cgatools Methods

Software v1.4.0

Table of Contents

Preface

The Complete Genomics Analysis Tools (**cgatools**) is an open source project to provide tools for downstream analysis of Complete Genomics data. This document describes the motivation and design decisions for **cgatools**.

Conventions

This document uses the following notational conventions:

Notation	Description
italic	A field name from a data file. For example, the *varType* field in the variations data file indicates the type of variation identified between the assembled genome and the reference genome. Also used to indicate the values found in data files. For example, *ChromosomeName* indicates that the value found in the data file is the name of a given chromosome.
bold_italic	A file name from the data package. For example, each package contains the file ***manifest.all***.

CGI Data

Complete Genomics, Inc. (CGI) delivers complete genome sequencing data to customers and collaborators. The data include sequence reads, their mappings to a reference human genome, and variations detected against the reference human genome. This document describes tools developed to analyze this data.

References

- Assembly Pipeline Release Notes — Indicates new features and enhancements by release.
- *Complete Genomics Variation FAQ* — Answers to frequently asked questions about Complete Genomics variation data.
- *Complete Genomics Technology Whitepaper* — A concise description of the Complete Genomics sequencing technology, including the library construction process and the ligation-based assay approach. It is available in the "Resources" section of the Complete Genomics website. [www.completegenomics.com]
- *Getting Started with CGI's Data FAQ* — Answers to questions about preparing to receive the hard drives of data.
- Complete Genomics *Data File Formats* — A description of the organization and content of the format for complete genome sequencing data delivered by Complete Genomics. It is available on the Complete Genomics website. [www.completegenomics.com]
- Complete Genomics *Science* Article — An article from the Complete Genomics chief scientific officer describing the process of how CGI maps reads (*Science* 327 (5961), 78. [DOI: 10.1126/science.1181498]). We also recommend you read the *Complete Genomics Service FAQ* as background for this document. You can access the paper at www.drmanac.com at no charge.
- SAM — The Sequence Alignment/Map format is a generic format for storing large nucleotide sequence alignments. This **cgatools** description is based on the SAM Format Specification "Sequence Alignment/Map (SAM) Format," Version 0.1.2-draft (August 20, 2009). [http://samtools.sourceforge.net/SAM1.pdf]
- NCBI RefSeq alignment data — **R**eference assembly and alignment data. [ftp://ftp.ncbi.nih.gov/genomes/H_sapiens/mapview/seq_gene.md.gz]

Reference Tools

At times, **cgatools** uses the reference sequence in a random-access manner. The most common reference sequence format, FASTA, is not ideal for processing tasks that require random access because the entire sequence must be read into memory at the start of the program, and this memory cannot be shared among processes.

CRR File Format

cgatools uses its own file format, Compact Randomly Accessible Reference (CRR), to represent a reference sequence. The CRR file format stores two bits per base of reference, plus lookup tables to resolve regions of the reference that are represented by ambiguous IUPAC codes. CRR files are memory mapped, so that processes can share a reference, and the overall memory requirement due to the reference for all processes is less than 1 GB. The CRR file format does not preserve character case—FASTA reference files often use case to denote the region's repeat status—and considers all the bases described in the reference FASTA sequence as upper case.

FASTA Reference Sequences

Complete Genomics supports two references. The first, which we refer to as "build 36," consists of the assembled nuclear chromosomes from NCBI build 36 (not unplaced or alternate loci) plus Yoruban mitochondrion NC_001807.4. This assembly is also known as UCSC hg18. The second reference, which we refer to as "build 37," consists of the assembled nuclear chromosomes from GRCh37 (not unplaced or alternate loci), plus the Cambridge Reference Sequence for the mitochondrion (NC_012920.1). This assembly (though with an alternate mitochondrial sequence) is also known as UCSC hg19. The resulting FASTA sequences are available here:

ftp://ftp.completegenomics.com/ReferenceFiles/build36.fa.bz2

ftp://ftp.completegenomics.com/ReferenceFiles/build37.fa.bz2

The CRR files can also be directly downloaded and are available here:

ftp://ftp.completegenomics.com/ReferenceFiles/build36.crr

ftp://ftp.completegenomics.com/ReferenceFiles/build37.crr

fasta2ccr

This tool converts FASTA sequences into a single reference CRR file. The result is the same as what is available on the FTP site.

crr2fasta

This tool converts CRR sequence files to the FASTA file format.

decodecrr

This command retrieves the sequence for a given range of a chromosome.

listcrr

This command lists the chromosomes, contigs, or regions of ambiguous sequence within the reference, depending on the parameters. The contigs described by listcrr are defined to be the contiguous sequence bases separated by at least `min-contig-gap-length`, no-call bases, where `min-contig-gap-length` defaults to 50. The default contigs correspond to the notion of contig employed in the Complete Genomics data, such as reference scores.

After you have successfully downloaded a build 36 or 37 CRR file (that is, converted the downloaded reference into CRR using fasta2crr) for use with Complete Genomics data, the listcrr command returns the output shown in **Figure 1**:

Figure 1: listcrr Output for Build 36

```
ChromosomeId Chromosome    Length Circular Md5
           0       chr1 247249719    false 9ebc6df9496613f373e73396d5b3b6b6
           1       chr2 242951149    false b12c7373e3882120332983be99aeb18d
           2       chr3 199501827    false 0e48ed7f305877f66e6fd4addbae2b9a
           3       chr4 191273063    false cf37020337904229dca8401907b626c2
           4       chr5 180857866    false 031c851664e31b2c17337fd6f9004858
           5       chr6 170899992    false bfe8005c536131276d448ead33f1b583
           6       chr7 158821424    false 74239c5ceee3b28f0038123d958114cb
           7       chr8 146274826    false 1eb00fe1ce26ce6701d2cd75c35b5ccb
           8       chr9 140273252    false ea244473e525dde0393d353ef94f974b
           9      chr10 135374737    false 4ca41bf2d7d33578d2cd7ee9411e1533
          10      chr11 134452384    false 425ba5eb6c95b60bafbf2874493a56c3
          11      chr12 132349534    false d17d70060c56b4578fa570117bf19716
          12      chr13 114142980    false c4f3084a20380a373bbbdb9ae30da587
          13      chr14 106368585    false c1ff5d44683831e9c7c1db23f93fbb45
          14      chr15 100338915    false 5cd9622c459fe0a276b27f6ac06116d8
          15      chr16  88827254    false 3e81884229e8dc6b7f258169ec8da246
          16      chr17  78774742    false 2a5c95ed99c5298bb107f313c7044588
          17      chr18  76117153    false 3d11df432bcdc1407835d5ef2ce62634
          18      chr19  63811651    false 2f1a59077cfad51df907ac25723bff28
          19      chr20  62435964    false f126cdf8a6e0c7f379d618ff66beb2da
          20      chr21  46944323    false f1b74b7f9f4cdbaeb6832ee86cb426c6
          21      chr22  49691432    false 2041e6a0c914b48dd537922cca63acb8
          22       chrX 154913754    false d7e626c80ad172a4d7c95aadb94d9040
          23       chrY  57772954    false 62f69d0e82a12af74bad85e2e4a8bd91
          24       chrM     16571     true d2ed829b8a1628d16cbeee88e88e39eb
```

After you have successfully downloaded a build 37 CRR file (or converted the downloaded reference into CRR using fasta2crr) for use with Complete Genomics data, the listcrr command returns the output shown in Figure 2:

Figure 2: listcrr Output for Build 37

```
ChromosomeId Chromosome    Length Circular Md5
           0       chr1 249250621    false 1b22b98cdeb4a9304cb5d48026a85128
           1       chr2 243199373    false a0d9851da00400dec1098a9255ac712e
           2       chr3 198022430    false 641e4338fa8d52a5b781bd2a2c08d3c3
           3       chr4 191154276    false 23dccd106897542ad87d2765d28a19a1
           4       chr5 180915260    false 0740173db9ffd264d728f32784845cd7
           5       chr6 171115067    false 1d3a93a248d92a729ee764823acbbc6b
           6       chr7 159138663    false 618366e953d6aaad97dbe4777c29375e
           7       chr8 146364022    false 96f514a9929e410c6651697bded59aec
           8       chr9 141213431    false 3e273117f15e0a400f01055d9f393768
           9      chr10 135534747    false 988c28e000e84c26d552359af1ea2e1d
          10      chr11 135006516    false 98c59049a2df285c76ffb1c6db8f8b96
          11      chr12 133851895    false 51851ac0e1a115847ad36449b0015864
          12      chr13 115169878    false 283f8d7892baa81b510a015719ca7b0b
          13      chr14 107349540    false 98f3cae32b2a2e9524bc19813927542e
          14      chr15 102531392    false e5645a794a8238215b2cd77acb95a078
          15      chr16  90354753    false fc9b1a7b42b97a864f56b348b06095e6
          16      chr17  81195210    false 351f64d4f4f9ddd45b35336ad97aa6de
          17      chr18  78077248    false b15d4b2d29dde9d3e4f93d1d0f2cbc9c
          18      chr19  59128983    false 1aacd71f30db8e561810913e0b72636d
          19      chr20  63025520    false 0dec9660ec1efaaf33281c0d5ea2560f
          20      chr21  48129895    false 2979a6085bfe28e3ad6f552f361ed74d
          21      chr22  51304566    false a718acaa6135fdca8357d5bfe94211dd
          22       chrX 155270560    false 7e0e2e580297b7764e31dbc80c2540dd
          23       chrY  59373566    false 1e86411d73e6f00a10590f976be01623
          24       chrM     16569     true c68f52674c9fb33aef52dcf399755519
```

Genome Comparison Tools

A Note on Conventions

To call low certainty regions or "no-call" regions, Complete Genomics augments the alphabet {A, C, G, T} with two additional characters:

- The "N" character corresponds to a one-base sequence that may be any of {A, C, G, T}.
- The "?" character corresponds to zero or more bases of unknown sequence.

The Problem of Genome Comparison

Genome comparison is the problem of identifying genomic sequence that is identical, compatible (perhaps with no-calls), or incompatible, with sequence from another genome. Within the space of genome comparison problems, there are three common tasks:

1. Is a genome identical, compatible, or incompatible with the reference genome at a given location?
2. Is a genome identical, compatible, or incompatible with a known common sequence?
3. Is a genome identical, compatible, or incompatible with a particular genome at a given location within the reference genome?

