 | KC951427.1 |
| 66 | 2 | 106493614 | 106522780 | KC951428.1 |
| 67* | 2 | 131216319 | 131248821 | KC951429.2 |
| 68 | 6 | 11838123 | 11869485 | KC951430.1 |
| 69 | 3 | 12841074 | 12881234 | KC951431.1 |
| 70 | 3 | 13612192 | 13654898 | KC951432.1 |
| 71 | 3 | 15353207 | 15385667 | KC951433.1 |
| 72 | 3 | 40870754 | 40911957 | KC951434.1 |
| 73 | 3 | 56210318 | 56255345 | KC951435.1 |
| 74 | 3 | 59791190 | 59827634 | KC951436.1 |
| 75 | 4 | 6070823 | 6107333 | KC951437.1 |
| 76 | 4 | 57791135 | 57826227 | KC951438.1 |
| 77 | 5 | 34824703 | 34855337 | KC951439.1 |
| 78 | 5 | 132691392 | 132724301 | KC951440.1 |
| 79 | 5 | 172315675 | 172353050 | KC951441.2 |
| 80 | 7 | 39598633 | 39637867 | KC951442.1 |
| 81 | 7 | 73940579 | 73986540 | KC951443.1 |
| 82 | 7 | 148955531 | 149001605 | KC951444.1 |
| 83 | 9 | 83592535 | 83629710 | KC951445.1 |
| 84 | 9 | 124612574 | 124655194 | KC951446.1 |
| 85 | 9 | 130946544 | 130984464 | KC951447.2 |
| 86 | 10 | 117634356 | 117679433 | KC951448.1 |
| 87 | 10 | 121430681 | 121475222 | KC951449.1 |
| 88 | 11 | 10602656 | 10645411 | KC951450.1 |
| 89 | 11 | 33312690 | 33350126 | KC951451.1 |
| 90 | 11 | 47933833 | 47973312 | KC951452.1 |
| 91 | 11 | 62264853 | 62298284 | KC951453.2 |
| 92 | 12 | 97595816 | 97633973 | KC951454.1 |
| 93 | 15 | 101709980 | 101744524 | KC951455.1 |
| 94 | 16 | 29831098 | 29872005 | KC951456.1 |
| 95 | 16 | 67944129 | 67979450 | KC951457.1 |
| 96 | 17 | 11024357 | 11062513 | KC951458.1 |
| 97 | 18 | 4717841 | 4760325 | KC951459.1 |
| 98 | 18 | 54920710 | 54946737 | KC951460.1 |
| 99 | 19 | 2110089 | 2149370 | KC951461.1 |
| 100 | 19 | 19230665 | 19258022 | KC951462.1 |
| 101 | 19 | 36693157 | 36732793 | KC951463.1 |
| 102 | 20 | 32071423 | 32114596 | KC951464.1 |
| 103 | 20 | 32619074 | 32662708 | KC951465.1 |
| 104 | 20 | 52417824 | 52454221 | KC951466.1 |
| 105 | 22 | 45807736 | 45847543 | KC951467.1 |
| 106 | 22 | 50330871 | 50364777 | KC951468.2 |

**Supplementary Table 9. Locations of Fosmids in hg19 and GenBank.** For each Fosmid clone, exact coordinates for the best location on the hg19 reference sequence are shown. Three clones (labeled *) had alternate locations, see **Supplementary Note, Section 7e**. GenBank identifiers are also shown.

## Supplementary Table 10. Coverage of Fosmids

| id | coverage (x) | id | coverage (x) | id | coverage (x) | id | coverage (x) |
|---|---|---|---|---|---|---|---|
| 1 | 56.6 | 27 | 56.6 | 56 | 51.7 | 82 | 50.0 |
| 2 | 50.9 | 28 | 50.9 | 57 | 51.2 | 83 | 57.9 |
| 3 | 57.9 | 29 | 46.0 | 58 | 62.3 | 84 | 51.5 |
| 4 | 52.0 | 30 | 57.8 | 59 | 63.4 | 85 | 47.3 |
| 5 | 55.3 | 31 | 45.8 | 60 | 54.2 | 86 | 58.4 |
| 6 | 51.2 | 32 | 51.7 | 61 | 54.4 | 87 | 52.2 |
| 7 | 57.4 | 34 | 53.5 | 62 | 56.5 | 88 | 56.3 |
| 8 | 58.2 | 37 | 44.3 | 63 | 61.0 | 89 | 59.1 |
| 9 | 48.8 | 39 | 57.5 | 64 | 52.6 | 90 | 50.9 |
| 10 | 49.6 | 40 | 81.2 | 65 | 52.4 | 91 | 54.6 |
| 11 | 46.0 | 41 | 54.4 | 66 | 52.2 | 92 | 59.9 |
| 12 | 58.9 | 42 | 53.3 | 68 | 57.3 | 93 | 53.7 |
| 13 | 60.1 | 43 | 57.4 | 69 | 52.7 | 94 | 48.9 |
| 14 | 55.4 | 44 | 46.8 | 70 | 48.4 | 95 | 46.6 |
| 15 | 56.6 | 45 | 42.8 | 71 | 54.7 | 96 | 57.1 |
| 17 | 52.4 | 46 | 48.7 | 72 | 55.4 | 97 | 59.0 |
| 18 | 52.9 | 47 | 53.5 | 73 | 58.4 | 98 | 52.9 |
| 19 | 60.6 | 48 | 52.5 | 74 | 57.7 | 99 | 44.8 |
| 20 | 54.4 | 49 | 44.1 | 75 | 52.7 | 100 | 48.1 |
| 21 | 57.9 | 50 | 48.6 | 76 | 54.2 | 101 | 53.4 |
| 22 | 44.5 | 51 | 46.0 | 77 | 54.5 | 102 | 53.9 |
| 23 | 53.4 | 52 | 55.1 | 78 | 55.3 | 103 | 52.1 |
| 24 | 48.2 | 53 | 56.7 | 79 | 47.8 | 104 | 56.6 |
| 25 | 44.1 | 54 | 55.8 | 80 | 59.0 | 105 | 50.4 |
| 26 | 52.3 | 55 | 58.8 | 81 | 49.0 | 106 | 42.1 |

**Supplementary Table 10. Coverage of Fosmids.** For each of the 100 Fosmid reference sequences that could be uniquely aligned to the hg19 reference sequence (**Supplementary Table 9**), the aligned coverage of this work's data set to the corresponding region on hg19 is shown. Coverage includes the full length of the reads and their partners.

## Supplementary Note

### 1. Sequence availability

For this work we generated data for the human cell line NA12878. FASTQ files are available at http://trace.ncbi.nlm.nih.gov/Traces/sra/sra.cgi?exp=SRX297987&cmd=search&m=search&s=seq. These data are 250 base reads from a PCR-free library.

Additional sequence data from NA12878 and her parents, of the same type described in this work, and generated as part of the 1000 Genomes Project, are available as BAM files from the SRA:

| sample | flowcell | lane | run accession |
|---|---|---|---|
| NA12878 | H06HDADXX | 1 | SRR826469 |
|  |  | 2 | SRR826463 |
|  | H06JUADXX | 1 | SRR826467 |
| NA12892 (mother) | H06JHADXX | 1 | SRR826428 |
|  |  | 2 | SRR826473 |
|  | H06JUADXX | 2 | SRR826471 |
| NA12891 (father) | H03N7ADXX | 1 | SRR826427 |
|  |  | 2 | SRR826448 |
|  | H05F1ADXX | 2 | SRR826465 |

These BAM files were produced as part of this work by aligning using BWA as in **Supplementary Note, Section 7a**, and are different from the BAM files available at `ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/data/`*x*`/high_coverage_alignment`, where *x* is NA12878, NA12891 or NA12892. The latter BAM files were created using BWA-MEM.

Illumina has deeply sequenced cell lines from the seventeen-member CEPH pedigree consisting of the mother (NA12878), the father, their parents (the 'grandparents') and their eleven children (http://www.ebi.ac.uk/ena/data/view/ERP001960). These data consist of pairs of 100-base reads from a PCR-amplified library.

Fosmid Pool data are available from the NCBI Short Read Archive, for Illumina (flowcell A2925; run accessions SRR835433 and SRR835426), and for Pacific Biosciences (run accessions SRR835425, SRR835427-29, SRR835431-32, SRR835436-37 and SRR849054-59; these accessions correspond to 7 physical runs).

### 2. Laboratory methods for generation of 250 base read data

PCR-free libraries for Illumina sequencing have been generated using gel size selection[14], however in this work we used a gel-free method to reduce costs and input requirements (based on http://www.illumina.com/truseq.ilmn). Briefly, Illumina PCR-free fragment shotgun libraries were prepared using the 'with-bead pond library' construction protocol[34] with the following modifications. 500 ng of genomic DNA, in a volume of 50 μl, was sheared to a size of ~400 bp using a Covaris E210 instrument (Covaris) using Illumina's TruSeq PCR-free protocol shearing parameters (Illumina, Part # 15036187 A): Duty cycle = 10%, intensity = 5, cycles per burst = 200, time = 45 seconds. Fragmented DNA was then cleaned up with 0.6x Agencourt AmPure XP SPRI beads and eluted in 40 μl Tris-HCl pH8.0, following manufacturer's recommendations (Beckman Coulter). DNA fragments were then further cleaned up with 3.0x Agencourt AmPure XP SPRI beads, following manufacturer's recommendations (Beckman Coulter), but DNA was not eluted from the SPRI beads. Then using the KAPA Library

Preparation Kit reagents (KAPA Biosystems, Catalog # KK8241), DNA fragments bound to the SPRI beads were subjected to end repair, A-base tailing and Illumina 'PCR-free' TruSeq adapter ligation (Illumina, Catalog FC-121-3001) following manufacture's recommendations (KAPA Biosystems). A second 0.7x SPRI clean up was performed following adapter ligation to remove adapter dimers and library fragments below ~150 bp in size. No library PCR amplification enrichment was performed. The sequence ready Illumina PCR-free library was then eluted off the SPRI beads following manufacturer's recommendations (Beckman Coulter). Libraries were quantified with quantitative PCR using KAPA Library Quant kit (KAPA Biosystems, Catalog # KK4824) and an Agilent Bioanalyzer High Sensitivity Chip (Agilent Technologies) following the manufacturer's recommendations.

Libraries were sequenced with 250 base paired-end reads on an Illumina HiSeq 2500 instrument in Rapid Run Mode, with the following modifications. Reagents from two 200 cycle TruSeq Rapid SBS Kit v1 (Illumina, catalog # FC-402-4001) were combined and run using a 500 cycle run. To enable a 500 cycle run the <SBSMAXCycleRR> value in the HiSeqControlSoftware.Options.cfg file was changed to 500 cycles *i.e.* <SBSMAXCycleRR>500</SBSMaxCycleRR>. According to Illumina it is also possible to define the number of cycles in the HiSeq Control Software under the Run Configuration tab, however entering non-supported read length will result in a warning message. Currently Illumina does not support read lengths greater than 150 bases on the HiSeq 2500 with the v1 chemistry, however they plan to do so with the next release.

**Supplementary Figure 3** shows that the actual distribution of read pairs corresponds to a modal fragment size of ~450 bp. We note that a lower loading density (yielding less sequence coverage per flowcell) could be advantageous either because it might reduce bias at high GC, or because it would reduce the likelihood of catastrophic overloading, although it is difficult to know if this would sufficiently compensate for the lower coverage.

Note that sequencing can also be carried on the Illumina MiSeq instrument.

