Trans-ABySS 1.4.4 User Manual

Last updated:
October 10, 2012

Written by:
Readman Chiu <rchiu@bcgsc.ca>
Ka Ming Nip <kmnip@bcgsc.ca>

Canada's Michael Smith Genome Sciences Centre, BC Cancer Agency
Vancouver BC Canada V5Z 4S6

Please direct your questions, suggestions, bug reports, and feature requests to our Google Group at: <trans-abyss@googlegroups.com>

Generating Assemblies with ABySS

The input to Trans-ABySS is one or more ABySS assemblies. ABySS can be compiled as described in the README for ABySS (http://www.bcgsc.ca/downloads/abyss/doc/). Should you run into any difficulties in compiling or running ABySS, please contact the ABySS Google Group at:
<abyss-users@googlegroups.com>

Trans-ABySS has been expanded to support 4 types of libraries, each of which has its own assembly protocol:

1. Transcriptome
    i. assemble contigs at multiple k-mer values with reads
2. Genome
    i. assemble the unitigs at 2 k-mer values with reads
    ii. assemble the unitigs at a higher k-mer value than those from (i) with reads and unitigs from (i)
    iii. assemble contigs at the same k-mer value from (ii) with reads and unitigs from (ii)

    Alternatively, you may simply create one paired-end assembly using only one k-mer value. Although simpler, you may risk losing contigs for some genomic events.
3. Targetted Genome
    i. align reads to reference genome
    ii. assemble contigs at multiple k-mer values with reads aligned to region(s) of interest

    This is particularly useful when a subset of your dataset is interesting because the runtime is relatively short compared to assembling the whole genome.
4. Strand-Specific Transcriptome
    i. align reads to reference genome
    ii. divide the reads into batches for (a) plus strand fragments, (b) minus strand fragments, (c) unknown strand fragments according to the orientation of alignments from (i)
    iii. assemble 2 sets of contigs at multiple k-mer values: one set using reads from batches (ii.a) and (ii.c) and another set using reads from batches (ii.b) and (ii.c)

    Currently, TA provides limited support for this protocol. For example, TA only supports input reads in BAM files and the read aligner is limited to BWA only. However, you may also run the regular transcriptome pipeline on your strand-specific transcriptome libraries. We are currently working on a more sophisticated protocol for strand-specific transcriptome libraries.

Installing Trans-ABySS

The TA package consists of the following files and directories:

```
bin/
setup
configs/
input/
annotations/
utilities/
analysis/
sample_dataset/
```

bin/
TA requires the following external software packages:

Software	Version	Purpose with respect to TA
ABySS	1.3.2+	Assembler for multiple-kmer assemblies. FEM. Aligner of reads to merged assembly for genome libraries (`abyss-map`).
bwa	0.6.2	Aligner of reads to merged assembly for transcriptome libraries. Aligner of contigs to reference genome for genome libraries (`bwasw`).
GMAP	2012-07-20 or later	Aligner of contigs to reference genome for transcriptome libraries
SAMtools	0.1.18	Utilities to work with alignments in SAM format.
Python	2.7	Interpreter for all wrappers and analysis modules in TA.
Pysam	0.6	Python interface for SAMtools.
Java	1.6+	Running Picard tools
Picard	1.73+	Convert BAM files to FASTQ files.

It is recommended to put or sym-link the executables of the above software in TA's bin directory. Alternatively, you may specify the paths in the setup file.

setup
The setup file defines all environment variables required by TA. Typically, this command is included in nearly all job scripts created by TA:

```
source /path/to/setup.sh
```

Content of setup:

```
export TRANSABYSS_VERSION=1.4.4
export TRANSABYSS_PATH=/trans-abyss/path
export PYTHONPATH=/python/path:$TRANSABYSS_PATH:$PYTHONPATH
export ABYSSPATH=/abyss/path
export PICARD_DIR=/picard/path
export PATH=.:$TRANSABYSS_PATH/bin:/java/path:$ABYSSPATH:$PYTHONPATH:$PATH
```

Please configure the setup file by giving each environment variable the correct path(s).

configs/

configs/transcriptome.cfg
This file contains the majority of the configurations for the transcriptome pipelines in TA. It has the following major sections:

- `[commands]`
  This section contains the default commands for running each module.