The particular way a genome is described by re-sequencing technologies goes a long way towards solving genome comparison problems 1 and 2: genomes are represented as a set of differences (or variants) against the reference genome. The Complete Genomics variant file format differs from most other common variant file formats in that in addition to describing the variants, it also distinguishes regions of the genome that are called as reference from those that are no-called. As we will see later, this distinction is essential in solving many comparison problems.

The following example shows how the Complete Genomics variations file describes the situation where chr1 is a diploid chromosome and chr2 is a haploid chromosome:

chr1 reference:	CATGACCCGCAAA-TCTGAAACTATCTGGCCCTTGGCAGGGG--A
chr1 haplotype 1:	?ATGACCTGCAAAATCTGAAACT--CTGGCCCTTGGCAGGGGGGA
chr1 haplotype 2:	?ATGACCCGCAAAATCTGAAACTATCTGGCTNTTGGCAGGGT--A
chr2 reference:	TGATATTTTTCATCAACATTACAGGCA
chr2:	TGATATTTTTNATCAACACGACAGGCA

Figure 3 shows the corresponding variant file.

Figure 3: Variant File

>locus	ploidy	allele	chromosome	begin	end	varType	reference	alleleSeq	totalScore	hapLink	xRef
1	2	all	chr1	0	1	no-call	=	?			
2	2	all	chr1	1	7	ref	=	=			
3	2	1	chr1	7	8	snp	C	T	87	1	dbsnp:123
3	2	2	chr1	7	8	ref	C	C	58	2	dbsnp:123
4	2	all	chr1	8	13	ref	=	=			
5	2	1	chr1	13	13	ins		A	36		
5	2	2	chr1	13	13	ins		A	42		
6	2	all	chr1	13	22	ref	=	=			
7	2	1	chr1	22	24	del	AT		47	1	
7	2	2	chr1	22	24	ref	AT	AT	55	2	
8	2	all	chr1	24	29	ref	=	=			
9	2	1	chr1	29	31	ref	CC	CC	57	1	
9	2	2	chr1	29	31	no-call-ri	CC	TN	65	2	
10	2	all	chr1	31	40	ref	=	=			
11	2	1	chr1	40	41	ref	G	G	101	1	
11	2	1	chr1	41	41	ins		GG	120	1	
11	2	2	chr1	40	41	snp	G	T	479	2	
12	2	all	chr1	41	42	ref	=	=			
13	1	all	chr2	0	10	ref	=	=			
14	1	1	chr2	10	11	no-call-rc	C	N	47		
15	1	all	chr2	11	18	ref	=	=			
16	1	1	chr2	18	20	sub	TT	CG	102		
17	1	all	chr2	20	27	ref	=	=			

The genome is first aligned to the reference, and then split into loci. Each locus may describe multiple alleles (if ploidy > 1), and for each allele at each locus, there may be one or more lines (or "calls") to describe the sequence. The variant file describes 0-based offsets within the reference chromosome.

In the variant file in Figure 3, locus 3 describes a heterozygous SNP (one-base polymorphism on one allele, reference on the other allele). Locus 5 describes a homozygous insertion (in which the confidence is slightly higher for allele 2 than allele 1). The allele column is used to distinguish the alleles of calls within a locus. For example, the "ref" and "ins" calls of locus 11 are on the same allele, whereas the "snp" call is on the opposite allele. To declare that two calls of different loci are on the same haplotype, the format uses the *hapLink* field. Calls known to be on the same haplotype have the same *hapLink* value; calls with different *hapLink* values may or may not be on the same haplotype (the phasing is no-called).

For a detailed reference of the Complete Genomics variant file format, see the *Data File Format* document provided with your genome (See "References").

Problems Not Solved by Variant File Format

One problem you may have noticed is that the problem of aligning a genome to the reference is not necessarily well-defined. For example in Figure 3, the homozygous insertion at locus 5 could have also been described by the same homozygous insertion three bases to the left. Or the substitution at locus 16 could have been described as two SNPs. Comparing two genomes that describe the same sequence in different ways can be complicated.

We could make canonicalization rules such as "always use the rightmost insertion for any insertion that has multiple possible representations" or "always decompose an allele consisting of a SNP, two reference bases, then another SNP, into separate calls." Indeed, Complete Genomics has rules like these that are

generally followed. But there are at least three remaining problems in solving the genome comparison problems described above:

- Known variants are not always described in their canonical form.

  For example, entries rs34330821 and rs34544546 in the dbSNP database of known variants describe equivalent insertions that are 18 bases apart. This may seem superficial, in that dbSNP entries that are not described in their canonical form can be canonicalized. But if our canonical form uses less decomposition than the dbSNP submission, this may not be possible; if a dbSNP submission has been decomposed, the submission has lost information about nearby variants that exist on the same haplotype.

- Canonical forms of near-identical sequences are not necessarily near-identical.

  For example, suppose we have a genome that is equivalent to a SNP and an insert against the reference, as described in canonicalization 1:

  Reference: `TG A TGTGAATTGGTG ---------------------------- AGT`

  Canonicalization 1: `TG C TGTGAATTGGTG TAGTGTGAATGAGTGTGTGAATTGGTG AGT`

  Reference: `TG A--------------------------- TGTGAATTGGTGAGT`

  Canonicalization 2: `TG CTGTGAATTGGTGTAGTGTGAATGAGTG TGTGAATTGGTGAGT`

  The insert in canonicalization 1 might be the simplest way to describe the genome if the SNP did not exist. But one could argue that the single substitution in canonicalization 2 is the simplest canonicalization of the genome, given that the SNP does exist. (This would be the case for a canonicalization which favors fewer calls.) It is not obvious by visual inspection that the insert from canonicalization 1 and the substitution of canonicalization 2 differ by only a SNP.

- No-calls may not be canonicalized like insertions or deletions, such that an insert may be compatible with another genome only when viewing a larger sequence of the genome.

  For example, suppose we have the following reference and the following genome:

  Reference: `CGAAAAAAA-TTTTCG`

  Genome: `CGAAAAAAAATTTTCG`

  Now suppose the genome reconstruction process discovers that an insertion has occurred, but it does not know if the first base in the run of A's is really an A, or perhaps was a C. In this case, we are forced to align the no-call at the beginning as follows:

  Reference: `CG-AAAAAAATTTTCG`

  Genome: `CGNAAAAAAATTTTCG`

  Length no-calls ("?") may further complicate the situation so that the alignment is unclear. For example, suppose in the same example above, in addition to not knowing if the first base of the run is an A or a C, we also don't know the length of the run of A's at all. Suppose also that we know that the run of T's has increased in length from four to five. There could be at least two reasonable alignments of the result, corresponding to a called insert or a called SNP:

  Reference: `CGAAAAAAATTTT-CG`

  Alignment 1: `CG?AAAAAATTTTTCG`

  Reference: `CG-AAAAAAATTTTCG`

  Alignment 2: `CG?AAAAAATTTTTCG`

Genome Comparison with cgatools

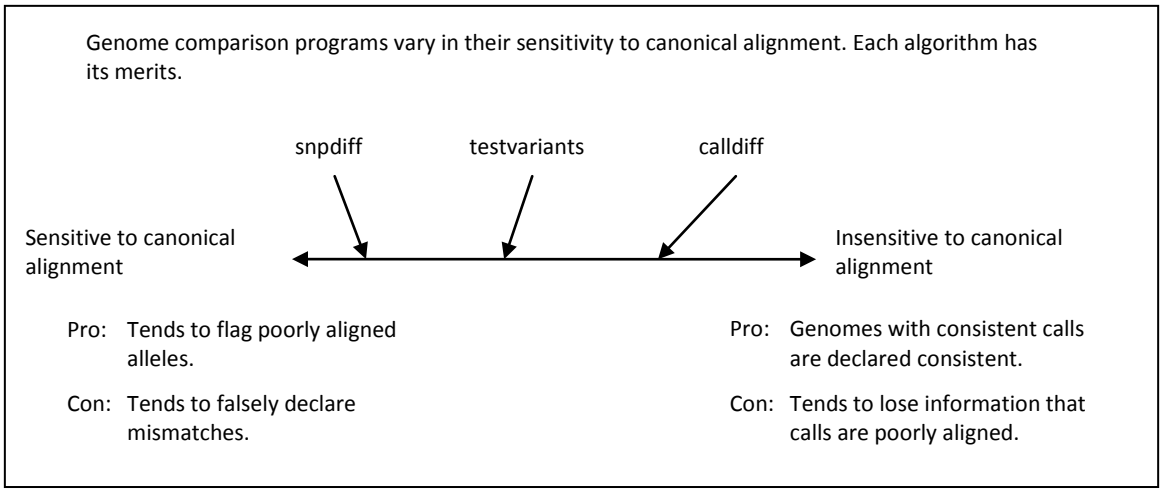
There is a wide spectrum of useful genome comparison methods, which range in their sensitivity to the canonical alignment of called sequence. Algorithms that are very sensitive to canonical alignment tend to declare sequences inconsistent when in fact they are consistent. Algorithms that are less sensitive to canonical alignment tend to be less discriminating in terms of the quality of the alignment of called sequence.

cgatools includes two genome comparison utilities that provide varying degrees of sensitivity to inconsistent canonical alignments:

- snpdiff can be used to compare the results of a SNP caller to a Complete Genomics variant file. It is quite sensitive to the canonical alignment of called sequence.
- calldiff can be used to compare two variant files. It is less sensitive to the canonical alignment of called sequence.

Figure 4 illustrates the tradeoffs between the two utilities.

Figure 4: Sensitivity of Genome Comparison Algorithms



snpdiff

The snpdiff tool compares SNP calls to a Complete Genomics variant file. It is particularly useful for comparing a Complete Genomics variant file to SNP calls provided by an alternative sequencing or genotyping platform that only produces SNP calls. The input SNP calls must be in a tab-delimited file with columns as in Figure 5. Note that the order of columns can vary but column titles must be conserved; the *Genotypes* column is optional.