## 3. Generation of Fosmid reference sequences

Two pools containing ~50 Fosmids each were generated from Human NA12878 genomic DNA using the gel free library-construction method[35]. Briefly, the DNA was end repaired and ligated to double stranded barcoded oligonucleotides with SapI overhangs (5'Phosphate-GATCTAGTTGCTT, 5'Phosphate-AAGCAACTAG). Subsequent ligation to AatII/SapI digested pFosill-4 vector arms, packaging in phage lambda extract and transformation of GC10T1 cells were as described except cells were grown on LB agar plates supplemented with 15 μg/ml chloramphenicol not in liquid culture. Plates were incubated overnight at 30$^{o}$C and two sets of ~50 colonies were picked and grown individually in 2 ml LB with 15 μg/ml chloramphenicol overnight. Each set of ~50 cultures was pooled and grown at 30$^{o}$C in 2XYT, plus 15 μg/ml chloramphenicol with shaking at 225 rpm, to an $OD_{600}$ of 1-2. Fosmids from each pool were purified using a High Speed Plasmid kit (Qiagen, 12662) according to the manufacturer's instructions. Contaminating *E. coli* genomic DNA was removed using 1.5 U Plasmid Safe ATP Dependent DNase (Epicentre, E3101K) per 1 μg Fosmid DNA by incubating at 37$^{o}$C for 30 min and heat inactivating at 70$^{o}$C for 30 min. Agencourt AMPure XP beads (Beckman Coulter Inc, A63880) were used to purify the Fosmid pools according to the manufacturer's instructions.

The pools were Illumina shotgun sequenced as described in **Supplementary Note, Section 2** but with the following changes in DNA shearing time parameters: time = 90 seconds instead of 45 seconds. The pools were shotgun sequenced by Pacific Biosciences, using the following conditions. Pacific Biosciences sequencing libraries were generated using the DNA Template Prep Kit V2.0 (Pacific Biosciences) following the manufacturer's recommendations but the following modifications. 2 μg of FosIll pool DNA was sheared to ~10 kb in size using a Covaris gtube (Covaris) following the manufacturer's recommendations but using the following parameters: centrifuge gtube at 6000 rpm for 1 min, flip gtube and centrifuge gtube at 6000 rpm for 1 min. DNA fragments were purified, end-repaired, and ligated with Pacific Biosciences SMRTbell sequencing adapters following manufacturer's recommendations (Pacific Biosciences) but with the following modification. Similar to the 'with-bead' method[34], SPRI beads added after shearing remained in the reaction until the final library elution step. Therefore after the first addition of SPRI beads all other SPRI reactions were performed by only adding Ampure bead PEG buffer without beads. Following adapter ligation, sequencing libraries immobilized on SPRI beads were cleaned up with three 0.45x SPRI reactions following the manufacturer's recommendations (Beckman Coulter Genomics) and eluted off the beads. Sequencing libraries were combined with sequencing primer and polymerase (version 2.0) following manufacturer's recommendations (Pacific Biosciences). The resulting complex was subjected to Pacific Biosciences sequencing (version 2.0 chemistry) followed by primary data analysis (version 1.3.3 analysis software) on a PacBio-RS instrument following manufacturer's recommendations (Pacific Biosciences).

The two Fosmid pools were assembled using a preliminary version of DISCOVAR. We first aligned the reads to the hg19 reference sequence and identified regions having high coverage. We assembled the Illumina reads from each of these regions individually, after normalizing coverage to ~150x where possible, then identified and removed the canonical sequences that would be expected at the Fosmid ends. In a number of cases these canonical sequences were not found, and in some cases chimeric junctions where identified instead. In such cases we trimmed the assemblies by removing sequence after the last point where the assembly matched the reference well. If the Illumina data did not yield a fully resolved assembly, we added in the Pacific Biosciences data, and assembled, adapting some methods from (ref. 36) for this application. We also assembled each Fosmid pool in its entirety and in some cases identified clones that could not be assembled completely from reads aligned to a single region. In a number of cases we examined the assembly graph and chose what appeared to be the correct path, a process akin to manual finishing.

In all cases where there was > 100x Illumina coverage, the clones were completely and unambiguously assembled, except for one heterochromatic clone, which was excluded.

We observed that one clone joined two distant regions on chr17, perhaps arising from a single mutated cell, and indeed the junction sequence between the regions was absent from the whole-genome data. In this case we broke the clone reference sequence at the junction point.

We were concerned that evaluating DISCOVAR variant calls using DISCOVAR-created Fosmid reference sequences could bias our analysis. To assess the potential effect of this, we applied another, orthogonal approach to assembling the Fosmids, and then assessed the impact.

Briefly, we assembled all the Fosmids using HGAP, applied to only the Pacific Biosciences reads. Of the 103 Fosmids, 57 yielded complete assemblies having no gaps. These were: 0,1,3,5,6,7,8,10,12,13,14,17, 18,19,20,21,25,26,29,30,34,39,40,41,43,47,49,52,53,55,56,61,65,69,71,72,73,75,76,77,78,79,80,81,84,85, 86,88,91,94,96,97,98,99,100,104,106.

These complete assemblies were further processed using iCorn, which used the Illumina reads to improve the HGAP assemblies. Of these 57 HGAP+iCorn assemblies, 32 agreed perfectly with the Fosmid reference sequence. We manually analyzed every difference between the remaining 25 assemblies and the Fosmid reference sequences (**Supplementary Tables 7, 8**). We found 29 errors in the HGAP+iCorn assemblies, 9 in the Fosmid reference sequences, and 2 cases where both were wrong. We note that 50 of

57 HGAP-only assemblies had errors, and that for those Fosmids having at least 200x PacBio coverage, 15 of 16 had errors. This suggests that it could be difficult to obtain essentially perfect reference sequences using the PacBio data alone.

We then corrected the Fosmid reference sequences, and computed the change in variant calling statistics resulting from these changes. They were as follows (at the time this change was made):

| **call set** | **ΔFN (%)** | **ΔFP-hom (%)** |
|---|---|---|
| Platinum-100 | 0.1 | 0 |
| GATK-250 | 0.1 | 0 |
| Cortex | 0.1 | 0 |
| DISCOVAR | 0.2 | -0.04 |

For example, for DISCOVAR, false negatives increased from 5.9% to 6.1%, and homozygous false positives decreased from 2.36% to 2.32%.

These changes were sufficiently small and neutral as to suggest that the overall bias introduced by using DISCOVAR in two roles is small.

Below, we describe how we used HGAP and iCorn.

To run HGAP, we first made the Pacific Biosciences SmrtAnalysis toolkit 2.0.1 available. Then we set up the environment using the SmrtAnalysis configuration script:
```
    source $SEYMOUR_HOME/etc/setup.sh
```
Next we created a file pacbio_h5_filelist containing the names of the Pacific Biosciences data files. Then we created a file input.xml file using fofnToSmrtpipeInput.py (from SmrtAnalysis toolkit):
```
    fofnToSmrtpipeInput.py pacbio_h5_filelist > input.xml
```
Next we obtained a file settings.xml from Pacific Biosciences. We have made this file available at ftp://ftp.broadinstitute.org/pub/crd/DiscovarManuscript/HGAP. We then modified this file, changing the "Approximate genome size in base pairs" tag to: 4,000,000 – the approximate total size of 100 Fosmids.

Then we ran HGAP:
```
    smrtpipe.py --params=settings.xml xml:input.xml
```
The final Pacific Biosciences assembly can be found here:
```
    data/polished_assembly.fasta.gz
```
Next we used iCorn, which is part of Post Assembly Genome Improvement Toolkit (PAGIT): (http://www.sanger.ac.uk/resources/software/pagit/) version 1 (ftp://ftp.sanger.ac.uk/pub4/resources/software/pagit/PAGIT.V1.64bit.tgz).

To run iCorn, for each Fosmid for which there existed a full-length contig, we selected the full-length contig. In some cases there were other contigs matching part of the Fosmid reference sequence, and these were ignored. We then selected the Fosmid pool reads that aligned to the corresponding region of the hg19 reference sequence, and downsampled to 150x if coverage exceeded that level. These reads were placed in a FASTQ file using phred offset 33. Then we used Post Assembly Genome Improvement Toolkit's iCorn:
```
    source ${PAGIT_DIR}/sourceme.pagit
    export CARMA_CORRECT_QUAL=20
    corn.start.sh fosmid_n.fasta 1 5 f.A.fastq f.B.fastq 50,1000 260
```
where fosmid_n.fasta is the HGAP contig, f.A.fastq is the FASTQ file providing the first read of a given read pair, and f.B.fastq provides the second read.

## 4. Validation of Fosmid reference sequences

We validated the Fosmid reference sequences by Sanger sequencing loci on the Fosmids that were inconsistent with an initial set of Platinum-100 variant calls. (In creating this we also included 14 cases where there was an inconsistency with the DISCOVAR variant call set.) We started with a list of 134 inconsistent windows (all those lying within a fixed set of the Fosmids), as described in the main text. From these we selected 95 loci for sequencing on a single plate. To do so we chose primers for all using Primer3 (http://primer3.sourceforge.net/releases.php) version 2.3.5. Primers were checked for uniqueness by alignment to the pooled Fosmid references. Primers that aligned multiple times were discarded. Starting from 'ideal' settings for melting temperature *etc.*, conditions were relaxed (as indicated by min/max below) until 95 primer pairs fell within bounds.

| Parameter | Min | Max | Mean |
|---|---|---|---|
| **GC%** | 37 | 70 | 52 |
| **TM** | 62.0 | 64.9 | 63.3 |
| **delta-TM** | 0.01 | 1.87 | 0.66 |
| **primer length** | 19 | 30 | 24 |
| **insert size** | 423 | 681 | 556 |

Sanger sequencing was performed at the Massachusetts General Hospital DNA Core Facility (https://dnacore.mgh.harvard.edu). Of the 95 events targeted, 93 events were covered by at least one read.

We aligned the Sanger reads to the Fosmid reference sequences. We first identified those windows where the entire window was covered by Q20 bases on at least one Sanger read. There were 49 of these. In 48 cases the window was confirmed, whereas in 1 case we identified a Fosmid reference sequence error within the window (see main text).

We then examined the remaining 44 cases in more detail. We classified 31 of these as being confirmed by Illumina reads if all of the following criteria were satisfied: (1) at least two reads from the Fosmid pool confirm the window; (2) at least two NA12878 WGS reads confirm the window; (3) amongst the Fosmid pool reads, the ratio of those confirming the window to those confirming the alternate allele (defined by hg19) is at least 5:1.

Of the remaining 13 cases, based on manual inspection, we identified 8 cases as being clearly confirmed by the Sanger reads (although not meeting the preceding Q20 criterion). Finally there were 5 remaining cases for which neither the Sanger nor Illumina data were diagnostic.

The number 87 of confirmed windows in the main text is 48 + 31 + 8.

## 5. DISCOVAR assembly method

Here we describe the DISCOVAR algorithm in detail, as it would be applied either to a small genome in its entirety, or a region of a larger genome (after selecting such reads by alignment to a reference). In the latter case the algorithm is applied only to the reads from the region. References below to 'all' reads refer to the reads from the region.

**Preprocessing of data**. Raw bases and quality scores were generated using the Illumina pipeline, including the EAMSS filtering algorithm that assigns the quality score 2 to bases in the tail of a read that appear to be 'untrusted'. If a data set has been generated without the EAMSS filter, we recommend that it be applied to the data before running DISCOVAR. We also recommend that quality scores not be recalibrated: DISCOVAR is designed to work with the native Illumina quality score distribution.

Reads were aligned to the hg19 human reference sequence using bwa[37], as part of the Picard pipeline http://picard.sourceforge.net.

The DISCOVAR assembly algorithm has two major stages:

- In the first stage, reads are error corrected, and as part of this process, pairs are closed by filling in intermediate bases. The output of the first stage consists of these pair closures.
- The pair closures are now formally merged into a graph, using a minimum overlap K that depends on the distribution of pair closure sizes.
- In the second stage, a series of operations are carried out to simplify and improve the assembly graph.