- `[memory]`
  This section contains the default memory request for cluster jobs.

- `[genomes]`
  This section contains the paths to your reference genomes.

- `[tmpmem]`
  This section contains the default space request for temporary directories used in cluster jobs.

- `[java]`
  This section contains the java options used for each java package.

TA processes data on a per-library basis. Each library must belong to only one project, but each project is expected to have multiple libraries. In `transcriptome.cfg`, a project should be set up a new section. Each project must have a working directory and a reference genome, which are specified in `topdir` and `reference` respectively. For example:

```
[your_project_name]
topdir: /your/transabyss/working/directory/for/this/project
reference: /name/of/the/reference/genome/configured/in/"[genomes]"/section
abyss-rmdups-iterative-cmd: -n LIB -i INPUT_DIR -o OUTPUT_DIR INDEL_ONLY -t 12
abyss-rmdups-iterative-mem: 3G,12
bwa_sam-tmpmem: 60G
samtofastq.jar-java: java -XX:-UseGCOverheadLimit –Xmx10g
```

You may override the defaults for processing each project with the postfixes `-cmd`, `-mem`, `-tmpmem`, `-java` for the sections for `command`, `memory`, `tmpmem`, and `java` respectively. As shown here, `abyss-rmdups-iterative` was configured use 12 threads and run on 12 CPUs and allocate 3G for each CPU (to a total of 36G available memory).

configs/genome.cfg
This configuration file serves the same purpose as `transcriptome.cfg` except it is used for the genome pipelines.

configs/model_matcher.cfg

This configuration file specifies the gene model files that are used by the module `model_matcher.py` for contig-transcript mapping.

Content of model_matcher.cfg:

```
[hg19]
k: knownGene_ref.txt
e: ensGene_ref.txt
r: refGene.txt
a: acembly_ref.txt
order: k,e,r,a
```

You should set up one section for each reference genome you use in TA. The gene model files referenced in each section are expected to be found in the `annotations` directory. See “annotations" for instructions on downloading annotation files. Each gene model file is assigned an alias for quick referencing. For example, `e` represents the Ensembl gene model file while `r` represents the Refseq gene model file. These aliases should be arranged in a comma-separated list in the `order` field from highest priority to lowest priority. Priority set here will be used in breaking ties when a contig can be mapped to genes from multiple models.

configs/job_script.cfg
This configuration file contains the configurations for job submissions.

Content of job_script.cfg:

```
local: gsc_local.txt
cluster_basic: gsc_sge_basic.txt
cluster_parallel: gsc_sge_parallel.txt
cluster_basic_array: gsc_sge_basic_array.txt
cluster_parallel_array: gsc_sge_parallel_array.txt
predecessors_list_delimiter: ,
run_local_job_command: bash
submit_cluster_job_command: qsub
submit_cluster_job_return_string: Your job ${JOBID} .* has been submitted
submit_cluster_array_job_return_string: Your job-array ${JOBID}\..* has been
submitted
```

- `local` defines the template for local jobs.
- `cluster_basic` defines the template for basic (single CPU) cluster jobs.
- `cluster_parallel` defines the template for parallel (multiple CPUs) cluster jobs.
- `cluster_basic_array` defines the template for basic array cluster jobs.
- `cluster_parallel_array` defines the template for parallel array cluster jobs.
- `predecessors_list_delimiter` defines the delimiter for the list of predecessors for each job.
- `run_local_job_command` defines the command to run local jobs.
- `submit_cluster_job_command` defines the command to submit batch jobs.
- `submit_cluster_job_return_string` defines the string returned when batch jobs are submitted. This string is used for retrieving the job id from a batch job submitted with the `submit_cluster_job_command`. `${JOBID}` corresponds to the

part the string representing the job id. You may use Python's regular expressions (http://docs.python.org/library/re.html) in this string.