Figure 5: SNP Calls As Input to snpdiff

```
Chromosome    Offset0Based  GenotypesStrand     Genotypes
chr13         17919222      +                   CC
chr13         17919650      +                   AT
chr13         17920392      +                   NN
chr13         17921548      +                   TT
```

Here, the *Genotypes* column specifies the base call for each allele. The output produced for this input may be something along the lines of Figure 6:

Figure 6: Output from snpdiff

Chromosome	Offset0Based	GenotypesStrand	Genotypes	Reference	Variants	DiscordantAlleles	NoCallAlleles
chr13	17919222	+	CC	T	CC	0	0
chr13	17919650	+	AT	A	AA	1	0
chr13	17920392	+	NN	G	GN	0	1
chr13	17921548	+	TT	T	.-	0	0

The result for each allele described in the *Variants* column above are any base call ("A", "C", "G", or "T"), a no-call ("N"), a deletion ("-"), or a larger variation that is not consistent with a SNP at all ("."). To compare the SNP calls to the calls in the variant file, snpdiff first determines the variant file's calls at the given position. The algorithm used is sensitive to the canonical alignment, and it is aggressive in terms of making a base call at positions where the call does not have a *varType* of "snp" or "ref". That being said, it is tested to be largely concordant with SNP calls made by several alternative technologies. A discordance found by snpdiff is likely to be a true discrepancy between the calls made by the SNP caller and the variant file. The algorithm employed by snpdiff is as follows, for each allele:

- Find the call in the variant file that overlaps the position in question. Use this call alone to determine the base call for the position in question.
- Walk the *alleleSeq* column of the call from the right and left until reaching the position in question. For each direction, any of the following outcomes may be reached:
  - WALK_OK – The position in question was reached.
  - WALK_EOS – The end of alleleSeq was reached before getting to the position in question.
  - WALK_INCOMPATIBLE – A base call incompatible with the reference base was found at some position before reaching the position in question.
  - WALK_LENGTH_NOCALL – A length no-call (represented by "?") is discovered before reaching the position in question.
- Combine the results of the walk from the right and left to determine the result. The results are combined by the following rules:
  - If the walk from the left and right both end up at the position of interest (WALK_OK):
    - If the base calls discovered by the two walks are in conflict, declare a larger variation (".").
    - If the base calls discovered by the two walks are consistent and at least one is called, use the base call.
    - If both walks end up with a no-call ("N"), the result is no-call.
  - If only one walk ends up at the position of interest (WALK_OK), use the base discovered by that walk.
  - If neither walk ends up at the position of interest, then:
    - If either walk ends up as WALK_LENGTH_NOCALL, mark the position as no-call ("N").
    - If either walk ends up as WALK_EOS, mark the position as deleted ("-").
    - Otherwise, mark the position as a larger variant (".").

Figure 7 shows some examples. The reference base we wish to determine a call for is highlighted in red:

Figure 7: Algorithm Logic from snpdiff

reference	alleleSeq	Walk L->R	Walk R->L	Outcome
A	C	WALK_OK: C	WALK_OK: C	C
ACGTACGT	ACGTACGT	WALK_OK: T	WALK_OK: T	T
G	CC	WALK_OK: C	WALK_OK: C	C
G	CG	WALK_OK: C	WALK_OK: G	.
ACGT	AGGN	WALK_OK: G	WALK_OK: G	G
ACGT	AGG?	WALK_OK: G	WALK_LENGTH_NOCALL	G
ACGT	?GG?	WALK_LENGTH_NOCALL	WALK_LENGTH_NOCALL	N
ACGT	CGGT	WALK_INCOMPATIBLE	WALK_OK: G	G
ACGT	CGGG	WALK_INCOMPATIBLE	WALK_INCOMPATIBLE	.
CACACAC	CAC	WALK_EOS	WALK_EOS	-

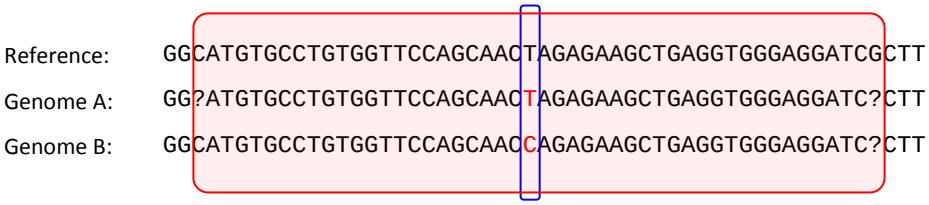
calldiff

The calldiff tool compares two variant files to determine where the two genomes differ, and how. To achieve this, it first gathers variants into superloci, which may account for several nearby variants. It compares the genomes for each superlocus then refines the comparison result to get call-level and locus-level detail. For example, it can be used to help find potential somatic mutations in a tumor-normal comparison, or to find where two assemblies of the same genome differ.

calldiff is less sensitive than snpdiff to the canonical alignment.

If the superloci are too small, superlocus comparison tends to be overly sensitive to canonical alignment. But if superloci are too large, superlocus comparison tends to allow any sequence from one genome to match in a gap of unknown sequence in the other genome. As an example of a superlocus that is too large, suppose we had the sequence from a haploid chromosome of two genomes shown in Figure 8:

Figure 8: Example of a Superlocus that is too Large



The superlocus is circled in red. Genomes A and B are consistent when considering the red superlocus as a whole because the called sequence between the "?" characters in Genome A may be aligned to the "?" character of Genome B.

When considering the red superlocus in Figure 8, and when interpreting the meaning of the calls literally, we can see that all the called bases between the "?" characters in Genome A may be aligned to the "?" character of Genome B, and the genomes are consistent. But when considering the blue box to be the superlocus, we see that the genomes are inconsistent. In different contexts, one superlocus or the other may be preferable, but generally for most comparisons, we would want a comparison algorithm in this case to state the inconsistency between the genomes. To achieve this, a comparison algorithm must either be very precise about how to compare superloci or very precise about how to define a superlocus:

- *Precise about how to compare superloci:* such as when using the red superlocus, determine that there is enough high complexity and uncommon sequence between the "?" characters in Genome A that the SNP in the middle must be aligned as called.
- *Precise about how to define a superlocus:* such as always use the blue superlocus in this situation.

calldiff achieves its specificity by being precise about its superlocus definition.

To determine the superloci, calldiff begins by labeling each reference region containing a variant in either variant file as a superlocus. The superloci are then extended according to the following criteria:

1. **Circular prefix/suffix matching.** For every call whose *alleleSeq* does not contain "N" or "?", do prefix matching to the right along the reference and suffix matching to the left along the reference of both the alleleSeq and the reference sequence, such that the superlocus extension does not exceed P bases (the P limit is necessary to limit the superlocus size for pathological situations). For example, if the call is for an insertion of "ACGT" and the reference sequence directly to the right is "ACGA", three prefix bases of the alleleSeq can be matched to the reference sequence directly to the right, indicating that an equivalent insertion exists at each position in that range. So the superlocus must be extended to account for any variants within three bases to the right of the variant. Additionally, in the example above, if the sequence directly to the right of the call was "ACGTACGA", then the entire insertion of four bases can be prefix matched, and continuing along the reference, the next three bases also match the prefix of the insertion. (This is circular prefix matching.) So the superlocus must be extended to the right by seven bases.
2. **Fixed base count.** Always extend superloci to the right and left by N bases, where N is a command-line configurable parameter. Currently, this parameter defaults to 0.
3. **Fixed count of distinct 3-mers.** Always extend by M distinct reference 3-mers to the right and left, where M is a command-line configurable parameter. In regions of low reference sequence complexity, this results in longer superloci. In regions of high reference sequence complexity, this results in shorter superloci. Currently, this parameter defaults to 4.

After the superloci have been fully extended, overlapping and abutting superloci are combined into a single superlocus.

After superloci have been found, all possible phasings consistent with the *hapLink* values in the calls are used to produce hypotheses about what the genome sequence is, for each variant file. Then each permutation of each hypothesis (one permutation for haploid, two for diploid, and six for triploid) is compared to each hypothesis of the other variant file according to a literal interpretation of their sequence. In other words, any number of bases may align against length no-calls ("?"). The best comparison is produced, such that the number of discordant haplotypes is minimized. The alleles of the best comparison are then segmented to get call-level comparison results. The call-level comparison results are defined to be no worse than the result for the allele as a whole; if a segment comparison results in a worse comparison result than the allele as a whole, the allele's comparison result is used in its place. The call-level comparison results are then used to classify the comparison of each locus as a whole.

The results of calldiff are, for each allele, a comparison classification as described in Table 1.

Table 1: Classification of Comparison Results

Classification	Description
ref-identical	The alleles of the two variant files are identical, and they are consistent with the reference.
alt-identical	The alleles of the two variant files are identical, and they are inconsistent with the reference.
ref-consistent	The alleles of the two variant files are consistent, and they are consistent with the reference.
alt-consistent	The alleles of the two variant files are consistent, and at least one is inconsistent with the reference.
onlyA	The alleles of the two variant files are inconsistent, but only file A is inconsistent with the reference.
onlyB	The alleles of the two variant files are inconsistent, but only file B is inconsistent with the reference.
mismatch	The alleles of the two variant files are inconsistent with each other and with the reference.
phase-mismatch	The two variant files would be consistent if the *hapLink* field had been empty, but the *hapLink* entry causes them to be inconsistent.
ploidy-mismatch	The superlocus did not have uniform ploidy.

For non-haploid superloci, the comparison results for the alleles are joined by a semi-colon. For example, for a diploid hypothesis where variant file A calls reference and variant file B calls a het SNP, you might have a comparison result that looks like “ref-identical;onlyB”.

For example, suppose we use calldiff to compare a tumor genome (file A) and a normal genome (file B) from the same individual. We can find purported somatic mutations by looking for “ref-identical;onlyA”. We can find purported loss of heterozygosity (LOH) by looking for “ref-identical;onlyB” or “alt-identical;onlyA”. We might expect fewer superloci classified as “alt-identical;onlyB”, as the likely reason for this is assembly error – overcall in the normal genome.

calldiff for scoring somatic variations (beta)

Somatic variation discovery is an important use case for calldiff. Because we may expect thousands of erroneous variant calls in a genome and thousands of true somatic variations, we need a mechanism to tease apart the true somatic mutations from false somatic mutations.

The calldiff tool uses the scores provided in Complete Genomics data to determine which somatic mutations are called with higher confidence, and simplifies the information into a single somatic score. It does so for all loci where the genome A has a simple variation (a single SNP, DEL, INS, or SUB) and genome B is called as reference. This functionality is provided through the Somatic Output report, which annotates the somatic calls from the variant file with the *SuperlocusId*, *locusClassification*, the *VarScoreA*, *RefScoreB*, *SomaticCategory*, *VarScoreARank*, *RefScoreBRank*, and *SomaticScore*. Figure 9 shows a portion of an example Somatic Output report; the definitions of the score columns follow.