**Error correction.** Error correction proceeds via a series of steps, as described below.

**Precorrection 1**. This is exactly as described in (ref. 16, Supplementary material), as the module PreCorrect. For convenience we provide the description here. The module examines all 25-mers in all reads. It performs a large, external sort in such a way as to bring together all the 25-mers that have the same initial and final 12-mers, ignoring the central base. For each pile of 25-mers that agree on the flanking bases, and potentially disagree only on the central base, it examines the quality scores associated with the base calls at the central base. Piles of size smaller than 6 are ignored. For larger piles, the quality scores are summed separately for each of the four potential calls of the central base. We declare a winner among the four calls as the call having the greatest sum of quality scores, but only if the sum is 60 or greater. If we have a winner, then we also check for losers. A loser call has no more than one call of quality 20 or more, and its sum of quality scores must be less than 1/4 that of the winner. Reads containing loser-calls are candidates for having those calls corrected to the winning call. One further round of sorting and screening applies the correction to a read only if it is well isolated from other proposed corrections: no changes are made to either call when two proposed corrections are within 12 bases of each other (*i.e.*, when the flanking bases that built a pile are themselves suspect). Whenever a correction is adopted, the associated quality score for that call is set to 0.

**Pair filling 1**. In this step, certain pairs are closed (or 'filled'). The primary purpose of this step is to fill pairs that we can easily and unambiguously identify as having a unique closure, thereby decreasing the computational requirements of the subsequent pair filling steps. Let K = 60. Consider all the kmers in the reads, however exclude kmers that occur less than 5 times in total in the reads and their reverse complements, and for a given read, exclude all kmers that occur after such an excluded kmer. Form the unipath graph associated to these truncated reads. Suppose that both the left and right reads in a pair map completely to the same unipath. Then that unipath defines a closure for the pair, which we accept. Closed pairs are marked as done, and are not subject to further processing *per se*, however they do participate in the subsequent correction of other pairs. For this purpose, the pair is replaced by two reads, that are obtained from the closure by taking the leftmost $n_1$ bases (for the left read), and the reverse complement of the rightmost $n_2$ bases (for the second read). Here $n_1$ and $n_2$ are the original read lengths for the pair. The quality scores are all set to 40.

**Quality score lowering**. We set each quality score to the minimum over the diameter 9 window of the quality scores centered at the given base. This step is designed to lower artificially high quality scores.

**Precorrection 2**. This step is carried out twice, first using K = 24 and then K = 40. It corrects certain bases, and changes the quality scores for those bases to zero. In outline, the process works by finding the true friends of the read, which consist (in principle) of all reads that come from the exact same locus on the exact same chromosome. Then we find the consensus of these reads. Note that a given friend may occur more than once in a stack, with different offsets relative to the founder.

First, for each read, we find its initial friends. Such a friend read comes with a gap-free alignment to the given read. (This makes sense because the indel error rate in the data is very low.) Such an alignment is defined by an orientation and offset. Note that a read may appear multiple times as a friend, using different orientations and offsets. Friend reads are defined by perfect kmer matches, however kmers occurring more than 1000 times in the reads and their reverse complements are ignored. The initial friends are now formed into a stack under the given read. If the stack height is $\geq 10^4$, precorrection is not carried out. In this stack we truncate friends on the left and right so that they do not extend beyond the given read.

Next we 'clean' the stack. Initial friends having a Q30 difference with the founder (*i.e.* Q30 on both reads) are now removed from the stack. Then we remove any friends having a 'high quality difference window' with the founder, as follows. For each 10-base window on the founder, suppose that some friend agrees with the founder for the entirety of the window, and has quality score $\geq 10$ at every base on it. Now suppose that another friend is defined on the window, has at least three differences on the window with the founder, and the sum of its quality scores at those bases is $\geq 30$. Then the friend is declared to have a 'high quality difference window' with the founder and marked for deletion.

Finally, we form the consensus for the stack. To do this we proceed through each of its columns. For purposes of this calculation, quality scores of 1 or 2 are changed to 0.2. For each of the four calls (A,C,G,T) appearing in the column, we compute the quality score sum for that call, and also note the top quality score that appears (for a given call). The call having the highest quality score sum is declared the winner. The top quality score is then deducted from the sum for each non-winner. We then test to see if the 'victory' should be accepted, according to the following requirements: the winner's quality score sum must be at least 50, the winning sum must be at least 10 times the runner up sum, and the runner up sum must be at most 100. If the victory is accepted, and the winning call disagrees with the base on the founder, it is 'corrected' and its quality score is set to zero.

**Pair filling 2**. This step is the same as Pair Filling 1, except that K = 80, and reads are truncated using a different method, namely each read is truncated at the first point where in consensus formation (above), the test fails.

**Pair correction and filling**. As in Precorrection 2, we form a set of initial friends for each read. For this we use K = 40. Both reads in a given pair are processed at the same time. We build stacks, this time allowing extension to the right for each read (thus moving towards the interior of the fragment). If the stack height for either read is $\geq 10^4$, pair correction and filling are not carried out.

We next exclude low-quality pairs. First we compute the mean quality of all bases in the pair. Then we find all the reads that appear in either the left or the right stack, and compute the mean quality of the totality of their bases. If this mean exceeds the given pair mean by more than 20, pair correction and filling are not carried out.

Next we remove friends having 'inadequate glue' to the founder. This is carried out for each read in the pair. For each friend, we find its maximal intervals of agreement with the founder. In such intervals, homopolymers longer than 10 are compressed to 10. To be adequately glued, after this compression, there must be an interval of agreement of length $\geq 20$.

Next we raise quality scores within the stacks (and this is internal to pair correction and filling). For each of the two founders and for each position on it that has quality > 0 but < 30, we consider the possibility of raising the quality to 30. To do this, we form the 11-base window centered at that position. We compute the number of friends that agree with the founder on this window and have quality $\geq 30$ at the middle base. There must be at least 3 such reads to proceed. Next we look for friends that agree with the founder on the window, except not on the middle base, and have quality $\geq 30$ at the middle base. If in this process we find 3 or more friends that assert some particular alternative middle base, we do nothing. Otherwise we raise the quality score at the founder position to 30.

Now we look for motifs that might allow deletion of more friends. We scan 10-base nonoverlapping windows on each founder. We find all sequences that occur in the stack on this window. Suppose that the founder sequence appears at least 10 times. Consider alternative sequences that occur at least 10 times, and for which at least one friend has quality $\geq 20$ at the middle base. All the friends carrying these sequences are deleted from the stack. After this we clean each stack by removing friends having a Q30 difference with the founder.

Next we compute the consensus for each stack, as in Procorrection 2. We then reverse complement the right stack and compute possible offsets for the relative position of the two stacks. This involves finding possible overlaps for the two consensus sequences. We start by finding 8-mer matches between them. These define an initial set of offsets. For each offset, we define a score, its number of 'bits'. This is computed by examining each subwindow of length at least 20 within the overlap, and defining the number of bits for that subwindow. The number of bits for the offset is defined to be the maximum of the bits for the subwindows. Subwindows contains a 40-base window having $\geq 20$ mismatches are ignored. Let k be the number of mismatches in the subwindow, and n its length. Then the number of bits in the subwindow is defined to be
$-(5/3)\log_{10}(\text{ BinomialSum}( n, k, 3/4 ) )$, where $\text{BinomialSum}(n,k,p) = \Sigma_{i=0}^{k} (n \text{ choose } i)p^{i}(1-p)^{n-i}$. We observe empirically that for a perfect overlap, the number of bits is close to the length of the overlap. Offsets having < 25 bits are ignored. Offsets that imply a Q30 mismatch between the founder reads are ignored.

Next we go through a process that allows certain offsets to 'invalidate' certain other offsets. For this, for each offset, we first find the intervals of perfect agreement between the two consensus sequences. We exclude the first and last 10 bases from each such interval, and mark the two consensus sequences con1, con2 as confirmed on the remaining bases. These markings are separately tracked for each offset. Next we say that offset i invalidates offset j if offset j implies that $\text{con1}[p1] \neq \text{con2}[p2]$ for some p1, p2, but offset i confirms both of these positions. Finally suppose that for some i, offset i is not invalidated by any other offset, but offset i invalidates offset j. Then offset j is deleted.

Now suppose that offset j has < 40 bits, and that offset i has at least 10 more bits than it. Suppose that the 'slope' (bits i – bits j) / |offset i – offset j| is $\geq 2$. Then we delete offset j. Note that the slope condition is designed to protect against deletion of alternative offsets that arise from tandem repeats. This completes the computation of offsets.

For each offset, we now merge the left and right stacks, creating a 'joint' stack. In cases where a (read, orientation, offset) triple appears twice, only one is retained. Quality scores of the two founders (now in the joint stack) are again raised, as described previously. After this we clean the joint stack by removing friends having a Q30 difference with one or both of the founders. If this removes one or both of the founders, we remove the offset from consideration.

Next, for each column in the joint stack, we compute the quality score sum for each of the four possible base calls, but using only those reads whose partner is placed at least once in the stack. If the winning sum is $\geq 100$, at least 10 times the runner up sum, and the runner up sum is < 100, we remove any friend that disagrees with this winning call and has quality $\geq 30$ there. If this removes one or both of the founders, we remove the offset from consideration.

Now we create consensus and consensus quality scores for the joint stack. The consensus is computed as was the consensus for the founders. To compute the consensus quality scores, we first raise quality scores on each friend based on its agreement with the consensus, as follows. Suppose that a base on a friend and its d flanking bases on both sides agree with the consensus. Then if the quality score is greater than zero but less than $10*\log_{10}(2d)*0.5$, we raise it to this number. Then we compute the quality score sums for each column. The consensus quality score for a given column is set to Min( winning sum – runner up sum, 50 ). However we then test for inconsistency in each column, as follows. Suppose that the runner up sum is > 100 and supported by at least two Q30 bases. Then we declare the column inconsistent and set the consensus quality score to zero.

Next we 'protect' certain bases in the founders. We take the first 10 bases on the left. If the left founder disagrees with the consensus and has quality ≥ 20 there, we force the consensus to be that base, and also copy the quality score to the consensus. The same operation is carried out on the right.

Next if any base on a founder has quality ≥ 30 and disagrees with the consensus, we set the consensus quality to zero.

We next check for suspicious inconsistencies between the founders and the consensus. Suppose that a founder base disagrees with the consensus, but that its 5 flanking bases on both sides agree. Suppose that there are at least 3 non-founder rows in the joint stack that agree with the founder at the base and on the 5 flanking bases on both sides. Then we set the consensus quality to zero at the base.

We now attempt to recover conflicted columns in the joint stack. These are columns having a 'low' consensus quality score. First let minq_floor be 10 if there is more than one offset, else 5. Consider columns in the joint stack for which the consensus quality value is less than minq_floor, however for which at least one founder has quality ≥ 2. If both founders have quality ≥ 2 and disagree and the difference between their quality scores is < 10, we do not consider the column. We choose the founder base having the highest quality score. Then we delete any non-founder row that disagrees with the base and has quality ≥ 2 at the position. We again compute the consensus and consensus quality scores for the joint stack, as described above, and protect certain bases in the founder, as above.

Now we decide if the closure is accepted. There are two criteria. First the minimum consensus quality must be ≥ 10 (or in the case of a single offset, ≥ 5). Second we impose the following 'minimum glue' requirement. Using only intervals of perfect overlap of length ≥ 40 between the joint consensus and some stack row, it must be possible to walk from left to right along the consensus, using the intervals, and requiring that consecutive intervals overlap by ≥ 30 (or 20, for a single offset).

Now we examine all the closures (at most one per offset). We note the following special case, not described in the main text: where all the closures are the same, except for differing homopolymer lengths at a single locus, they are all retained. Otherwise, we let L denote the maximal leftmost segment where all consensus sequences agree, and let R be the maximal rightmost segment. If L = R we report L alone. Otherwise we report both L and R.