- `submit_cluster_array_job_return_string` defines the string returned when array jobs are submitted. Its purpose is same as `submit_cluster_job_return_string`.

configs/templates/
We attempt to use job script templates to simplify the process of setting up batch job submission of TA jobs in different HPC environment. Although our templates were written to work with the Sun Grid Engine of our cluster, you can create your own templates for your HPC environment.

The following variables in templates would be replaced with the appropriate values when job scripts are generated:

- `${JOB_NAME}` is the name of the job.
- `${WORKING_DIR}` is the working directory of the job. The stdout and stderr logs would be place in this directory.
- `${PREDECESSORS}` is the list of predecessors' job id. Note that `predecessors_list_delimiter` from jobs_script.cfg would be used here.
- `${MEM}` is the amount memory to request for the job.
- `${QUEUE}` is the list of cluster queues for the job.
- `${THREADS}` is the number of CPUs for the parallel job.
- `${FIRST_TASK_ID}` is the first task id for the array job.
- `${LAST_TASK_ID}` is the last task id for the array job.
- `${TMPMEM}` is the amount of disk space to request for the job.
- `${SETUP_PATHS}` would be replaced with the command, `source /path/to/setup`
- `${CONTENT}` is the commands to be run in the job. This variable is mandatory for all templates.

The following variables must be defined properly:

- `$TMPDIR` is the prefix for temporary files. Typically, the scheduler of your HPC cluster should configure it automatically for each job. Otherwise, please configure it in the template to use the cluster node's local temporary directory along with a unique prefix, ie. `TMPDIR=/tmp/$JOB_ID.$TASK_ID.$QUEUE.`
- `$TA_JOBID` is the task id of the array job. This variable is mandatory for all array jobs in TA. You should link this variable with the task id of the job, ie. `TA_JOBID=$SGE_TASK_ID`

input/
An input file defines the set of libraries to process with TA. There are no restrictions on the name and location of an input file. This is the format of an input file:

```
LIBRARY ASSEMBLY_DIR PROJECT READLENGTH LIBRARYTYPE METALIBRARY
```

- `LIBRARY` is the name of the library
- `ASSEMBLY_DIR` is the path to the directory containing the library's multiple-kmer assemblies
- `PROJECT` is the project name of the library
- `READLENGTH` is the smallest read length for the library
- `LIBRARYTYPE` is the type of the library (ie. transcriptome, genome, targetted_genome, plus_strand, minus_strand)
- `METALIBRARY` is the name for strand-specific transcriptome library

Not all fields are required.

If either option `-T` or `-G` or `-tG` is used, `LIBRARYTYPE` is not required, ie:

```
LIBRARY ASSEMBLY_DIR PROJECT READLENGTH
```

Otherwise, `LIBRARYTYPE` is required, ie:

```
LIBRARY ASSEMBLY_DIR PROJECT READLENGTH transcriptome
LIBRARY ASSEMBLY_DIR PROJECT READLENGTH genome
LIBRARY ASSEMBLY_DIR PROJECT READLENGTH targetted_genome
```

Each strand-specific transcriptome library consists of 2 lines, one line for the plus strand and another line for the minus strand. For example:

```
LIB001_plus /assembly/dir/plus MyProject 100 plus_strand LIB001
LIB001_minus /assembly/dir/minus MyProject 100 minus_strand LIB001
```

annotations/

Analysis modules of TA require comparisons to a reference genome and gene annotation files. TA organizes annotation files by genome under the annotations folder, for example:

```
annotations/
|-- hg19/
|   |-- genome.2bit
|   |-- splice_motifs.fa
|   `-- ...
`-- shared/
    `-- splice_motifs.txt
```