Figure 9: Example of Somatic Output

SuperlocusId	LocusClassification	locus	…	varType	reference	alleleSeq	…	VarScoreA	RefScoreB	SomaticCategory	VarScoreARank	RefScoreBRank	SomaticScore
220	het-sub	15	…	sub	TCC	ACT	…	401	50	diploid-indel	0.418	0.052	0.102
427	hom-snp	16	…	snp	G	T	…	313	63	diploid-snp	0.180	0.099	0.188
659	het-ins	18	…	ins		A	…	52	102	diploid-indel	0.031	0.348	0.062
753	no-call-ins	19	…	ins		T	…	89	19	diploid-indel	0.090	0.005	0.010
924	het-snp	21	…	snp	C	T	…	65	50	diploid-snp	0.008	0.052	0.016

To determine the somatic score of a variant in genome A but not genome B, loci of genome A are first categorized as haploid-snp, haploid-indel (for DEL, INS, or SUB loci), diploid-snp, or diploid-indel. This categorization is called the “somatic category” and is recorded in the column of that name.

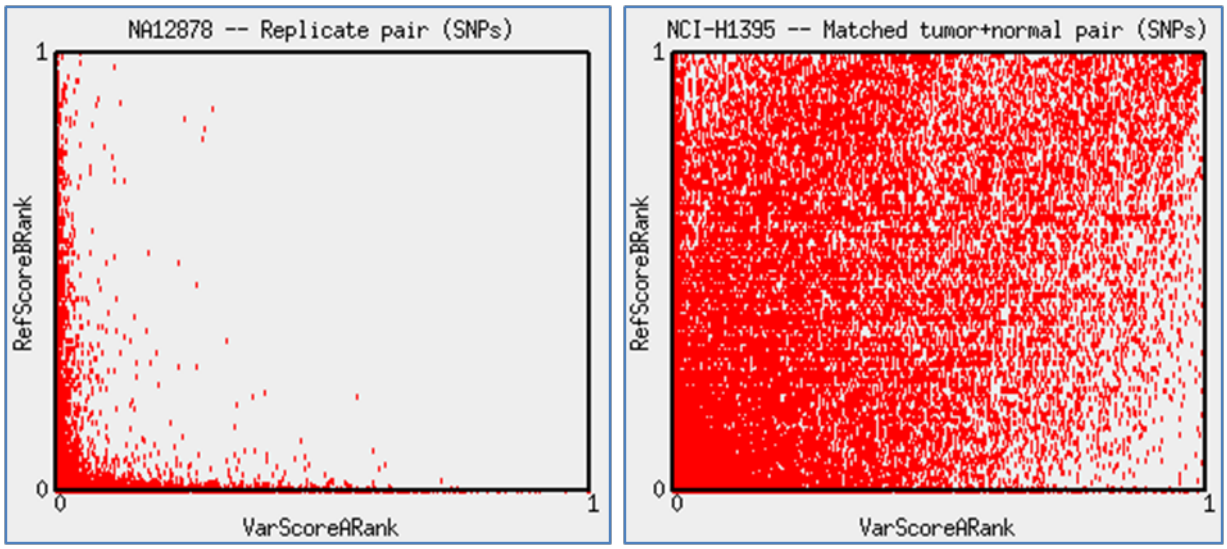
VarScoreA for each locus is then assigned, based on the input data, as indication of the strength of evidence for the variant. For haploid-snp or diploid-snp loci, *VarScoreA* equals the negative of the reference score. For haploid-indel or diploid-indel loci, *VarScoreA* equals the evidence score. In either case, a greater *VarScoreA* indicates increased confidence, but *VarScoreA* is not necessarily comparable from one *SomaticCategory* to another.

RefScoreB is also assigned for each variant present in genome A and not B. *RefScoreB* is the reference score at the locus in genome B and does not depend on the *SomaticCategory*.

From *VarScoreA* and *RefScoreB*, we compute *VarScoreARank* and *RefScoreBRank*. *VarScoreARank* is the fraction of all variant loci in genome A in a *SomaticCategory* whose *VarScoreA* is less than the *VarScoreA* for this locus. *RefScoreBRank* is an estimation of the fraction of all loci where genome B is correctly called as reference and has a *RefScoreB* less than the *RefScoreB* for this locus.

Given these definitions of *VarScoreARank* and *RefScoreBRank*, we might expect both *VarScoreARank* and *RefScoreBRank* for true somatic mutations to be distributed uniformly. The distribution for false somatic variants corresponding to false positives in the tumor will have low values of *VarScoreARank*, and the variants corresponding to false negatives in the normal will have low values of *RefScoreBRank*. Figure 10 shows an example of the distributions.

Figure 10: Scatterplots of RefScoreBRank versus VarScoreARank for Somatic Variant Calls



True somatic variants are expected to be evenly distributed on the plot. False positive variant calls in the tumor are expected to cluster near the y axis. False negatives in the normal are expected to cluster near the x axis. For the replicate pair, all the somatic variants are false and cluster near the axes. For the tumor+normal pair, a large fraction of the called somatic variants are true, so they are better distributed on the plot.

Suppose we filter somatic variants where *VarScoreARank* < X or *RefScoreBRank* < X. Under the assumption that true somatic mutations have evenly distributed values of *VarScoreARank* and *RefScoreBRank*, the relative sensitivity of the filtered data set is $(1 - \mathrm{X})^2$. The filtered set of mutations corresponds to the upper right area of the scatter plots given in the above figure. *SomaticScore* is computed as:

$$SomaticScore = 1 - (1 - \min(VarScoreARank, RefScoreBRank))^2$$

Thus, filtering where *SomaticScore* < Y is equivalent to filtering the data set where *VarScoreARank* or *RefScoreBRank* is less than $\mathrm{X} = 1 - \sqrt{1 - Y}$. As shown before, the sensitivity of such a filter is $(1 - X)^2$, so the sensitivity of the filter where *SomaticScore* < Y is 1-Y. We can consider *SomaticScore* to be the same as 1- sensitivity. As a result, a *SomaticScore* threshold applied across different genomes will represent the same relative sensitivity. Figure 11 shows an ROC curve plotting the expected sensitivity as a function of the number of somatic mutations. At an expected sensitivity of 90%, there are 174 false somatic SNPs in the NA12878 replicate pair, or one false somatic SNP every 17.7 Mb.

Figure 11: ROC Curve for Somatic Variant Calls

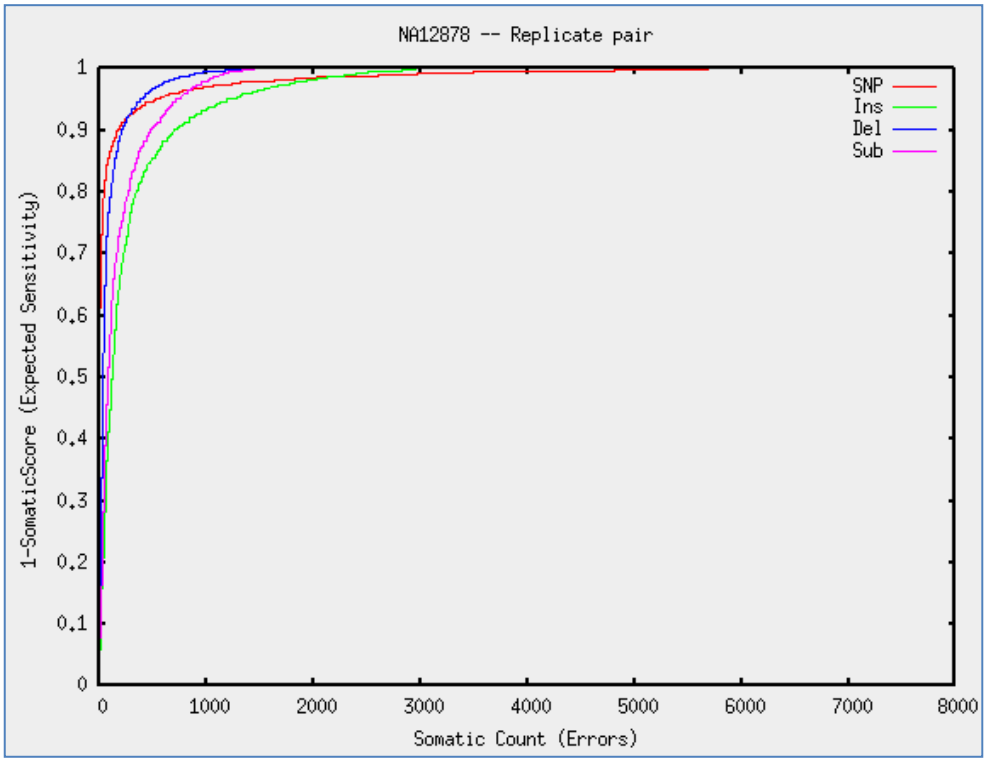


Table 2 lists the number of apparent somatic variations observed at various thresholds.

Table 2: False Positive and Total Discovered Somatic SNVs at Various Somatic Score Thresholds

Somatic Score Threshold	False Positives (Genome-wide, estimated from NA12878 replicates)	Somatic SNVs Discovered in NCI-H1395
0.00	7,522	34,600
0.01	2,609	28,150
0.02	1,501	26,348
0.03	971	25,402
0.04	705	24,715
0.05	519	24,165
0.10	174	22,092

The trade-off between sensitivity and specificity achievable in any particular tumor-normal comparison would vary depending on several factors:

- The extent of aneuploidy, impurity, and heterogeneity within the tumor sample
- The coverage characteristics of the two genomes
- The number of true somatic events

Many-Genome Comparison: listvariants (beta)

The superlocus approach to genome comparison achieves a good combination of sensitivity to genomic variation and insensitivity to canonical alignment for a small number of genomes. But as the number and variety of genomes grows, superloci can grow arbitrarily large. As shown in "calldiff," when superloci grow too large, the literal interpretation of sequence compatibility employed by calldiff tends to be insensitive to real genomic differences.

cgatools supports many-genome comparison through the combination of two tools: listvariants and testvariants. The listvariants command lists all the genomic mutations found in an arbitrary number of genomes, and the testvariants command tests each of those mutations against a set of genomes to determine the compatibility of the genomes to each mutation.