We next attempt to identify pairs that were not closed, but which appear to bridge gaps. To do this, first we find all 40-mers x that appear at least 5 times in the initially corrected reads and their reverse complements. If such an x lies on a closure c, so that x's beginning lies at least 200 bases from the right end of c, we mark x as *right-extended*. An x which is not right-extended is said to *fail*. An unclosed read pair is now declared *special* if one of its reads contains a failing kmer or contains the reverse complement y of a failing kmer x, where the right end of y lies at least 200 bases from the beginning of the read. The entire pair correction and filling step is now repeated for these special read pairs, however with more lenient parameters, so as to increase the likelihood of closing the pair:

- In the exclusion of low-quality pairs, the mean is allowed to exceed the given pair mean by up to 25.
- When computing consensus, we allow quality scores of zero to be raised.
- Minimum glue is reduced to 15.
- The minq_floor parameter is taken to be 0 if there is only one offset.

This completes the error correction and pair closure part of the assembly process.

**Graph formation**. The corrected pairs are now formed into an initial assembly graph. This graph is a directed graph, whose edges are DNA sequences, and such that abutting edges overlap by K-1, where K is 0.18 times the median pair closure length, rounded to a fixed set of values ..., 60, 72, 80, 84, 88, 100, ... . For the data type described in this work, the typical value is around 80. The initial assembly graph is exactly the unipath graph[33].

Each pair closure c and its reverse complement are mapped to the graph and recorded as sequences of edges within it, which we refer to as *closure paths*. We also refer to the *median closure path length*, measured in kmers. Below, if we refer simply to *paths*, we are referring to arbitrary paths within the assembly graph.

The assembly graph data structure records counts for each observed edge sequence $e_1,\ldots,e_n$, denoted $|e_1,\ldots,e_n|$. Similarly we would write *e.g.* |e| or |ef| for the inferred sum associated with one or two edges. We let len(e) denote the length in kmers of an edge e.

The data structure also includes an 'involution' *i* that maps each edge to its reverse complement, and having the property that $i^2$ is the identity map. These data structures are maintained as the assembly graph is modified. During this process, the involution *i* is allowed to deviate in two ways from the original definition:

- while *i* always maps an edge e to *a* reverse complement of e, there may be more than one such reverse complement, and thus i must pick between them;
- in cases where a component c of the graph is disjoint from its reverse complement, we allow the reverse complement to be deleted, and in such cases *i* is undefined on c.

**Graph simplification and improvement**. The assembly graph is now operated on iteratively, using several different steps that are described below. We note that in some cases the operations as described are asymmetrical, and in such cases the same operation is carried out on the reverse complemented assembly.

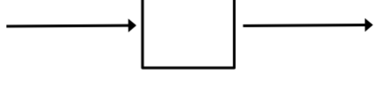
*Reverse complement removal*. When both a graph component and its reverse complement are present and disjoint, one is removed.

*Hanging end removal*. First, for each vertex v, form the set of paths that start at v but contain no edge twice. This set is computed iteratively, and if at a given stage we obtain > 100 paths, the process is terminated and the computation deemed incomplete. Let D[v] be the maximum length L in kmers of all these paths. Now consider a vertex v having multiple edges $e_i$: v → $w_i$ emanating from it. Let d[i] = L[$e_i$] + D[$w_i$] and order the edges so that d[0] is greatest. If for some i, d[i] ≤ 1000, d[0] ≥ 10·d[i], and the computation for D[$w_i$] was complete, delete edge i. Also remove terminal loops of length ≤ 50 kmers.

*Remove small components*. Acyclic components having ≤ 1500 – K + 1 kmers are deleted, as are cyclic components having ≤ 50 kmers.

*Delete low coverage edges*. For an edge e, suppose that |e| ≤ 2, and that there is another edge f, either starting at the same vertex that e starts at, or ending at the same vertex that e ends at, and that |f| ≥ 5·|e|. Then we delete edge e.

*Assembly unwinding*. This is a method for simplifying the assembly graph. It looks for loci of the form



where the box represents 'something between' two edges. The method proceeds according to the following steps:
1. We consider every assembly edge as a potential left bounding edge in the above diagram. We then walk right from the edge (as described below), eventually defining the right bounding edge.
2. Starting from the left bounding edge, we iteratively walk right, allowing for branching, until we have 10 paths or no more are possible. We terminate a given path as soon as it contains 20 edges. This

prevents infinite looping around a circular genome. At each iteration, we extend a path having the least number of edges in it.
3. As we walk right, certain paths are excluded. In particular, we reject a path if it contains a subpath a,b1,...,bn,c, seen in the closure paths ≤ 2 times, and such that the total kmer count in the bi is ≤ the median closure path length, provided that for some c', we see a,b1,...,bn,c' at least 5 times more often. The use of the median closure path length circumvents a complex statistical problem relating to how often we would expect to see a particular sequence. It is likely that more repeats can be resolved by pushing this limit.
4. Next we look for a common truncation of the paths on the right so that they all terminate in a common edge (the right edge in the above diagram). Having done this, we then define the 'box' in the middle of the above diagram. We require that the paths traverse essentially all of its edges: if there is an untraversed edge having weight ≥ 5, we reject the box, truncate all the paths on the right, and try again.
5. If we find a common right truncation that is acceptable, the resulting data define a recommendation, namely that the entire diagram be replaced by the set of truncated paths. These paths still have to be squeezed back into a graph (see below).
6. We prune the recommendations to eliminate conflicts.
7. Finally, for each surviving recommendation, we squeeze its paths into a graph. Then the graph is inserted back into the main assembly graph, replacing the diagram.

*Pull apart simple branches*. Consider a diagram in which there is an edge r, with exactly two edges $x_1$, $x_2$ abutting r on the left, and exactly two edges $y_1$, $y_2$ abutting r on the right. Let $w_{ij} = |x_i r y_j|$. Suppose either that:

- $w_{11} \geq 2$, $w_{22} \geq 2$ and $w_{12} = w_{21} = 0$, or
- $w_{11} \geq 5$, $w_{22} \geq 5$, $w_{12} + w_{21} \leq 1$ and len(r) ≤ median closure path length, or
- $w_{11} \geq 10$, $w_{22} \geq 10$, $w_{12} + w_{21} \leq 2$ and len(r) ≤ median closure path length.

Then replace all five edges by two new edges $z_1 = x_1 r y_1$, $z_2 = x_2 r y_2$.

*Pull apart complex branches*. We look for a complex with two edges $e_1$, $e_2$ entering and two edges $f_1$, $f_2$ exiting. Let $w_{ij}$ be the number of closure paths that contain $e_i \ldots f_j$, *i.e.* $e_i$ followed eventually by $f_j$. Then test the same criteria for the $w_{ij}$ as in the pull apart of simple branches. If successful, we replace the entire complex of edges by the graph obtained from all closure paths of the form either $e_1 \ldots f_1$ or $e_2 \ldots f_2$.

*Bubble popping*. We look for edges $e: v \rightarrow w$ whose multiplicity is ≤ 2. Then we find all alternative paths from v to w whose length in kmers is within 10 of len(e), and whose multiplicity is > 2. Suppose that there is a unique such alternative path p, that its length in kmers is within 2 of len(e), and that $|p| \geq 5 \cdot |e|$. Then we delete edge e and replace all occurrences of it by p.

*Graph reconstruction*. For this method, it is convenient refer to the closure paths as words on the alphabet of edge indices. The idea of graph reconstruction is to define a collection of words that are 'trusted', and a collection of overlaps between them that are 'trusted', then formally glue the words together with each other along the trusted overlaps, yielding a new assembly. In describing graph reconstruction, we describe a general framework for pulling apart a graph assembly, together with an initial heuristic model that defines 'trusted'.

The process starts by finding all the overlaps between all the words. Here, by an overlap, we mean a proper overlap, *i.e.* extending fully to the left on at least one word and extending fully to the right on at least one word.

The main text illustrates graph reconstruction using the example of **Supplementary Fig. 5**. In general, with the goal of eliminating some overlaps (and in the process, also eliminating some words), we first find all subwords that occur as the overlap between two words. Each of these subwords s now seeds an analytic step, as follows. We form the matrix whose rows consist of all the words containing s. We also track the weights associated with each word. This is illustrated in **Supplementary Fig. 5**.

For the general method, we proceed in the following fashion. Let left_exts denote the lengths in kmers of subwords appearing to the left of the seed, and let right_exts denote the lengths on the right side. Next we traverse all possible choices for an element of left_ext and of right_ext. (If there would be > 1000 such pairwise choices, the calculation for the seed is terminated.) We now modify the matrix, local to the calculation for these two elements.

For each row of the matrix, walk left from the seed, counting kmers. As soon as the chosen number from left_ext of kmers is achieved, declare the row to be left_full, and blank out all further entries to the left. Do the same thing on the right. Call a row full if it is both left_full and right_full. Merge rows that are now identical, and in so doing, sum their weights. We require that the highest weight row has weight ≥ 10, and that it is full. (Otherwise the computation for the given left_ext/right_ext elements is terminated.) This condition establishes strong coverage at the locus.

Now define potential joins. These are obtained by taking a left_full row that stops on the right after the seed, and a right_full row that stops on the left before the seed. Each such pair defines a join, and the unique joins (not already in the matrix) are merged in with weight zero.

Call a row a keeper if it is full and its weight is at least 2 or else it exceeds 10% of the highest weight. Call a row a follower if it is non-full and it is a subrow of a keeper.

Finally, consider non-full rows i that are not followers. Let 'total' be its weight, plus the weight of all non-full non-follower rows that are subrows of i. Let 'control' be the maximum weight over all non-full rows whose left and right extent are at least that of i. Declare i to be a loser if total ≤ 2 and 10·total ≤ control. From each loser row we infer that certain overlaps should be deleted, and also in some cases that certain words should be deleted. This completes the main computation of the algorithm.

At this stage we have a collection of words and a collection of accepted overlaps between them. We first form the digraph that is the disjoint union of the words, with each word stretched out over a sequence of edges (corresponding to the letters in the word). Next from the accepted overlaps we deduce an equivalence relation on the vertices and edges of this graph. Now if the resulting quotient graph would have cases where two edges labeled by the same letter both exit from source vertices (but not necessarily the same one), identify the two edges. Carry out the analogous operation for sink vertices. Next extend the equivalence operation by 'zippering up', so that the quotient graph does not have cases where identically labeled edges exit the same vertex (or enter the same vertex). Finally, form the quotient graph. This is the new graph assembly.

*Make and unroll loops*. Making loops: when we have a picture like this

$$\begin{tikzcd} u \arrow[r, "e"] & v_1 \arrow[r, bend right, "f_2"'] & v_2 \arrow[l, bend right, "f_1"'] \arrow[r, "g"] & w \end{tikzcd}$$

replace $f_1$ and $f_2$ by a loop. Now unroll loops, as follows. First find loops. Loops whose length in kmers exceeds the median closed fragment length are ignored. Find the numbers c of consecutive loops (zero or more) that are observed in closure paths that include both bounding edges. Also find the numbers of consecutive loops (one or more) that occur in any closure path. If the second count set includes a larger number than the first, do nothing. Otherwise, replace the loop by the linear paths defined by c.

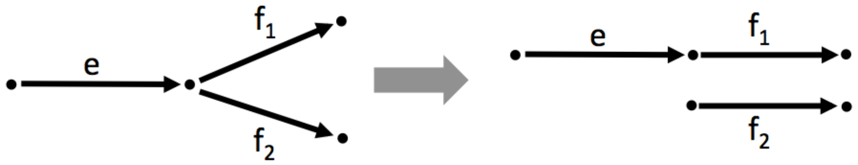
*Remove weakly supported loops*. Consider vertices v having a 'canonical' loop f

$$\begin{tikzcd} u \arrow[r, "e"] & v \arrow[loop above, "f"] \arrow[r, "g"] & w \end{tikzcd}$$

Suppose that $|eg| > 0$. If $|ef| \leq 2$ and $|eg| \geq 10\cdot|ef|$, delete edge f. If $|fg| \leq 2$ and $|eg| \geq 10\cdot|fg|$, delete edge f.