TA mainly uses the annotation files available from the UCSC genome browser:

```
ftp://hgdownload.cse.ucsc.edu/goldenPath/<genome>/database
```

A list of files required (`<genome>_annot.txt`) and a downloading script (`<genome>_annot.sh`) available for the genomes hg18, hg19, and mm9 are provided in the annotations folder for executing the `wget` downloads and running the following processing steps. This is an example usage of setting up the hg19 annotation files:

```
cd <TA_DIR>/annotations
./hg19_annot.sh hg19/ hg19_annot.txt hg19 <TA_DIR>
```

where:
`hg19/` is the destination folder
`hg19` is the name of the genome

Note that a `snp1xx.txt.gz` is included in all genome's file lists. This dbSNP file is used to annotate the snv/indel events detected. To speed up this annotation process, the dbSNP annotation should be split by chromosome with this command:

```
split_dbsnp.sh ./split_dbsnp.sh <TA_DIR>/annotations/<genome>/snp1xx.txt
<TA_DIR>
```

Note that `dgv.txt.gz` is also included. This is the DGV database flat file used to annotate fusions and large scale rearrangement events detected.

The user is expected to have the single reference genome sequence FASTA file available on the cluster for contig alignments. For example, the reference genome hg19 can be downloaded from:

ftp://ftp.ncbi.nih.gov/genbank/genomes/Eukaryotes/vertebrates_mammals/Homo_sapiens/GRCh37/special_requests/

After that, put the path to the downloaded reference FASTA file in configs/transcriptome.cfg under `[genomes]`, ie.

```
[genomes]
hg19: /path/to/your/hg19/fasta_file/here
```

A `2bit` version of the same genome sequence is expected to be present in the genome folder for quick random access to the reference sequence. A `<genome>.2bit` file can be generated from the utility `faToTwoBit` available from: http://users.soe.ucsc.edu/~kent/src

Running Trans-ABySS

All stages in TA are initiated with the Python driver script “trans-abyss.py”. To show all available options in “trans-abyss.py”, run this command:

```
python trans-abyss.py -h
```

Typically, each stage can be run like so:

```
python trans-abyss.py -<stage> -p <project> -l <library> -a <assembly dir> \
-L <read length> -<sample type>
```

Alternatively, an input file can be used:

```
python trans-abyss.py -<stage> -n <input file>
```

See Figure 1 for the workflow of the 10 stages in TA.