The listvariants command merges and lists all the fully called mutations from a set of variant files (that is, each line from each variant file that is fully called and inconsistent with the reference). listvariants is as specific as it can be about mutations without splitting up called mutations from the variant file. For example:

Reference:	CGAATTACAT
Allele 1:	CGCATTATAT
Allele 2:	CGAATTACAT

In this case, suppose the variant file listed this sequence as two SNP mutations with the same *hapLink* to indicate they are on the same haplotype. In this case, listvariants also lists the two SNPs separately. In this way, the many-genome comparison can be very specific about where genomes differ, although it loses information about which variants occurred on the same haplotype as other variants.

listvariants also canonicalizes any input variants it encounters before writing them to the output. It uses the rightmost variant that is equivalent to the input variant. For example:

Reference:	CG-AAAAA-CAT
Alternative 1:	CG-AAAAAACAT
Alternative 2:	CGAAAAAA-CAT

The two alternatives above have the same sequence, but have different alignments against the genome. If the input genomes list both insertions, the alternative 2 alignment is canonicalized (transformed) into the alternative 1 alignment because it is the rightmost alignment that describes an equivalent sequence. The two variants are then merged as equivalent, and a single output record is produced.

The output records retain the annotations present in the input variant file. Figure 12 shows an example:

Figure 12: Output Records Retain Input Annotations

```
variantId chromosome  begin     end     varType  reference   alleleSeq xRef
1034      chr1        972803    972804  snp      T           C         dbsnp:rs3128102
1035      chr1        972856    972857  snp      T           C         dbsnp:rs10267
1036      chr1        975024    975025  snp      G           T
1037      chr1        975128    975129  snp      C           T         dbsnp:rs2275813
1038      chr1        975311    975313  sub      GG          A         dbsnp:rs56255212
1039      chr1        975322    975323  snp      T           C         dbsnp:rs2275811
1040      chr1        975371    975372  snp      G           A
1041      chr1        975900    975901  snp      G           A
```

Many-Genome Comparison: testvariants (beta)

The testvariants command processes the variants listed by listvariants and writes each input record to the output, along with a flag for each allele of each genome to indicate if the variant is present on that allele. Table 3 lists set of possible flags for each allele:

Table 3: testvariants Flags

Flag	Description
0	The allele is inconsistent with the variant.
1	The allele is fully called, and is consistent with the variant.
N	The allele has no-calls, and the allele is consistent with the variant.

For example, for the listvariants example in Figure 12, the testvariants output against three genomes is shown in Figure 13:

Figure 13: testvariants Output against Three Genomes

```
variantId  chromosome  begin   end     varType  reference  alleleSeq  xRef              ASM1  ASM2  ASM3
1034       chr1        972803  972804  snp      T          C          dbsnp:rs3128102   11    01    11
1035       chr1        972856  972857  snp      T          C          dbsnp:rs10267     11    11    11
1036       chr1        975024  975025  snp      G          T                            00    01    NN
1037       chr1        975128  975129  snp      C          T          dbsnp:rs2275813   00    01    00
1038       chr1        975311  975313  sub      GG         A          dbsnp:rs56255212  11    1N    1N
1039       chr1        975322  975323  snp      T          C          dbsnp:rs2275811   11    01    1N
1040       chr1        975371  975372  snp      G          A                            00    01    00
1041       chr1        975900  975901  snp      G          A                            1N    00    01
```

In this example, ASM1 and ASM3 are homozygous for variant 1034, but ASM2 is heterozygous. ASM1 does not have variant 1036, but ASM2 is heterozygous for the variant, and ASM3 is no-called at that position.

The testvariants command tests each variant against each genome independently. To do so, it first constructs a one-genome superlocus to keep track of which loci may be used in the comparison. The superlocus is constructed using the same superlocus rules as calldiff, except the "Fixed count of distinct 3-mers" used is 6 instead of 4.

After a superlocus has been found, all possible phasings consistent with the *hapLink* values in the calls are used to produce hypotheses about what the genome sequence is. testvariants chooses the phasing that results in the most 1's, then the most 0's.

To test a particular phasing, for each allele, testvariants first finds a base set of calls consisting of the minimal set of calls overlapping the variant, extended to the left and right according to the prefix/suffix matching rule of calldiff. Then testvariants compares the call sequence to the variant sequence (extended to the left and right by reference sequence), for every sequential sequence of calls in the superlocus that covers the base set of calls. This results in "1" if any compared sequence matches the variant sequence, or "N" if any compared sequence is compatible with the variant sequence but contains no-calls. Otherwise, the result is "0".

Although the testvariants algorithm may achieve a reasonable middle ground between sensitivity to real genomic variation and insensitivity to canonical alignment for the many-genome comparison problem, the following limitations apply:

- testvariants is more sensitive to the canonical alignment than calldiff. As such, it is not the ideal tool for comparing a small number of genomes (such as 2 or 3).
- For simplicity of output file format and interpretation, testvariants does not transfer the score of the variant calls or the reference scores in the input genomes to the output file. Not having scores further limits the testvariants output for use in analyzing the genomic differences of a small number of genomes.

junctiondiff (beta)

The junctiondiff tool finds junctions present in one genome (genome A) but not another (genome B). It addresses the following pitfalls when comparing the set of junctions present in two genomes:

- Junction coordinates are not always exact.

  When a junction's sequence cannot be resolved (indicated when the value in the *JunctionSequenceResolved* column of the junction file is "N"), the coordinates of the junction are estimations based on the expected distribution of mate gaps of the DNB reads that contribute to the discordant mate pair analysis. To resolve this issue, junctiondiff considers all junction coordinates within N bases to be equivalent. Here, N is controlled by the command-line configurable parameter "`--distance`".

- Junction coordinates for the same junction are not always the same.

  Just as for short indels, the same junction sequence can sometimes be described at slightly different coordinates. Additionally, small variants near the junction may cause the junction coordinates to shift slightly. This pitfall is again resolved by the "`--distance`" command-line parameter.

- Complete Genomics junction detection algorithm has low sensitivity to very short deletions.

  Slight differences in mate gap distribution or slight changes in coverage bias characteristics for the two genomes may mean that a junction that is present with good support in genome A is not called in genome B due to lack of support. To resolve this issue, junctiondiff provides the option "`--minlength`" to filter out junctions consistent with short deletions.

- There is low sensitivity for junctions with few discordant mate pair alignments.

  Complete Genomics junction caller requires 3 discordant mate pair alignments of support to call a junction. If a junction has a low expected count of discordant mate pair alignments (for example, 3), whether the junction achieves sufficient support is a matter of chance. To address this issue, junctiondiff provides the option "`--scoreThresholdA`" to filter out junctions with few discordant mate pair alignments.

- Genome sample B may be contaminated by genome sample A.

  This often occurs, for example, if genome sample B is a tumor and genome sample A is the matched normal. For this reason, the junctiondiff tool allows you to specify a cutoff of support using the "`--scoreThresholdB`" option for genome B.

Format Conversion Tools

The primary goal of the Complete Genomics export formats is to represent the data in a concise and simple way. As such, they are not always the best formats for doing certain kinds of data processing. Moreover, some users have existing programs that expect inputs in various other data formats. As a result, **cgatools** aims to provide data conversion capabilities.

map2sam

The map2sam tool converts Complete Genomics exported reads and initial reference mappings to the SAM format. For pipelines that require eventually converting to the BAM format, the output of map2sam can be sent to standard output, which can be processed by SAM Tools. For example, this command pipeline creates an indexed, reference-sorted BAM file:

```
cgatools map2sam --reads=/path/to/reads.tsv.bz2 \
                 --mappings=/path/to/mappings.tsv.bz2 \
                 --library=/path/to/lib_DNB.tsv \
                 --reference=/path/to/build36.crr | \
                       samtools view -uS - | \
                       samtools sort - result && samtools index result.bam
```

Complete Genomics reads are initially mapped to the reference genome using a fast algorithm, and these initial mappings are later both expanded and refined by a form of local *de novo* assembly applied to putatively variant regions of the genome.

IMPORTANT: The map2sam tool converts the initial reference mappings, and not the additional mappings to variants discovered during the assembly process.

The following additional limitations apply to map2sam output:

- The converted mappings are reference mappings only. The mappings used as evidence to make indel calls are not included.
- SAM does not have strong support for overlapping sub-reads (for example, the negatively sized intra-read gaps), which are present in Complete Genomics data. To represent overlapping reads, the strongest base call is put in the SAM mapping record, and the alternative base calls are represented using the GS/GQ/GC tags of the mapping record.
- The SAM validator provided by the Picard project (picard.sourceforge.net) does not allow specifying a primary mapping for reads that do not have consistent mates. As a result, only reads which have consistent mate pair mappings have a mapping marked as the primary mapping record (mapping record with the “not primary” FLAG, 0x0100, set to 0).
- The NM tag (edit distance to reference sequence) is not currently produced by map2sam.
- The R2 and Q2 tags (mate sequence and quality scores) can be generated optionally.

Representation of the Complete Genomics data in the SAM output

The detailed descriptions of the map2sam output in Table 4 and Table 5 assume some familiarity with Complete Genomics data and terminology. We recommend you consult the Complete Genomics Data File Formats document and FAQs if you are mostly familiar with other Next-Gen platforms. These documents can be obtained from support@completegenomics.com. Additionally, the Complete Genomics assembly process and some of its implications are described in the Complete Genomics technology whitepaper and in more detail the *Science* paper. We recommend you consult the Complete Genomics FAQ documents in considering how to best use these data. See "References" for more information on these documents.

This description is based on the SAM Format Specification described in "References."