*Delete weakly competing edges*. Assign entering and exiting weights enter[e], exit[e] to edges e. Each closure path containing the edge and an edge after it contributes to the entering weight, whereas each closure path containing the edge and an edge before it contributes to the exiting weight. For each vertex v, find a neighborhood that starts from v. This neighborhood consists of all vertices within distance 4 of v, but if there are more than 20 such vertices, the computation is aborted. Consider an edge e starting from v, and edges $e_1$ and $e_2$, starting from vertices $v_1$ and $v_2$ in the neighborhood. Suppose that any path in the graph that starts from e must ultimately pass through either $e_1$ or $e_2$. Suppose that $\text{exit}[e_1] \leq 2$, $\text{enter}[e] \geq 10 \cdot \text{Max}(1, \text{exit}[e_1])$, and that $\text{exit}[e_2] \geq \text{Max}(1, 10 \cdot \text{exit}[e_1])$. Then we delete edge $e_1$.

Now consider vertices that have one edge e entering, and two edges $f_1$, $f_2$ exiting. Let $w[i] = |ef_i|$, $i = 1,2$. Reorder if needed so that $w[1] \geq w[2]$. If $w[2] \leq 2$ and $w[1] \geq 10 \cdot \text{Max}(w[2], 2)$, separate $ef_1$ from $f_2$:



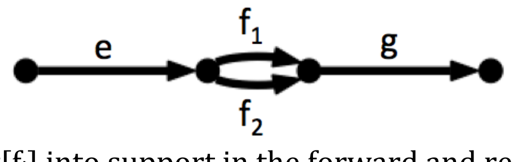
*Graph cleaning using the uncorrected reads*. In this step we find gap-free alignments between the uncorrected reads and the assembly graph. Such an alignment goes from end to end on the read, and lands on a path of edges within the graph. It is scored by taking the sum of read quality scores > 2 at mismatches. Only alignments that begin with a 12-mer match are used. Only the alignment(s) having the best score are returned.

Define the support of an edge e to be the sum over all instances of e within an alignment as above, of 1/n, where n is the number of equal-scoring alignments of the given read.

Consider a simple branch in the graph, given by an edge e and exactly two edges $f_1$, $f_2$ that follow it. Suppose that $\text{support}[f_2] \leq 1$ and that $\text{support}[f_1] \geq 5$. Then we delete edge $f_2$.

Consider each vertex v, and consider two edges $e_1$, $e_2$ emanating from it. We assume that neither edge starts with a homopolymer of length 10 or greater. First we find all instances of the first kmers of $e_1$, $e_2$ in the uncorrected reads. Then we compute the quality score sums $q_1$, $q_2$ associated to the last base on these first kmers. We require that $q_2 \geq 100$, $q_1 \leq 100$, and that $q_2 \geq 10 \cdot q_1$. This is a prefiltering step. Next we consider not just the first kmers on $e_1$, $e_2$, but up to K kmers (and only as many kmers as are present in the shorter edge). On the first kmer we consider the last base; on the second kmer we consider the next to the last base, and so forth. Again we compute the quality score sums, but not counting the same position on a given read more than once. We also compute the location counts $c_1$, $c_2$ associated with these sums. We require that $c_2 \geq 8 \cdot \text{Max}(1, c_1)$, and we impose the same conditions as before on $q_1$, $q_2$. If all these conditions are satisfied, we delete edge $e_1$.

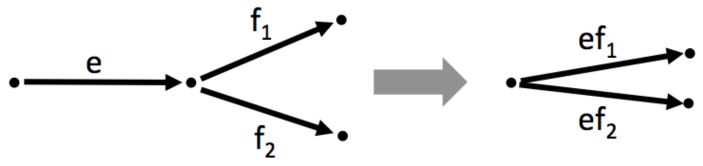
Now consider a simple bubble



We subdivide support[$f_i$] into support in the forward and reverse orientations, defining $F_i$ and $R_i$. Let $p = \text{Min}(0.5, F_1/(F_1+R_1)) / 2$. Let $n = \lfloor F_1+R_1+F_2+R_2 \rfloor$. Let $q = \text{BinomialSum}(n, \text{ceil}(F_2), p)$. If $q < 10^{-5}$ and $F_2 + R_2 < 10$, delete edge $F_2$.

Now consider simple bubbles corresponding to one or more substitutions. Consider reads that place uniquely on one branch of the bubble, and score their placement on the other branch. The differences between the scores are assigned to the winning branches. These differences comprise two distributions, and we check to see if one has a 'much' larger mean than the other, and in that case delete the branch with the smaller mean. To measure the difference, we assign to each distribution the associated normal distribution that reflects our expectation of its true mean. Then we take the difference of these two normal distributions. If the mean of the difference distribution is ≥ 3 standard deviations away from zero, we deem one distribution to be much larger than the other.

*Gulp edges*. Whenever we have an edge e of length ≤ 20 kmers, abutting two edges $f_1$ and $f_2$, replace the three edges by two edges $ef_1$, $ef_2$.



The opposite process is also carried out. Note that these operations do not change the semantics of the assembly graph.

*Orient assembly to reference*. If a reference is provided, reverse complement any components that appear to be in reverse orientation to the reference.

## 6. DISCOVAR variant calling method

<u>Best path construction</u>. The first stage in the calling of variants is to find a path through the assembly graph that is as close as possible to the reference sequence. This proceeds in several steps. First, in cases where the reverse complement of assembly components have been deleted during the assembly process, we add them back. Next, we align each assembly edge to the reference. These alignments are seeded on perfect 40-mer matches with the reference:

1. Ignore kmers in an edge mapping to too many places in the reference, as follows. First let n be the length of the edge in kmers. Then to each position p in the edge, let m(p) denote its multiplicity, the number of times that the kmer starting at p occurs in the reference. Define the maximum multiplicity to be the smallest M such that for ≥ n/2 positions, 1 ≤ m(p) ≤ M. Positions p for which m(p) > M are ignored.
2. Group the matches and reject some groups, as follows. Sort the offsets on the reference associated to each accepted position on the edge. Group the offsets, breaking groups when their difference exceeds 10. Now reverse sort the offset groups by np = the number of positions on the edge that they correspond to. Keep only the offset groups having at least 75% of the maximum np.
3. For each accepted offset group, align the edge to the reference, using a banded Smith-Waterman algorithm. Alignments having an error rate > 5% are rejected.
4. For each alignment of each edge, define its start and stop positions. The stop position is defined to be K-1 bases from the right end, where K is from the assembly graph.

Define a new graph Z whose vertices are pairs (v,p) where v is a vertex in the assembly graph and p is a position on the reference occurring as a start or stop position. Via its start and stop positions, each edge alignment defines an edge in Z. These start and stop positions are perturbed slightly to avoid cases were there are two consecutive edges e, f in the assembly graph, mapping to edges e', f' in Z, and target(e') ≠ source(f') because of slight differences in reference positions. Such instances are associated with alignment indels occurring near the ends of edges.

Whenever there is a sink and a source in Z, associated to reference positions within 10 kb from each other, connect them by a 'gap edge'. For each chromosome in the reference, add a 'left end' vertex and gap edges from it to source vertices on the chromosome. Add a 'right end' vertex and gap edges from sink vertices on the chromosome to it.

Using Djikstra's algorithm, we find shortest paths through Z from left end to right end vertices for each chromosome. To do this, for each path we let e denote its number of errors (mismatch plus indel bases), g denote its number of gaps, and d its number of gap bases, where the gap bases are computed from the gap edges. Then we find the path that minimizes the penalty e + 100*g + d/100. This is the best path.

Linearized assembly graph construction. The best path is used to form the backbone of a new graph, to which will be added appropriate alternative paths found in the original assembly graph. Initially this graph consists of only the edges and vertices contained within the best path. Each edge or vertex corresponds to an edge or vertex in the original assembly graph. As this new graph has been 'unrolled' along the reference, it is possible that edges and vertices from the original assembly graph may be repeated at different locations in this graph.

For each vertex in the best path graph, each corresponding vertex in the assembly graph is found. The assembly graph is then explored from this point using a depth first search, looking for those paths which reach another vertex contained in the best path. These paths provide alternatives to that defined by the best path. An alternative path is considered valid if it: rejoins the best path downstream of the starting vertex (*i.e.* no cycles are allowed); the length of the path is less than 2000 bases; the length of the path is within 150 bases of the length of corresponding section of the best path; the path is not identical to the best path. All valid alternative paths are then added to the best path graph, forming a linearized acyclic assembly graph that contains both the best path and a subset of the alternative paths present in the original assembly graph.

Bubble graph construction. The linearized assembly graph is further simplified, replacing any complex graph features with sets of parallel edges to form a 'bubble' graph (terminology explained below). To do this, 'anchor' edges are chosen: these are edges in the linearized assembly graph that lie in the best path and such that every path through the graph must traverse them (so that there is no 'parallel' edge).

The graph between each pair of consecutive anchor edges is expanded into a set of parallel edges, enumerating all possible alternative paths between the anchor edges. This yields a new 'bubble graph', which consists of series of unambiguous anchor edges separated by bubbles containing two or more alternative edges in parallel. Generally, anchor edges carry homozygous variants, whereas bubble edges carry heterozygous variants.

Variant detection. Variants are detected from the bubble graph in two stages. First, the anchor edges are aligned to the reference using an affine Smith-Waterman algorithm. Alignment differences between the reference sequence and the anchor edges are marked as potential variants. Second, the parallel edges within each bubble are examined. These edges are aligned to the reference using an affine Smith-Waterman algorithm, with alignment boundaries constrained by the adjacent anchor edges' alignments. For each edge in the bubble, differences from the reference are marked as potential variants. The same variant may appear across multiple parallel edges, and variants found on the same edge are phased together.

Variant masking. Some variants calls occurring in repetitive regions of the genome are eliminated. Prior to running DISCOVAR, a repeat mask for the reference genome is prepared. Each 100-mer in the reference is identified, and if its multiplicity is greater than one, then all the bases in the reference corresponding to the 100-mer are masked out. Any edge of less than 600 bases in the assembly graph that corresponds to a masked section of the genome is ignored when calling variants.

Variant graph construction. The potential variants found from the bubble graph are now used to construct a new 'variant graph' for the purpose of computing variant call probabilities. Starting with the reference, additional edges are added to represent each variant found in the variant detection step, forming a series of bubbles containing two or more edges. Where variant bubbles overlap they are merged, resulting in larger bubbles. To reduced the computational complexity of the next step, variant bubbles are merged together if they are separated by 20 bases or less, and the total span of the merged variants is also less than 20 bases. Variant bubbles are merged using phasing information obtained from the corresponding edges in the bubble graph.

Probability computation. Variant probabilities are computed from gap-free alignments of reads to the edges of the variant graph. A score is associated with each read alignment, computed as the sum of the quality scores for only the mismatched bases in the alignment - a perfect alignment having score 0. The best (lowest scoring) alignment to a variant edge is taken, and a delta quality score computed by subtracting the score for the 2nd best alignment of the read. This is repeated for all reads that align to a variant edge, and the delta quality score is summed. From the delta quality score sums for each variant edge in a bubble (which measures the preference of reads supporting this edge), a probability can be calculated for the variants represented by that edge using the expression $P = 1 - 10^\wedge(\Sigma(\text{delta } Q)/10)$.

Variant list. Finally, a standard VCF file is generated using the positions and probabilities found above. In addition, a DISCOVAR-specific .variant file is generated, containing additional information not easily represented in VCF format.

## 7. Generation of variant calls

All the variant calling programs that we used produced as output a list of variants, each of which was either a substitution or an indel. In comparing variant calls, we treated two variants as equivalent if their effect on the genome (as an edit) was the same. We note that were cases where variants could be more efficiently represented as inversions, however there was only one longer than 25 bp, and was represented as an inversion plus a deletion plus a substitution.