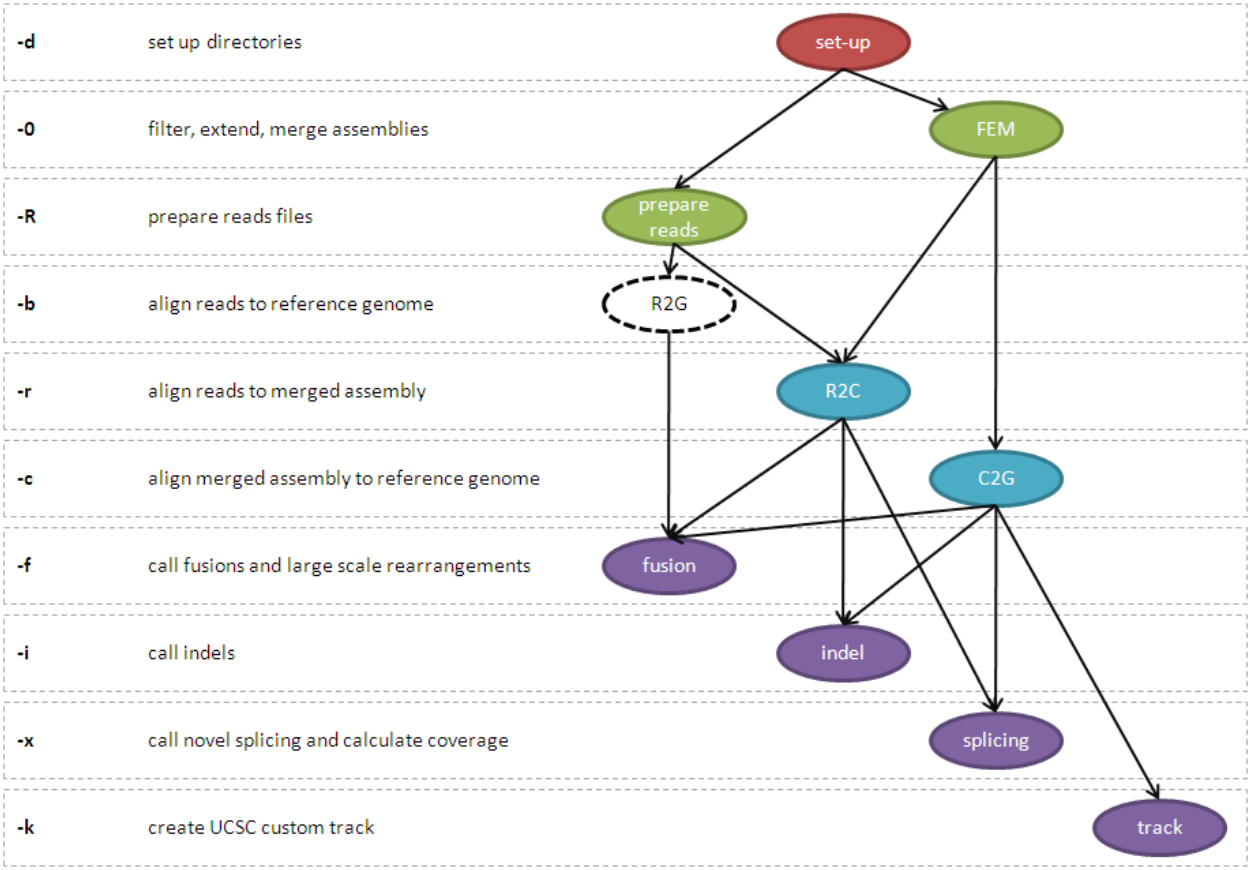

Figure 1. The workflow of Trans-ABySS 1.4.4.
Stages having the same color in this figure can be done in parallel.

-d Set up directories

TA sets up the output directories and makes sym-links to your input ABySS assemblies from the assembly directory specified with `-a` or within your input file.

Example output:

```
assembly/
|-- in
|-- k62 -> ../../../../ABySS/SampleProject/abyss-1.3.2/sim0003/k62
|-- ...
`-- k74 -> ../../../../ABySS/SampleProject/abyss-1.3.2/sim0003/k74
```

`in` is a text file listing all input read files; it is an exact copy of the one in the assembly directory.

-0 Filter, extend, merge assemblies

This stage is frequently referred to as FEM and it was part of Stage 0 in TA 1.3.*.
Transcriptome libraries:

- junction contigs and indel bubbles are extended with `abyss-junction`
- short contigs and short islands are removed with `abyss-filtergraph`

Genome libraries:

- only indel bubbles are extended with `abyss-junction`
- no contigs are filtered by length

If there is no reference genome for your library, you may stop after this stage.

Example output:

```
filter/
|-- cluster/
|   `-- ...
|-- k62/
|   `-- ...
|-- ...
`-- k74/
    |-- sim0003-b.fa
    |-- sim0003-contigs.fa
    |-- sim0003-f.fa
    |-- sim0003-j.fa
    |-- sim0003-nb.path
    `-- sim0003.74.abyss-ta-filter.COMPLETE
merge/
|-- cluster/
|   `-- ...
|-- sim0003-contigs.fa
|-- sim0003.merge.abyss-rmdups-iterative.COMPLETE
`-- stats.txt
```

`filter/k*/*-b.fa` contains extended indel bubbles.
`filter/k*/*-f.fa` contains contigs passing the length filter.
`filter/k*/*-j.fa` contains extended junction contigs.
`filter/k*/*-contigs.fa` is the concatenate of `*-b.fa`, `*-f.fa`, and `*-j.fa`.
`merge/*-contigs.fa` is the merged assembly.