Table 4: Header Fields

Section	Tag	Value	Description	Example
@HD	VN	0.1.2	Version of SAM spec	
	SO	"DnbId sorted"	Sort Order. **Note:** A DNB is a clone.	
@SQ	SN	*ChromosomeName*	Sequence Name. Included for all chromosomes in the reference genome.	SN:chr1
	LN	*ChromosomeLength*	Length of the chromosome. Included for all chromosomes in the reference genome.	LN:247249719
	UR	*ReferenceFilePath*	Path to the input reference file	UR: reference.crr
	AS	*ASM ID*	CGI Data Analysis Pipeline Run ID	AS:GS19240-ASM
@RG	ID	*LaneID*	Slide and Lane ID	ID:GS08081-FS3-L02
	SM	*SampleID*	Sample Id	SM:GS00028-DNA-C01
	LB	*LibraryID*	Library ID of the library	LB:GS00433-CLS
	PU	*LaneID*	Slide and Lane ID	PU:GS08081-FS3-L02
	CN	"Complete Genomics"	Name of sequencing center producing the read	
	DT	*ExportDate*	The CGI data analysis timestamp stored in the CGI reads file.	DT:2010-01-21
	PL	"Complete Genomics"	Platform/technology used to produce the read.	
@PG	ID	"cgatools"	Program name	
	VN	*Version*	Program version	VN:0.5.0
	CL	*Command line*	Command line	Complete string

Table 5: Mapping Record Fields

Field	Value	SAM Definition	map2sam Usage
QNAME	*SlideID-LaneID*: *DnbOffset*	Query Name	The QNAME value is constructed from the full lane Id (*SlideId+LaneId*) and 0-based DNB offset from the beginning of the Reads file provided as input. For example: GS08081-FS3-L02-3:244
FLAG	0x0001	The read is paired in sequencing.	The flag is always set for CGI data. The current CGI technology always produces paired reads.
	0x0002	Each fragment properly aligned according to the aligner	The flag is set to 1 if both mates are mapped.
	0x0004	The query sequence itself is unmapped.	The flag is set when there are absolutely no mappings found for this HalfDNB.
	0x0008	The mate is unmapped.	The flag is set only when there are no mappings found for this HalfDNB's mate.
	0x0010	Strand of the query	0 for forward; 1 for reverse strand.
	0x0020	Strand of the mate	0 for forward; 1 for reverse strand.
	0x0040	The read is the first read in a pair.	The flag is set if the current HalfDNB is from the 5' end of the original cloned insert.
	0x0080	The read is the second read in a pair.	The flag is set if the current HalfDNB is from the 3' end of the original cloned insert.
	0x0100	The alignment is not primary.	The flag is set if there is a better mapping of the same HalfDNB having higher value of *MAPQ* (see MAPQ in this table) or the other HalfDNB is not mapped.
	0x0200	The read fails platform/vendor quality checks.	Always set to 0.
	0x0400	The read is either a PCR duplicate or an optical duplicate.	Always set to 0.
RNAME	*ChromosomeID* or "*"	Reference sequence NAME	Can be "*" if this HalfDNB doesn't have mappings. If the HalfDNB is not mapped itself but has a mapped mate the RNAME of the mate is reported.
POS	*Current Mapping Position* or 0	1-based leftmost POSition/coordinate of the clipped sequence	The position reported in a Mappings file record from a CGI export package offset by 1 (CGI export format reports mapping positions 0-based). 0 is reported if there are no mappings found for this HalfDNB and there are not mapped mates. If the HalfDNB is not mapped itself but has a mapped mate the POS of the mate is reported.
MAPQ	*CG_Mapping weight*	MAPping Quality (Phred-scaled probability that the mapping position of this read is incorrect.)	The probabilities are reported in different ranges for consistent pair reads (Flag 0x0002, case 1) and for non-paired mappings. CGI does not recommended that values of consistent and inconsistent mappings be directly compared.
CIGAR	C*igarString* and GS/GQ/GC flags	Extended CIGAR string	Currently, CGI initial mappings files do not allow insertions or deletions. Therefore, only M and N operations are used in the *CIGAR* field. The negative gaps are represented using GS/GQ/GC flags. The CIGAR sequence will represent the positive gaps using N and ignore the negative gaps.

Field	Value	SAM Definition	map2sam Usage
MRNM	"=" or *ChromosomeID* or "*"	Mate Reference sequence NaMe; "=" if the same as *RNAME*	Reports "*" if there is no consistent mate found.
MPOS	*MatePosition* or 0	1-based leftmost mate POSition of the clipped sequence	Reports 0 if there is no consistent mate found.
ISIZE	*DistanceToMate* or 0	Inferred Insert SIZE	The distance between the consistent mate start position and the start position of the current HalfDNB mapping. The value is 0 if the mates are mapped to different chromosomes.
SEQ	*Sequence*	Query SEQuence; "=" for a match to the reference; n/N/. for ambiguity	The regions of overlapping bases in the negative gaps contain the bases with higher scores.
QUAL	*QualityScores*	Query QUALity; ASCII-33 gives the Phred base quality	The values are copied from the corresponding record of the CGI Reads file.
TAG	GS/GQ/GC RG R2/Q2 XS	Tags	GS/GQ/GC flags are used to represent CGI-specific negative wobble gaps in HalfDNBs. See SAM Format Specification in "References" for the description of the flags. RG is a standard tag containing the read group name. R2, Q2 – are the optional standard tags. XS:I:1 the optional user-defined tag is marking an SV candidate.

Rules to Set the "not primary" Flag (0x0100)

The flag "not primary" is set for a HalfDNB mapping in the following cases:

- There is another mapping of the same HalfDNB having a higher *MAPQ* value.
- The mapping of a HalfDNB doesn't have a consistent mate pair mapping, and there are mappings found for the mate HalfDNB.
- The mapping's best mate has a best mate that is not the current mapping.

Combining Mapping Records in SAM

1. The best mapping pair (the best score) of a DNB is reported with the "non-primary" flag set to 0. Both mappings should refer to each other as the best mates.
2. All the other mappings of that DNB are reported in non defined order and have the "non-primary" flag set to 1.
3. If both HalfDNBs are mapped uniquely but not consistently, they are not reported as primary ("non-primary" flag is set to 0) even though they are not consistent mates.
4. If only one HalfDNB is mapped, the best mapping of that HalfDNB is reported followed by a "non-mapped" mapping record of the other HalfDNB. The alignment position of the other HalfDNB is set to the same values as the mapped HalfDNB and the "non-primary" flag of the records is set to 0. The not mapped record will be marked as "not mapped" by appropriate flags.
5. All the other mappings of the mapped read from number 4 above are reported one record per mapping having "non-primary" flag set to 1.
6. All the not mapped reads are reported in pairs aligned to the 0 position. The alignment position is important to keep the records together while sorting and merging BAM files.

evidence2sam (beta)

The evidence2sam tool converts Complete Genomics evidence mappings to the SAM format. The current implementation is in beta form. For pipelines that require eventually converting to the BAM format, the output of evidence2sam can be sent to standard output, which can be processed by SAM Tools. For example, this command pipeline creates an indexed, reference-sorted BAM file:

```
cgatools evidence2sam \
        --beta \
        --evidence-dnbs=/path/to/evidenceDnbs-chrN-XXX.tsv.bz2 \
        --reference=/path/to/build36.crr | \
    samtools view -uS - | \
    samtools sort - result && samtools index result.bam
```

Complete Genomics evidence mappings are the mappings that were used to call variations found by the CGI genome assembly process. The assembly process uses a local *de novo* method to find likely alleles for a variation interval (small region of the genome, typically less than 200 bases), then an optimization process to refine the allele choices. The evidence mappings are DNB alignments that indicate support for the best hypothesis found during the assembly process. The evidence2sam tool can be used to convert these mappings to SAM for visualization in a genome browser like IGV. The details of the Complete Genomics data representation in the SAM output are covered in the map2sam tool description in this document.

In two situations, a DNB may have multiple mapping records present in the evidence DNB mappings provided by Complete Genomics. First, if the best hypothesis is heterozygous and contains two non-reference alleles, support is also given for the reference allele. In this case, if a DNB supports two of the three alleles equally well (or similarly well) and not the third allele, then the evidence DNB mappings contain a record showing alignment of the DNB to each of the two alleles it supports. Second, if there are two regions of the genome with similar sequence such that DNBs align well to either sequence, those DNBs may be used as evidence for alleles in both regions. A post-processing step of the CGI assembly process finds such regions and no-calls them.

Because most tools that visualize SAM do not have rich features to specify an allele a DNB maps against, visualization of the duplicate mapping records present in the evidence can be confusing. For this reason, the evidence2sam tool has an option to de-duplicate the mappings present in the evidence, for both forms of duplication described above, for duplicate DNB mappings that are nearby on the reference. Specifically, the evidence2sam tool de-duplicates using the following algorithm, for each variation interval:

1. Update the read-ahead buffer to ensure it contains all evidence mapping records up to 1 Kb to the right of the position of the rightmost evidence mapping record for this interval.
2. Moving from position 0 to the end of the chromosome, processing each mapping record of the current variation interval as follows:
    a. Collect all the mappings of the same DNB that belong to the current interval or mappings from different intervals that overlap the corresponding arm/both arms of the selected DNB.
    b. Run one-DNB de-duplication. This operation deletes all the collected DNB mappings from the buffer except the “best” one.
    c. Write the “best” mapping into the SAM output stream.
    d. Remove the “best” mapping from the buffer and proceed to the next mapping in the current interval.
    e. If the last mapping in the current interval has been processed, remove the mappings processed for the current interval from the read-ahead buffer.

During de-duplication, the following rules are used to determine the best mapping for a DNB:

1. If several mapping records belong to the same variation interval, leave only the record that has maximum mapping quality.
2. If several mapping records belong to adjacent variation intervals (same side and strand), leave only the record that has maximum mapping quality.
3. If there are only two mapping records in the set and their different arms support different intervals, construct a composite mapping inheriting MAPQ, position in the reference, and reference alignment from a corresponding mapping record.
4. If there is still more than one mapping record in the input set, select the mapping with highest MAPQ and remove the other mappings.