DISCOVAR calls were filtered as noted below, and are those labeled PASS in the .filtered.vcf file. For Platinum-100 and Cortex, we used the calls labeled PASS. For GATK-250, we used the calls that were labeled PASS and that had QUAL ≥ 180 (for SNPs) and QUAL ≥ 100 (for indels), as per the suggestion of Heng Li (personal communication).

The VCF files that we generated as part of this work are available at ftp://ftp.broadinstitute.org/pub/crd/DiscovarManuscript.

### *7a. Generation of variant calls from 250 base reads using DISCOVAR*

To prepare a BAM file for DISCOVAR, reads were aligned to the hg19 human reference sequence using BWA[37] version 0.5.9 (http://github.com/lh3), as part of the Picard pipeline http://picard.sourceforge.net. Each lane was aligned separately using "bwa aln" with parameters "-q 5 –l 32 –k 2 –o 1," followed by "bwa sampe" with options "–a 1350." The resulting BAM file was sorted in coordinate order using the SortSam module of the Picard tools (http://picard.sourceforge.net).

Before running DISCOVAR, several auxiliary files must be generated from the reference using:

```
PrepareDiscovarGenome REF=Homo_sapiens_19.fasta
```

This step requires SamTools (http://samtools.sourceforge.net), and needs to be done only once per genome.

The following command calls variants  on a region using DISCOVAR:

```
Discovar REFERENCE=Homo_sapiens_19.fasta READS=<input bam file> REGIONS=<chr>:<start>-
<stop> TMP=<tmpdir> OUT_HEAD=<assembly>
```

where `<chr>:<start>-<stop>` represents a coordinate range, such as 2:1000000-10030000 for the 30 kb region of chromosome 2 starting 1,000,000 bases in.  We used revision 47606 of the DISCOVAR code.

The called variants are in the file `<assembly>.final.variant.vcf`. For each variant line in the file, probabilities of presence are assigned to each allele via fields REFP and ALTP, which we used to define passing variants, for purposes of this work:

- alleles were treated as *called* if they were assigned probability > 0.995;
- variant lines in the file were ignored completely if any probability was > 0 and < 0.9;
- lines were also ignored if only the reference was called, or more than two alleles were called.

Genotypes for these passing variant lines were then inferred from the alleles that were called. A VCF file with these variants marked as passing (PASS) is also generated. It is `<assembly>.final.variant.filtered.vcf`.

To run DISCOVAR on the entire genome, we called variants in 50 kb chunks having 10 kb overlaps. We used a cluster of seven Dell PowerEdge R815 servers with 2.8 GHz AMD Opteron 6348 processors. There were 48 cores and 256 GB RAM per server.  We ran three simultaneous jobs per server. We terminated 0.16% of jobs because they used more than 30 minutes. These were typically associated with regions having high aligned coverage (for example, pericentromeric regions). Apart from the time-outs, every job succeeded. The resulting variant calls were merged into a single file using combine_region_vcfs.py, which comes with DISCOVAR. The total elapsed time was 47 hours. We note that as the total cluster cost was about $66,000, and assuming that the hardware cost was amortized over a three year period, ~$100 would be contributed to total costs, not counting overhead.

### *7b. Generation of variant calls from 100 base reads using GATK*

The GATK calls for 100 base reads for NA12878 and her father were embodied in Variant Call Format (VCF) files NA12878_S1.genome.vcf and NA12891_S1.genome.vcf distributed by Illumina as part of the Platinum Genomes project (www.illumina.com/platinumgenomes).

### *7c. Generation of variants calls from 250 base reads using GATK*

We followed GATK best practices (http://www.broadinstitute.org/gatk/guide/topic?name=best-practices), adapted for 250 base reads from a PCR-free library:

1. The process starts with FASTQ files, denoted here `H01UJ.1.fastq` and `H01UJ.2.fastq` corresponding to the two lanes of data used.
2. Paired reads were aligned using BWA-MEM and sorted, per-lane, and the resulting files were merged and indexed.
3. Indels were realigned using GATK RealignerTargetCreator and IndelRealigner.
4. Quality score were recalibrated using GATK BaseQualityScoreRecalibration.
5. Variants were called using GATK HaplotypeCaller.
6. Variants were recalibrated using GATK VariantRecalibrator and ApplyRecalibration.

We note that to run GATK optimally, several parameters had to be chosen by hand. This is described in the following text, and the table below it. In addition we imposed added filters as described in **Supplementary Note, Section 7**, above.

Prior to filtering, GATK generates a set of calls designed to be highly sensitive at the expense of specificity. The Variant Quality Score Recalibration (VQSR) process is then used to "filter" this set with the goal of eliminating false positives. The HaplotypeCaller provides various metrics, known as "annotations," for each variant generated, and the VQSR process uses these annotations as a machine-learning feature-space, in order to identify outliers. VQSR fits a Gaussian mixture model to the annotations, and then computes the log-odds, under that model, that a particular variant is true. The final choice of how variants are thresholded, and therefore the final sensitivity-specificity tradeoff, is left as a choice for the end user. The GATK online guide recommends (http://www.broadinstitute.org/gatk/guide/article?id=1259) using 99.9% as a starting value, but notes that this may only be appropriate for multi-sample studies with large population diversity. VQSR calculates a minimum log-odds score, at various threshold values, for the calls made, and we used this value to guide selection of a threshold (--ts_filter_level for ApplyRecalibration), attempting to find the point where the minimum log odds turned from negative to positive and was rising rapidly. For NA12891, the log-odds were not rapidly rising as the threshold reduced, so the highest threshold at which the minimum log odds was greater than 2.0 was chosen. Finally, during the training phase of VQSR, several model parameters needed to be changed in order to achieve convergence of the model. Values chosen for each of these parameters is shown in the table below.

| **sample** | **--ts_filter_level** | | **--maxGaussians** | | **--numBadVariants** | |
|---|---|---|---|---|---|---|
| | **SNP** | **INDEL** | **SNP** | **INDEL** | **SNP** | **INDEL** |
| NA12878 | 99.9 | 99.5 | default | 4 | 3000 | 1000 |
| NA12891 | 99.8 | 97.0 | 4 | 4 | 6000 | 10000 |

Software Versions. Unless otherwise noted, GATK version 2.4-9 was used for GATK processing steps 1 through 4 above and GATK version 2.7-2-g6bda569 was used for steps 5 and 6. The switch from GATK 2.4 to GATK 2.7 was made mid-stream in order to avail ourselves of recent advances in the HaplotypeCaller made just prior to completion of this manuscript. In a few cases, a nightly update (vnightly-2013-04-04-g41a26fa) was used due to the inclusion of critical bug fixes that impacted our analysis. The BWA-MEM alignment program (https://github.com/lh3) used was version 0.7.3-r376-beta. The Picard Tools (http://picard.sourceforge.net) version 1.90 were used to sort and merge SAM files.

Description of input data files.

Two sets of auxiliary data were required for the analysis. In the documentation that follows, the file `Homo_sapiens_assembly19.fasta` refers to the human genome reference assembly GRCh37/b37 available from ftp://ftp.broadinstitute.org/pub/seq/references. The following variant call format (VCF) files are known variant sites used by various stages of GATK's analysis pipeline: `Mills_and_1000G_gold_standard.indels.b37.sites.vcf`, `1000G_phase1.indels.b37.vcf`, `hapmap_3.3.b37.vcf`, `1000G_omni2.5.b37.vcf`, `dbsnp_137.b37.vcf`. See http://gatkforums.broadinstitute.org/discussion/1247/what-should-i-use-as-known-variantssites-for-running-tool-x#latest.

Step 1 – Obtaining the Reads.

The data can be obtained from NCBI's SRA repository as experiment number SRX297987:

http://trace.ncbi.nlm.nih.gov/Traces/sra/sra.cgi?exp=SRX297987&cmd=search&m=search&s=seq
We suggest downloading each lane of data as separate FASTQ files and naming as follows:
SRR891258 → H01UJ.2.fastq, SRR891259 → H01UJ.1.fastq.

Step 2 - Alignment, Sorting, Indexing. We used BWA-MEM (http://github.com/lh3/bwa, http://arxiv.org/abs/1303.3997v2) to align reads to the hg19 reference. BWA-MEM takes a reference FASTA file and reads to align in FASTQ format. We used BWA-MEM's pairwise alignment procedure and provided a pair of FASTQ files. BWA-MEM includes an option (-R) to graft readgroup (@RG) information on to the resulting BAM. This step also uses SamTools (http://samtools.sourceforge.net).

```
RG="@RG<TAB>ID:H01UJ.1"          # note: replace <TAB> with a tab character
bwa mem -p -t 24  -R "$RG" Homo_sapiens_assembly19.fasta H01UJ.1.fastq | samtools view -b -S -o
H01UJ.1.aligned.wholegenome.bam -

RG="@RG<TAB>ID:H01UJ.2"          # note: replace <TAB> with a tab character
bwa mem -p -t 24  -R "$RG" Homo_sapiens_assembly19.fasta H01UJ.2.fastq | samtools view -b -S -o
H01UJ.2.aligned.wholegenome.bam -
```

Each lane was sorted:

```
java -jar SortSam.jar TMP_DIR=tmp I=H01UJ.1.aligned.wholegenome.bam
O=H01UJ.1.aligned.wholegenome.sorted.bam MAX_RECORDS_IN_RAM=100000000 SORT_ORDER=coordinate
```

And finally, the lanes were merged:

```
java -jar MergeSamFiles.jar I=H01UJ.1.aligned.wholegenome.sorted.bam
I=H01UJ.2.aligned.wholegenome.sorted.bam O=H01UJ.12.aligned.wholegenome.sorted.bam
USE_THREADING=true MAX_RECORDS_IN_RAM=50000000
```

Step 3 - Indel Realignment. The GATK RealignerTargetCreator was used to determine intervals likely to need realignment:

```
java -jar GenomeAnalysisToolkit.jar -T RealignerTargetCreator -nt 45
-I H01UJ.12.aligned.wholegenome.sorted.bam -R Homo_sapiens_assembly19.fasta
-o H01UJ.12.aligned.wholegenome.sorted.indel_cleaned_local.intervals
--known Mills_and_1000G_gold_standard.indels.b37.sites.vcf
--known 1000G_phase1.indels.b37.vcf
```

The "known sites" to use for this and each subsequent step were chosen from GATK recommendations (see http://gatkforums.broadinstitute.org/discussion/1247/what-should-i-use-as-known-variantssites-for-running-tool-x#latest). The IndelRealigner was run separately for each chromosome using the option –L <chr>. For example, the command for chromosome 1 was:

```
java -jar GenomeAnalysisToolkit.jar -T IndelRealigner -U -R Homo_sapiens_assembly19.fasta
-o H01UJ.12.aligned.wholegenome.sorted.indel_cleaned_local.1.bam
-I H01UJ.12.aligned.wholegenome.sorted.bam
-L 1 -targetIntervals H01UJ.12.aligned.wholegenome.sorted.indel_cleaned_local.intervals
-model USE_SW -maxInMemory 100000000
-known Mills_and_1000G_gold_standard.indels.b37.sites.vcf -known 1000G_phase1.indels.b37.vcf
```

and the resulting files for each chromosome were merged using Picard's MergeSamFiles with the following options:

```
java –jar MergeSamFiles.jar
I=H01UJ.12.aligned.wholegenome.sorted.indel_cleaned_local.1.bam
I=H01UJ.12.aligned.wholegenome.sorted.indel_cleaned_local.2.bam
.
.
```

```
.
I=H01UJ.12.aligned.wholegenome.sorted.indel_cleaned_local.X.bam
O=H01UJ.12.aligned.wholegenome.sorted.indel_cleaned_local.bam
USE_THREADING=true  MAX_RECORDS_IN_RAM=100000000    SORT_ORDER=coordinate ASSUME_SORTED=false
MERGE_SEQUENCE_DICTIONARIES=false VERBOSITY=INFO QUIET=false VALIDATION_STRINGENCY=STRICT
COMPRESSION_LEVEL=5 CREATE_INDEX=false CREATE_MD5_FILE=false
```