The following additional limitations apply to evidence2sam output:

- The converted evidence support mappings are the mappings that belong only to the regions where variations were called.
- SAM does not have strong support for overlapping sub-reads (e.g., the negatively sized intra-read gaps), which are present in Complete Genomics data. To represent overlapping reads, the strongest base call is put in the SAM mapping record, and the alternative base calls are represented using the GS/GQ/GC tags of the mapping record.
- When the option to de-duplicate mapping records is not used, evidence2sam reports all mappings as non-primary mappings.
- The NM tag (edit distance to reference sequence) is not currently produced by map2sam.
- The R2 and Q2 tags (mate sequence and quality scores) can be generated optionally.
- XI:I – an optional tag, contains the number of the evidence interval the mapping comes from.
- XA:I – an optional tag, contains the number of the allele the mapping comes from.

generatemasterVar (beta)

The generatemasterVar tool produces a simple, integrated master variation file (***masterVar***) to report the variant calls and annotation information produced by the Complete Genomics assembly process. The file format is derived heavily from the existing variation file format and can be used with all **cgatools** commands anywhere a variation file is expected. This format has the following important features:

- The format includes one line for any given locus of the genome. All lines are in the same format and contain the same number of fields, although some fields may be blank for certain types of loci. This allows for easier processing of the data with simple command line tools such as awk or grep.
- The format integrates annotation information from data in other Complete Genomics export files. For example, loci are annotated with read counts from the evidence files and with copy number calls from the CNV result files.
- For every locus line, the *zygosity* field can be used to quickly determine if the locus is fully called on one, both or none of the alleles. Fully called loci are further classified into haploid, homozygous, heterozygous reference (where one of the alleles is equal to the reference), and heterozygous alternate (where neither of the alleles is equal to the reference).
- Loci that contain simple isolated variations (SNP, INS, DEL or SUB) can be easily identified using the *varType* field.
- Users can filter the data by removing locus data lines. The regions that correspond to any removed lines are treated by **cgatools** as if the locus was no-called on all alleles.
- Users can add extra annotation columns that are appended to every line. **cgatools** commands that both consume the format and produce it as output will transfer the additional columns intact.

- The format provides a structured content that can more easily be converted into other standard variation file formats.

Table 6 summarizes the columns that must always be present in the file.

Table 6: Mandatory Columns

Column Name	Description
locusId	Integer ID of the locus. When converting a Complete Genomics variant file, all loci will retain the original IDs. When processing filtered files where regions have been removed, the loci that correspond to the removed regions are recreated with a locus ID 0 and are considered fully no-called.
ploidy	Number of alleles (same as in the Complete Genomics variant file).
chromosome	Chromosome name (same as in the Complete Genomics variant file).
begin	Locus start. Zero-based offset of the first base in the locus, the same as in the Complete Genomics variant file.
end	Locus end. Zero-based offset of the first base downstream of the locus, same as in the Complete Genomics variant file.
zygosity	Call completeness and zygosity information. *zygosity* is assigned one of the following values: `no-call` All alleles are partially or fully no-called. `half` Diploid locus where one of the alleles is fully called and the other contains no-calls. `hap` Haploid, fully called locus. `hom` Diploid, homozygous, fully called locus. `het-ref` Diploid, heterozygous, fully called locus where one of the alleles is identical to the reference. `het-alt` Diploid, heterozygous, fully called locus where both alleles differ from the reference.
varType	Variation type for simple, isolated variations. *varType* is assigned one of the following values: `snp`, `ins`, `del`, or `sub` Fully called or half-called locus that contains only a single isolated variation. `ref` Fully called or half-called locus that contains only reference calls and no calls and at least one allele is fully called. `complex` Locus that contains multiple variations or has no-calls in all alleles. This is also the value for all loci where the reference itself is ambiguous. `no-ref` Locus where the reference genome is N. `PAR-called-in-X` Locus on the pseudo-autosomal region of the Y chromosomes in males.
reference	Reference sequence. Loci called as homozygous reference and loci that are fully no-called on all alleles will contain "=" instead of the literal reference sequence.
allele1Seq	Sequence of the first allele. May contain "N" (one-base-no-call) and "?" (unknown-length-no-call) characters, with the same semantics as used in the Complete Genomics variant file. The field is empty whenever the called variant is a deletion of all bases in the locus. For a given locus, if the allele in the variation file spans multiple lines, then the sequences for each call corresponding to that allele are concatenated.
allele2Seq	Sequence of the second allele. The value of *allele2Seq* follows the same rules as *allele1Seq*. This field is always blank for haploid loci (whenever the ploidy field contains `1`). The values of *allele1Seq* and *allele2Seq* are assigned such that a variation allele always precedes a pure reference allele, and a fully called allele always precedes any allele that contains no-calls. As a result, the allele order may differ from the order in the corresponding source variant file.
allele1Score	Minimum score of all calls in the first allele of the locus.
allele2Score	Minimum score of all calls in the second allele. Blank for haploid loci.
allele1HapLink	Integer ID that links the first allele to the alleles of other loci that are known to reside on the same haplotype.

Column Name	Description
allele2HapLink	Integer ID that links the second allele to the alleles of other loci that are known to reside on the same haplotype.

Table 7 describes the annotation information from various sources which can also be included. The values listed in Table 7's Source column are described in Table 8.

Table 7: Annotation Columns

Column Name	Source	Description
xRef	Input File	Semicolon-separated list of all xRef annotations for all alleles of the locus.
evidenceIntervalId	`Evidence`	Integer ID of the interval in the evidence file. Multiple loci may share the same evidence interval.
allele1ReadCount	`Evidence`	Number of reads that support the first allele. A read is included in the count if it overlaps the locus interval and supports the allele by at least 3 dB more than the other allele or the reference. For length-preserving variations, at least one base in the read must overlap the interval to be included in the read count. For length-changing variations, the read may be counted even if it overlaps the variation with its intra-read gap.
allele2ReadCount	`Evidence`	Number of reads that support the second allele. For homozygous loci, this number is identical to *allele1ReadCount*.
referenceAlleleReadCount	`Evidence`	Number of reads that support the reference sequence. For loci where one of the alleles is reference, this number is identical to the read count of that allele.
totalReadCount	`Evidence`	Total number of reads in the evidence file that overlap the interval. Note that this count also includes reads that do not strongly support one allele over the other and consequently are not accounted for in *allele1ReadCount* or *allele2ReadCount*. For loci where one of the alleles contains a no-call, the *totalReadCount* also includes the reads that support that no-called allele. The *totalReadCount* does not include reads that do not overlap the locus, even if they do overlap the evidence interval, and, hence, are present in the evidence file.
allele1Gene	Gene	Semicolon-separated list of all gene annotations for the first allele of the locus. For every gene annotation, the following fields from the gene file are concatenated together using colon as separator: *geneId*, *mrnaAcc*, *symbol*, *component*, and *impact*.
allele2Gene	Gene	Gene annotation list for the second allele, formatted in the same way as *allele1Gene*.
pfam	`Gene`	Pfam domain information that overlap with the locus.
miRBaseId	`ncRNA`	Semicolon-separated list of all ncRNA annotations for this locus.
repeatMasker	Repeat	Semicolon-separated list of all RepeatMasker records that overlap this locus. Within every record, the following data is concatenated together using colon as separator: ▪ repeat name ▪ repeat family ▪ overall divergence percentage (number of bases changed, deleted or inserted relative to the repeat consensus sequence per hundred bases) Mitochondrion loci are not annotated.
segDupOverlap	Segdup	Number of distinct segmental duplications that overlap this locus.

Column Name	Source	Description
relativeCoverage	CNV	Normalized coverage level for the segment that overlaps the current locus (for loci that overlap two segments, the data from the CNV segment with the longer overlap are chosen).
calledPloidy	CNV	Ploidy of the segment. Only present if the ploidy calls were made during the assembly (only when the *calledPloidy* column is present in the source CNV file).

Table 8: Annotation Data Sources

Source	Description
`Input File`	Data is copied from the input file. If the input file is the master variations file (***masterVar-[ASM-ID].tsv.bz2***), its annotation columns are simply copied to the output. If the input file is the variation file (***var-[ASM-ID].tsv.bz2***), the *xRef* data is re-formatted as described above.
`Evidence`	Data from the Complete Genomics evidence export file. Root directory of the genome package must be specified when invoking the tool.
`Gene`	Data from the Complete Genomics gene annotation file (***gene-ASM.tsv***). Root directory of the genome package must be specified when invoking the tool.
`ncRNA`	Data from the Complete Genomics non-coding RNA annotation file (***ncRNA-ASM.tsv***). Root directory of the genome package must be specified when invoking the conversion tool.
`Repeat`	RepeatMasker information. If repeat annotations are required then the RepeatMasker annotation data file must be specified when invoking the command. The file may be downloaded from the Complete Genomics website: ▪ Build 36 — ftp://ftp.completegenomics.com/AnnotationFiles/rmsk36.tsv.gz ▪ Build 37 — ftp://ftp.completegenomics.com/AnnotationFiles/rmsk37.tsv.gz The data is derived from the RepeatMasker table available from the UCSC genome browser website.
`Segdup`	Information about segmental duplications. If segmental duplication data is required, then the location of the segmental duplication data file must be specified when invoking the command. The files for each reference build are available for download from the Complete Genomics website: ▪ Build 36 — ftp://ftp.completegenomics.com/AnnotationFiles/segdup36.tsv.gz ▪ Build 37 — ftp://ftp.completegenomics.com/AnnotationFiles/segdup37.tsv.gz The data is derived from the segmental duplication table available from the UCSC genome browser website.
`CNV`	Information about the CNV calls made by the Complete Genomics pipeline for the region that covers this locus. Root directory of the genome package must be specified when invoking the tool.

In addition to the annotations produced by the conversion tool, the calldiff tool can produce output files in the master variant format, with the additional columns that contain the results of the comparison for every locus, and the somatic ranks and scores for the loci of the first file in the pair.

Modifying the *masterVar* file

When processing the ***masterVar*** file with your own tools, it is important to adhere to the following rules to maintain compatibility with **cgatools**:

1. Header line TYPE must be preserved as is. It is recommended, but not required, that the other header values are also preserved.
2. Every locus data line must contain the same set of columns. The column header line (starting with a ">" character) must be present and must contain the same number of columns as the rest of the file.
3. Order of lines must remain intact. The loci in the file are sorted in the order of the reference.
4. The values of the mandatory columns must not be modified and columns themselves must not be removed. In particular, loci may not be split or merged.

Additional Information about the *masterVar* File

- The allele sequence is a concatenation of all calls from the variation file for the given allele. As a result, in some complex loci, the information about the exact alignment of the called sequence to the reference may be lost.
- Certain simple variant calls embedded in more complex loci may not be as easy to identify in the ***masterVar*** file format compared to the variation file. For example, a locus that contains a SNP opposite a two-base substitution will be classified as "complex" after the conversion.
- **cgatools** commands that consume the ***masterVar*** file as input may produce results that differ slightly from those produced using the variation file:
- The ordering of the alleles may change, due to the allele ordering rules described in Table 6.
- When using the ***masterVar*** file as input to snpdiff, in some rare cases involving multiple variations in the same locus, the walk algorithm used by the snpdiff tool may result in a no-call, even though it was possible to resolve the base using the variation file.
- Annotation information in the source files pertains to a particular call line. The association between the annotation and a particular call is lost in ***masterVar*** file. However, in most cases, the association is obvious.

Annotation Tools

Most files used as input or output for **cgatools** are simple tab-delimited files that can be interpreted as tables. As such, **cgatools** provides tools that manipulate the files as tables.

join (beta)

The join tool works like a database join to merge the results of two delimited input files. It can be used, for example, to annotate the variant file with your own set of annotations. For example, suppose you have the following file:

```
chromosome      begin        end        region
chr1            13           23         InterestingRegion1
chr2            19           20         InterestingRegion2
```

You can annotate the variant file from Figure 3 as follows:

```
cgatools join –-match=chromosome:chromosome \
    --overlap=begin,end:begin,end \
    --select='a.*,b.region' \
   /path/to/var-tsv.bz2 /path/to/annotations.tsv
```

The result is all the records of the variant file that overlapped with your regions of interest, as shown in Figure 14:

Figure 14: join Example Results

>locus	ploidy	allele	chromosome	begin	end	varType	reference	alleleSeq	totalScore	hapLink	xRef	region
5	2	1	chr1	13	13	ins		A	36			InterestingRegion1
5	2	2	chr1	13	13	ins		A	42			InterestingRegion1
6	2	all	chr1	13	22	ref	=	=				InterestingRegion1
7	2	1	chr1	22	24	del	AT		47	1		InterestingRegion1
7	2	2	chr1	22	24	ref	AT	AT	55	2		InterestingRegion1
16	1	1	chr2	18	20	sub	TT	CG	102			InterestingRegion2

To accomplish this, the join tool first reads the annotations file (file B) into memory. Then it streams the variant file (file A); for each record of file A, it finds the records of file B that match the user-selected columns or that overlap the record. As a consequence of this implementation, file B must fit into memory, but file A may be arbitrarily large. Additionally, the output records are in the same order as they are found in file A.

To recap, the limitation of the join tool is:

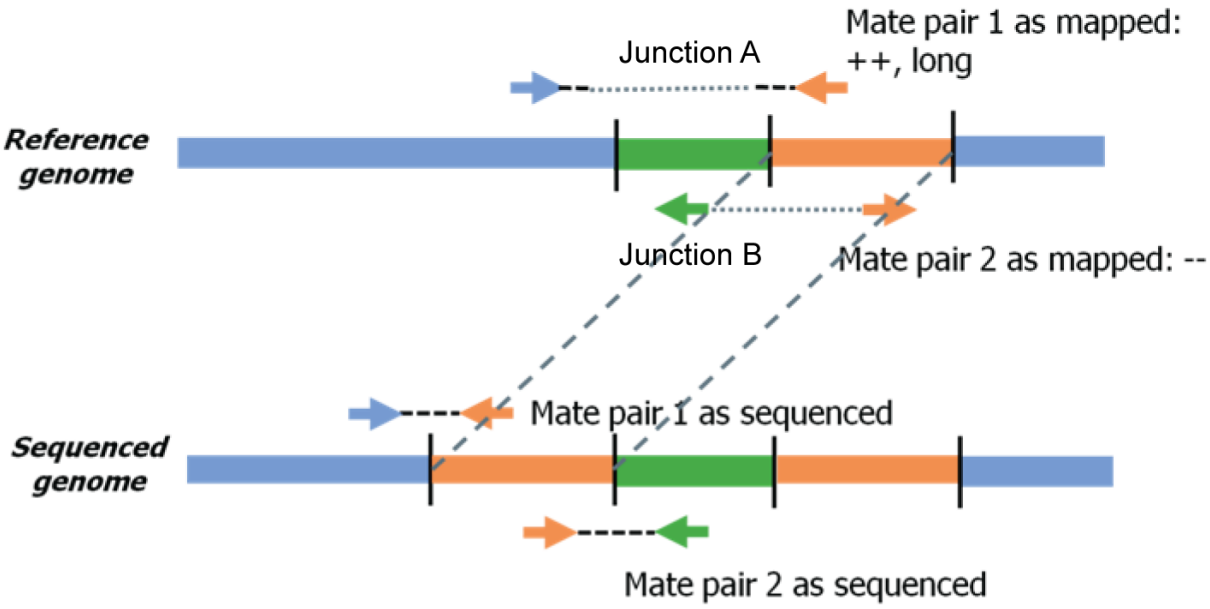
- File B must fit into memory.

junctions2events (beta)

Junctions are defined as regions of the genome being sequenced that are not adjacent or in the same orientation on the reference genome. Structural variation events such as deletions, inversions, and translocations are represented by one or more junctions. Currently the Complete Genomics SV pipeline does not attempt to rationalize sets of junctions into such events. Instead, Complete Genomics provides this functionality in **cgatools**. The junctions2events tool enables researchers to group related junctions and annotate each group with information about the structural rearrangement ("event") that these junctions represent.

To understand the utility of junctions2events, consider a distal duplication event within the same chromosome as shown in Figure 15. This distal duplication event generates two junctions detected with mate pair 1 and mate pair 2.

Figure 15: Distal Duplication Event



In this example, the genomic region represented by the second orange segment of the sequenced genome is duplicated and inserted into a region upstream of the original copy. Mate pair 1 from the sequenced genome maps to the reference genome in the expected orientation, but at a distance that is greater than the expected mate gap size. Thus, Junction A represented by mate pair 1 would have both *LeftStrand* and *RightStrand* as "+" in the junction file. Mate pair 2 from the sequenced genome maps to the reference genome in an anomalous orientation: the right end maps to an earlier position in the reference genome than the left end. Thus, Junction B represented by mate pair 2 would have both strand columns as "-".

When interpreted in isolation, Junction B has the signature of a tandem duplication of the green and orange sequence. Junction A indicates a deletion of the green sequence. Both junctions seem to indicate more disruptive events than the duplication that caused them, particularly if the green sequence is long and contains many genes and the orange sequence is very short. Junction-by-junction interpretation becomes even more misleading if the duplicated orange sequence is inserted into a different chromosome such that each junction may seem to indicate a rearrangement of chromosomal arms.

Missing the detection of a true junction and detecting false junctions can lead to misinterpretation of the underlying event. To limit the chances of misinterpreting the events, junctions2events requires access to the most complete list of junctions available. However, you can specify the shorter list of junctions of interest, so an event will be listed only if it contains at least such junction. For example, you can use

junctiondiff to create the list of putative somatic junctions in a tumor genome, and then interpret these junctions in the context of all the junctions of the tumor using junctions2events:

```
cgatools junctions2events –-junctions=somaticJunctionsTumor.tsv \
    --all-junctions=allJunctionsTumor.tsv \
    --repmask-data rmsk36.tsv.gz \
    --gene-data gene36.tsv.gz \
    --reference build36.crr
```

This approach can also be employed to find the events related to a high-confidence set of junctions in the context of all junctions.

One can deduce structural variation event types from junction data by generating an undirected graph of related junctions and then stepping through the following heuristic process:

- Junctions that are close on at least one side are considered connected. The distance threshold is controlled by the junctions2events `max-related-junction-distance` parameter.
- Connected components with more than two junctions are considered "complex" events.
- For every other junction, junction2events attempts to find a compatible junction in such a way that together they may be interpreted as a distal duplication of contiguous sequence. For example, for junction B in Figure 15, junction2events starts the search from the right position of the junction upstream, until it encounters the right position of the junction A. In general, it scans in the direction away from the break indicated by the junction side. The maximum scan distance is controlled by the `max-pairing-distance` parameter.
  Junctions are considered compatible when their sides can be paired to bound a contiguous piece of sequence from the inside, similar to the orange sequence in Figure 15, while their remaining sides bound a small piece of sequence from the outside.
- Pairs of compatible junctions are considered "inversion" events when the junctions change strand, and the sequence chunk bounded from the inside overlaps to a large degree with the sequence chunk bounded from the outside (one can think of this as the sequence being copied, inverted, and pasted over its old location). For the cases with no significant overlap, the event is classified as "distal-duplication".
- Junction pairs that are connected, but not compatible in the sense described above, are considered "complex" events.
- For isolated junctions, junction2events attempts to find a nearby mobile element that may have caused the junction by copying the adjacent sequence. The search distance is controlled by the `max-distance-to-m-e` parameter, and the list of mobile elements that are known to copy an adjacent sequence may be specified using the `mobile-element-names` parameter.
- Remaining isolated junctions that connect sequence on different chromosomes are not classified any further and are listed as "interchromosomal" events.
- Finally, the isolated junctions that have both sides on the same chromosome are interpreted based on the strands of the junction sides: Junctions with +/+ sides are classified as deletions, -/- as tandem duplications, and strand-inconsistent junctions as probable inversions. In all cases the subject sequence of the event lies between the junction side positions.

In addition to classifying the events by the type, junctions2events uses the list of known genes to annotate the events. Every event (including the "complex" events) is annotated with the list of all potentially disrupted genes; these are the genes that overlap at least one of the junction side positions for any of the junctions that were grouped into the event.

junctions2events generates the list of possible gene fusions for an event as follows:

- When a junction appears to connect two different genes (for example, A and B), it is considered a possible gene fusion (described in the file as "A/B").

- When a junction connects the region upstream of gene C to an intact gene D in a strand-consistent manner, it is annotated using "TSS-UPSTREAM[C]/D" notation; the size of the upstream region that triggers this annotation is set using the regulatory-region-length parameter.
- For the events that may indicate a copy number change of a stretch of sequence (that is, "deletion", "tandem-duplication", and "distal-duplication" events), all the genes that are completely contained in the affected sequence are included.

junctions2events produces two files. The first file contains the list of the original junctions of interest annotated with the event type, the list of related junctions, and the unique ID of the event. The second file contains the list of the events and the corresponding gene-related annotations.

junctions2events requires two external data files. The first file is the list of repeat annotations, described in "generatemasterVar (beta)". The second file is the list of the known gene locations in the genome, derived from the NCBI RefSeq alignment data and reformatted to better fit the **cgatools** conventions. These files for each of the reference builds supported by Complete Genomics may be downloaded from ftp://ftp.completegenomics.com/AnnotationFiles/.