Step 4 - Base Quality Score Recalibration (BQSR). The GATK BaseRecalibrator was used to determine a transfer function for quality score recalibration and PrintReads was used to generate a new set of reads with transformed quality scores:

```
java -jar GenomeAnalysisToolkit.jar -T BaseRecalibrator -nct 8 -nt 1
-I H01UJ.12.aligned.wholegenome.sorted.indel_cleaned_local.bam
-o H01UJ.12.aligned.wholegenome.sorted.indel_cleaned_local.recal_data.grp
-R Homo_sapiens_assembly19.fasta
-knownSites Mills_and_1000G_gold_standard.indels.b37.sites.vcf
-knownSites 1000G_phase1.indels.b37.vcf
-knownSites dbsnp_137.b37.vcf
```

```
java -jar GenomeAnalysisToolkit.jar -T PrintReads -nct 8 -R Homo_sapiens_assembly19.fasta
-I H01UJ.12.aligned.wholegenome.sorted.indel_cleaned_local.bam
-BQSR H01UJ.12.aligned.wholegenome.sorted.indel_cleaned_local.recal_data.grp
-o H01UJ.12.aligned.wholegenome.sorted.indel_cleaned_local.recal.bam
```

Step 5 - Variant Calling. The GATK HaplotypeCaller was used for variant calling of both SNPs and indels. Parallel efficiency in this module is such that it is not possible to fully utilize a large number of CPUs using thread parallelism alone. For this module, and several others, the work can be divided along the genome and different instances of the algorithm could process each 'chunk' of work. The resulting output (variant call format (VCF) files) could then be merged into a single VCF file covering the entire genome. This "scatter-gather" parallelism can be implemented in GATK by manually dividing up the region of analysis, however GATK has a facility called Queue that we used to automate this process. We present here an equivalent, thread-parallel version of the command that was used. This command should produce precisely the answers that we generated, but would take considerably longer to complete. Information on GATK Queue can be found at (http://gatkforums.broadinstitute.org/discussion/1306/overview-of-queue).

```
java –jar GenomeAnalysisToolkit.jar –T HaplotypeCaller –I
H01UJ.12.aligned.wholegenome.sorted.indel_cleaned_local.recal.bam –R
Homo_sapiens_assembly19.fasta –nct 4 –pcrModel NONE –o
H01UJ.12.aligned.wholegenome.sorted.indel_cleaned_local.recal.unfiltered.vcf
```

Step 6 - Variant Quality Score Recalibration (VQSR). The GATK VariantRecalibrator was used to generate recalibration data for SNPs:

```
java -jar GenomeAnalysisToolkit.jar -T VariantRecalibrator -R Homo_sapiens_assembly19.fasta
-input H01UJ.12.aligned.wholegenome.sorted.indel_cleaned_local.recal.unfiltered.vcf
-recalFile H01UJ.12.aligned.wholegenome.sorted.indel_cleaned_local.recal.unfiltered.snp.recal
-tranchesFile H01UJ.12.aligned.wholegenome.sorted.indel_cleaned_local.recal.unfiltered.snp.tranches
-rscriptFile H01UJ.12.aligned.wholegenome.sorted.indel_cleaned_local.recal.unfiltered.snp.plots.R
--numBadVariants [see table] --maxGaussians [see table]
-resource:hapmap,known=false,training=true,truth=true,prior=15.0 hapmap_3.3.b37.vcf
-resource:omni,known=false,training=true,truth=true,prior=12.0 1000G_omni2.5.b37.vcf
-resource:1000G,known=false,training=true,truth=false,prior=10.0
1000G_phase1.snps.high_confidence.b37.vcf
-resource:dbsnp,known=true,training=false,truth=false,prior=2.0 dbsnp_137.b37.vcf
-nt 22 -an QD -an MQRankSum -an ReadPosRankSum -an FS -an MQ -an DP -mode SNP
```

and for INDEL calls:

```
java -jar GenomeAnalysisToolkit.jar -T VariantRecalibrator -R Homo_sapiens_assembly19.fasta
-input H01UJ.12.aligned.wholegenome.sorted.indel_cleaned_local.recal.unfiltered.vcf
-recalFile H01UJ.12.aligned.wholegenome.sorted.indel_cleaned_local.recal.unfiltered.indel.recal
-tranchesFile
H01UJ.12.aligned.wholegenome.sorted.indel_cleaned_local.recal.unfiltered.indel.tranches
-rscriptFile H01UJ.12.aligned.wholegenome.sorted.indel_cleaned_local.recal.unfiltered.indel.plots.R
--maxGaussians [see table] --numBadVariants [see table]
-resource:mills,VCF,known=false,training=true,truth=true,prior=12.0
-resource:dbsnp,VCF,known=true,training=false,truth=false,prior=2.0 dbsnp_137.b37.vcf
Mills_and_1000G_gold_standard.indels.b37.sites.vcf
-nt 22 -an DP -an MQRankSum -an FS -an ReadPosRankSum -mode INDEL
```

Note that the precise value of the --numBadVariants argument varied from 1000 to 3000 to 10000 depending on when the model converged. The numbers shown were used for NA12878.

Finally, ApplyRecalibration was run first for SNPs. Scatter-gather parallelization via Queue (see step 5) was used to distribute the work to multiple computers, however an equivalent thread-parallel command line is:

```
java -jar GenomeAnalysisToolkit.jar -T ApplyRecalibration -R Homo_sapiens_assembly19.fasta -nt 12
-input H01UJ.12.aligned.wholegenome.sorted.indel_cleaned_local.recal.unfiltered.vcf
-recalFile H01UJ.12.aligned.wholegenome.sorted.indel_cleaned_local.recal.unfiltered.snp.recal
-tranchesFile H01UJ.12.aligned.wholegenome.sorted.indel_cleaned_local.recal.unfiltered.snp.tranches
-o H01UJ.12.aligned.wholegenome.sorted.indel_cleaned_local.recal.unfiltered.recal_snp_raw_indel.vcf
-ts_filter_level [see table] -mode SNP
```

and similarly for INDELs:

```
java -jar GenomeAnalysisToolkit.jar -T ApplyRecalibration -R Homo_sapiens_assembly19.fasta -nt 12
-input
H01UJ.12.aligned.wholegenome.sorted.indel_cleaned_local.recal.unfiltered.recal_snp_raw_indel.vcf
-recalFile H01UJ.12.aligned.wholegenome.sorted.indel_cleaned_local.recal.unfiltered.indel.recal
-tranchesFile
H01UJ.12.aligned.wholegenome.sorted.indel_cleaned_local.recal.unfiltered.indel.tranches
-o H01UJ.12.aligned.wholegenome.sorted.indel_cleaned_local.recal.unfiltered.recal_sn
p_recal_indel.vcf
-ts_filter_level [see table] -mode INDEL
```

### *7d. Generation of variants calls from 250 base reads using Cortex*

0. Prepare reference data for Cortex's calling pipeline.

```
${cortex_loc}/bin/cortex_var_31_c1 --se_list list_fa --kmer_size 31 --mem_height 25 --mem_width 120
--sample_id HG19 --dump_binary hg19.k31.ctx

${cortex_loc}/bin/cortex_var_63_c1 --se_list list_fa --kmer_size 61 --mem_height 25 --mem_width 120
--sample_id HG19 --dump_binary hg19.k61.ctx

stampy.py -G HG19 <HG19 reference sequence path>

stampy-1.0.22/stampy.py -g HG19 -H HG19
```

where list_fa is a file containing a single line consisting of the path of the HG19 reference sequence.

1. Create fastq files from bam files using Picard tools v1.584.

```
java -jar SortSam.jar I=bam O=sorted.bam SO=queryname

java -jar SamToFastq.jar I=sorted.bam F=R1.fastq F2=R2.fastq CLIP_ATTR=XT CLIP_ACT=X
INCLUDE_NON_PF_READS=true TMP_DIR=tmp
```

1. Build a k=31 graph using cortex v1.0.5.19.

```
perl ${cortex_loc}/scripts/calling/run_calls.pl --first_kmer 31 --mem_height 26 --mem_width 100
--auto_clean yes --fastaq_index INDEX_UNCORR --outdir OUTDIR_UNCORR --qthresh 20 --do_union no
--workflow joint --logfile step1.log --ploidy 2 --ref Absent --vcftools_dir ${vcf_loc}
--genome_size 3000000000
```

2. Perform read correction using the k=31 graph generated in step 2.

```
${cortex_loc}/bin/cortex_var_31_c1 --multicolour_bin
OUTDIR_UNCORR/binaries/cleaned/k31/NA12878.kmer31.q20cleaned_3.ctx --mem_height 27 --mem_width 100
--max_read_len 300 --quality_score_threshold 20 --err_correct
list_uncorrected_fastq.s3,corrected.q20,corrected_reads_outdir,1,0 --kmer_size 31
```

3. Use gzip v1.3.12 to compress the corrected fastq files.

```
find `pwd`/corrected_reads_outdir -name '*gz' > list_corrected

printf 'NA12891\tlist_corrected\t.\t.\n' > INDEX_CORRECTED
```

4. Invoke K=31 and K=61 "bubble caller" pipeline concurrently.

```
perl ${cortex_loc}/scripts/calling/run_calls.pl --first_kmer 31 --fastaq_index INDEX_CORRECTED
--auto_clean yes --bc yes --pd no --outdir FINAL_RESULTS --ploidy 2 --stampy_hash ${ref_dir}/HG19
--stampy_bin ${stampy_loc}/stampy.py --list_ref_fasta ${ref_dir}/list_c_fa --refbindir ${ref_dir}
--genome_size 3000000000 --qthresh 10 --mem_height 26 --mem_width 100 --do_union no --ref
CoordinatesAndInCalling --workflow independent --logfile logfile.bc31 BC_log.txt.bc31
--vcftools_dir ${vcf_loc}

perl ${cortex_loc}/scripts/calling/run_calls.pl --first_kmer 61 --fastaq_index INDEX_CORRECTED
--auto_clean yes --bc yes --pd no --outdir FINAL_RESULTS --ploidy 2 --stampy_hash ${ref_dir}/HG19
--stampy_bin ${stampy_loc}/stampy.py --list_ref_fasta ${ref_dir}/list_c_fa --refbindir ${ref_dir}
--genome_size 3000000000 --qthresh 10 --mem_height 26 --mem_width 100 --do_union no --ref
CoordinatesAndInCalling --workflow independent --logfile logfile.bc61 BC_log.txt.bc61
--vcftools_dir ${vcf_loc}
```

5. After the cleaned binaries have been generated during step 5, invoke the K=31 "path divergence caller".

```
perl ${cortex_loc}/scripts/calling/run_calls.pl --first_kmer 31 --fastaq_index INDEX_CORRECTED
--auto_clean yes --bc no --pd yes --outdir FINAL_RESULTS --ploidy 2 --stampy_hash ${ref_dir}/HG19
--stampy_bin ${stampy_loc}/stampy.py --list_ref_fasta ${ref_dir}/list_c_fa --refbindir ${ref_dir}
--genome_size 3000000000 --qthresh 10 --mem_height 27 --mem_width 100 --do_union yes --ref
CoordinatesAndInCalling --workflow independent --logfile logfile.pd31 BC_log.txt.pd31
--vcftools_dir ${vcf_loc} --max_var_len 40000
```

6. After step 5 has finished, invoke the union of K=31 and K=61 calls with the v1.0.5.20 code.

```
perl ${cortex_loc}/scripts/calling/run_calls.pl --first_kmer 31 --last_kmer 61 --kmer_step 30
--fastaq_index INDEX_CORRECTED --auto_clean yes --bc yes --pd no --outdir FINAL_RESULTS --ploidy 2
--stampy_hash ${ref_dir}/HG19 --stampy_bin ${stampy_loc}/stampy.py --list_ref_fasta
${ref_dir}/list_c_fa --refbindir ${ref_dir} --genome_size 3000000000 --qthresh 10 --mem_height 26
--mem_width 100 --do_union yes --ref CoordinatesAndInCalling --workflow independent --logfile
logfile.bc31 BC_log.txt.bc31 --vcftools_dir ${vcf_loc}
```

7. Merge the VCF files generated by cortex's BC and PD workflow with VCF tools v0.1.9.

```
vcf-concat
FINAL_RESULTS/vcfs/default_vcfname_wk_flow_I_RefCC_FINALcombined_BC_calls_at_all_k.decomp.vcf
FINAL_RESULTS/vcfs/default_vcfname_wk_flow_I_RefCC_FINALcombined_PD_calls_at_all_k.decomp.vcf |
vcf-sort -c > bc3161pd31.sorted.vcf
```

8. Discard calls without a PASS in the VCF FILTER column. Group calls which overlap with one another, and remove all PD calls from each of those groups.

```
python rm-overlapping-pd-calls.py bc3161pd31.sorted.vcf > bc3161pd31.sorted.rm_overlapped_pd.vcf
```

9. Invoke cortex v1.0.5.20's mechanism to resolve overlap remaining overlapping calls.

```
cat bc3161pd31.sorted.rm_overlapped_pd.vcf | ${cortex_loc}/scripts/analyse_variants/bioinf-
perl/vcf_scripts/vcf_remove_overlaps.pl --filter_txt OVERLAPPING_SITE | ${vcf_loc}/perl/vcf-sort -c
| ${cortex_loc}/scripts/analyse_variants/bioinf-perl/vcf_scripts/vcf_revert_consistent_overlaps.pl
> FINAL.vcf
```

### *7e. Generation of variant calls from Fosmid reference sequences*

To generate Fosmid reference variant calls we aligned the Fosmid reference sequences to the hg19 reference sequence using an affine Smith-Waterman alignment algorithm, with mismatch penalty 3, gap open penalty 10 and gap extension penalty 1. The Fosmids were aligned to the regions defined in **Supplementary Table 9**. Three Fosmids could not be confidently aligned to a single location and were thus excluded from the analysis. To document this, for each case we show the coordinates of the 'best' and 'second best' placements, and the number of reference disagreements (substitutions, or indels of one or more bases), as computed by an affine Smith-Waterman alignment as above.

| **Fosmid id** | **placement** | **substitutions** | **indels** |
|---|---|---|---|
| 0 | chr1:405,014-436,839 | 2 | 4 |
| | chr5:180,831,564-180,863,393 | 3 | 3 |
| 35 | chr15:21,334,868-21,375,963 | 156 | 36 |
| | chr15:20,328,256-20,369,354 | 160 | 35 |
| 67 | chr2:131,216,319-131,248,821 | 147 | 13 |
| | chr2:131,387,014-131,419,519 | 168 | 16 |

We also considered the possibility that for some Fosmid there might be two or more similar copies in the NA12878 genome, but only one in the hg19 reference. To test for this, we computed the mean coverage in each case (**Supplementary Table 10**), finding a mean value of 53.4x, and with all 100 values within 20% of this, with 2 exceptions, Fosmid 106, at 21% below the mean, and Fosmid 40, at 52% above the mean. We thus deemed only Fosmid 40 as worthy of further examination for the purposes of testing whether the region appears multiply in the NA12878 genome.

We then examined Fosmid 40 in detail. To begin the analysis, we computed some general coverage statistics. For each of 10,000 randomly selected 21-mers that are unique in the hg19 reference sequence, we computed the coverage in the combined NA12878 data set (from both this work and the 1000 Genomes Project), *i.e.* the number of times that the 21-mer occurs in the reads and their reverse complements. The mean value of this coverage is 90.5x. We then computed the relative value ('fraction of mean coverage') achieved by each of the 10,000 21-mers, finding the following:

| **Fraction of mean coverage** | **% of 21-mers** |
|---|---|
| < 0.5 | 2.40 |
| 0.5-0.6 | 1.70 |
| 0.6-0.7 | 2.93 |
| 0.7-0.8 | 5.89 |
| 0.8-0.9 | 9.99 |
| 0.9-1.0 | 16.76 |
| 1.0-1.1 | 21.47 |
| 1.1-1.2 | 18.20 |
| 1.2-1.3 | 11.61 |
| 1.3-1.4 | 4.62 |

| | |
|---|---|
| 1.4-1.5 | 1.96 |
| > 1.5 | 2.47 |

We note that 66.42% of 21-mers occur between 0.8 and 1.2 of the mean, and that only 4.87% of 21-mers occur at either < 0.5 or > 1.5 of the mean. Thus the multiplicity of a 21-mer in the reads provides a good indicator of the actual multiplicity in the genome.

As a second control, we computed the coverage by the Fosmid pool reads of 21-mers that occur exactly once in Fosmid 40. We normalized these coverage values so that the mean was 1. We would thus expect that 21-mers occurring in particularly difficult sequence contexts would have lower normalized coverage values.

We next looked for loci in the Fosmid 40 reference sequence that appear to occur exactly once in the genome (meaning, once on each of two homologous chromosomes). To do this we identified 21-mers whose coverage (for the combined aligned and unaligned whole-genome NA12878 data set) is close to the expected value of 90.5x, as described above, and for each of these we checked the normalized Fosmid pool coverage. We considered only 21-mers that appear in a non-bubble edge in an assembly graph: meaning an edge where it appears topologically that both chromosomes come together. Moreover we carried out this test in a combined DISCOVAR assembly of NA12878 and her parents, including both the data of this work and the 1000 Genomes Project data sets. By increasing coverage in this way we reduced the likelihood that another branch of the graph was accidentally missed. Here we note some examples:

| **21-mer** | **position on Fosmid** | **WGS coverage, as fraction of expected value** | **normalized Fosmid pool coverage** |
|---|---|---|---|
| `AAAATTGTTTTAATTACCTGA` | 20135 | 0.983 | 0.983 |
| `ACACATGCTGTGTTCTTACCA` | 26020 | 1.071 | 0.843 |
| `ACTTTTACTAGAGACAGGAAT` | 26904 | 0.806 | 0.868 |
| `AGCCACAGTCTGAACTCTAAC` | 32792 | 0.880 | 0.863 |

These examples suggest that if there is another mutated copy of Fosmid 40 somewhere in the genome, then it is sufficiently different from the hg19 reference sequence (at the aligned location of Fosmid 40, at the given 21-mers) that the reads from it are not aligned there. By contrast, the reference sequence for Fosmid 40 can be exactly threaded through the above assembly graph, with the exception of a single homopolymer length discrepancy. These data suggest that Fosmid 40 (and the homologous region) are the regions in the NA12878 genome that are most similar to the hg19 region where they are aligned.

## 8. Data cost estimate

Estimated costs are shown below for HiSeq 2500 data (250 or 100 base reads), and HiSeq 2000 (100 base reads), exclusive of overhead, and labor costs, which vary. Cost per PF Gb summarizes the overall cost of each data type, with these provisos. Library costs are not included as these will be the same irrespective of read length and instrument type. We note that per base costs do not fully reflect data utility, as longer reads have lower mean quality. For the datasets used in this work, 92% of bases are Q30 or better in 101-base reads, but only 65% in 250-base reads. However in a dataset shared by Illumina very recently (not shown), and based on newer chemistry, 79% of bases are Q30.

| **HiSeq 2500 2x250 base reads** | | **HiSeq 2500 2x100 base reads** | | **HiSeq 2000 2x100 base reads** | |
|---|---|---|---|---|---|
| **Flowcell reagent costs (2 lanes)** | | **Flowcell reagent costs (2 lanes)** | | **Flowcell reagent costs (8 lanes)** | |
| **Reagents** | **List price** | **Reagents** | **List price** | **Reagents** | **List price** |
| 1x cluster kit | $1,275 | 1x cluster kit | $1,275 | 1x cluster kit | $5,540 |
| 2.5x 200 cyc seq kits | $4,400 | 1x 200 cyc seq kit | $1,760 | 1x 200 cyc seq kit | $7,645 |
| **Total** | **$5,675** | **Total** | **$3,035** | **Total** | **$13,185** |
| **Machine depreciation costs** | | **Machine depreciation costs** | | **Machine depreciation costs** | |
| List price | $750,000 | List price | $750,000 | List price | $700,000 |
| ~Runs/year | 122 | ~Runs/year | 209 | ~Runs/year | 30 |
| ~Flowcells/year | 243 | ~Flowcells/year | 417 | ~Flowcells/year | 61 |
| Service downtime | 10% | Service downtime | 10% | Service downtime | 10% |
| Failure Rate | 10% | Failure Rate | 10% | Failure Rate | 10% |
| Actual flowcells/year | 195 | Actual flowcells/year | 334 | Actual flowcells/year | 49 |
| Depreciation/year (3 years) | $250,000 | Depreciation/year (3 years) | $250,000 | Depreciation/year (3 years) | $233,333 |
| **Depreciation/Flowcell** | **$1,284** | **Depreciation/Flowcell** | **$749** | **Depreciation/Flowcell** | **$4,795** |
| **Total costs per flowcell (2 lanes) (reagents and depreciation)** | | **Total costs per flowcell (2 lanes) (reagents and depreciation)** | | **Total costs per flowcell (8 lanes) (reagents and depreciation)** | |
| Reagents | $5,675 | Reagents | $3,035 | Reagents | $13,185 |
| Depreciation | $1,284 | Depreciation | $749 | Depreciation | $4,795 |
| **Total** | **$6,959** | **Total** | **$3,784** | **Total** | **$17,980** |
| Yield PF Gb/flowcell | 120 | Yield PF Gb/flowcell | 51 | Yield PF Gb/flowcell | 300 |
| assumes 150M reads per lane, 80% PF i.e. (150M x 500 bases) x 80% = 60Gb per lane | | assumes 150M reads per lane, 85% PF i.e. (150M x 200 bases) x 85% = 25.5Gb per lane | | assumes 210M reads per lane, 90% PF i.e. (210M x 200 bases) x 90% = 37.5Gb per lane | |
| **Cost per PF Gb** | **$58** | **Cost per PF Gb** | **$74** | **Cost per PF Gb** | **$60** |
| **Cost per lane** | **$3,480** | **Cost per lane** | **$1,892** | **Cost per lane** | **$2,247** |
| **Gb per lane** | **61** | **Gb per lane** | **25.5** | **Gb per lane** | **37.5** |

## 9. Analysis of two variant clusters

We examined the two large variant clusters described in the text, consisting of 49 variants for Fosmid 62 and 21 variants for Fosmid 83. First we identified the source clones used in assembling these sequences (GenBank AC010969.11, AL354682.22), and noted that no problems were flagged in the cluster regions. We next sought to examine the original data used to assemble the clones. The reads for the second clone were available but those for the first were not. The cluster region for the second clone was covered by two reads, in the same orientation, each having a mismatch with the reference sequence. Therefore, particularly in the first case, it was not possible for us to confirm that the hg19 reference sequence was exactly correct at the two clusters. Interestingly, we note the Fosmid reference seqiuence for the first cluster closely matches a Sanger chemistry read in the NCBI Trace Archive. Indeed it matches read 1742047327 perfectly except for a single substitution and four insertions of sizes 6, 4, 2 and 2.

## 10. Supplementary references

34. Fisher S. *et al*. A scalable, fully automated process for construction of sequence-ready human exome targeted capture libraries. *Genome Biol.* **12**, R1 (2011).

35. Williams L.J. *et al*. Paired-end sequencing of Fosmid libraries by Illumina. *Genome Re*s. **22**, 2241-2249 (2012).

36. Ribeiro F.J. *et al*. Finished bacterial genomes from shotgun sequence data. *Genome Res.* **22**, 2270-2277 (2012).

37. Li H., Durbin R. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics* **25**, 1754-1760 (2009).