`merge/stats.txt` contains the statistics for the ABySS assemblies and the merged assembly.

-R Prepare reads

If your read files are FASTQ files, you may skip this stage.

Example output:

```
reads_to_contigs/
|-- cluster/
|   `-- ...
|-- reads/
|   |-- 1.reads_1_export.fq
|   `-- 2.reads_2_export.fq
`-- sim0003.in
```

`*.in` is a text file listing all input read files.

-b Align reads to reference genome

TA does not do anything for this stage out of the box. This stage is meant to be done on your own.

Transcriptome libraries:

- You must align reads to the genome and exon-exon junction reference with JAGuaR (or other gap-aligner such as GSNAP).

Genome libraries:

- You may align reads to the reference genome with any short-read aligner such as BWA that outputs in SAM format.

No matter which ever route you take, you must create one indexed BAM file in the "reads_to_genome" directory. This indexed BAM file is required in stage `-f` for both transcriptome and genome libraries, and in stage `-i` for genome libraries only.

-r Align reads to merged assembly

Transcriptome libraries:

- TA is defaulted to use BWA to align reads to the merged assembly.

Genome libraries:

- TA is defaulted to use `abyss-map` to align reads to merged assembly.

Example output:

```
reads_to_contigs/
|-- cluster/
|   `-- ...
|-- reads/
|   `-- ...
|-- sim0003.in
|-- sim0003-contigs.fa -> ../merge/sim0003-contigs.fa
|-- sim0003-contigs.fa.fai
|-- sim0003-contigs.fa.amb
|-- sim0003-contigs.fa.ann
|-- sim0003-contigs.fa.bwt
|-- sim0003-contigs.fa.pac
```

```
|-- sim0003-contigs.fa.sa
|-- 1.reads_1_export.fq.sai
|-- 2.reads_2_export.fq.sai
|-- 1.reads_1_export.fq.sorted.bam
|-- 1.reads_1_export.fq.sorted.bam.bai
|-- sim0003-contigs.bam -> 1.reads_1_export.fq.sorted.bam
`-- sim0003-contigs.bam.bai -> 1.reads_1_export.fq.sorted.bam.bai
```

`*-contigs.fa.*` are the various index files for the merged assembly.
`*.sai` are the suffix array index files.
`*.sorted.bam` is the BAM file for each pair of read files.
`*.bai` are the BAM indexes.
When there is only one `*.sorted.bam`, `*-contigs.bam` is a sym-link to that `*.sorted.bam`. Otherwise, `*-contigs.bam` is the merged BAM file of all `*.sorted.bam`.

-c Align merged assembly to reference genome

The merged assembly is split into multiple FASTA files, where each contains at most 5000 contigs by default. Then, each FASTA file is aligned to the reference genome.
Transcriptome libraries:

- TA uses GMAP to align the merged assembly to the reference genome.

Genome libraries:

- TA uses BWA-SW to align the merged assembly to the reference genome.

Example output:

```
contigs_to_genome/
`-- sim0003-contigs/
    |-- cluster/
    |   `-- ...
    |-- input/
    |   `-- seq.1.fa
    `-- output/
        `-- seq.1.sam
```

`seq.*.fa` are the split-up FASTA files.
`seq.*.sam` are the alignment output files.

The output files are in SAM format, but TA also accepts alignments in PSL format, such as those from BLAT.

-k Create UCSC custom track of the merged assembly

This stage is only applicable to transcriptome libraries.

Example output:

```
tracks/
|-- cluster/
|   `-- ...
`-- sim0003.merged.best.unique.m90.gmap.psl.gz
```

`*.psl.gz` is the track that can be uploaded to the UCSC genome browser.

-f Call fusion events and other large scale rearrangement events

Example output:

```
fusions/
|-- cluster/
|   |-- 1/
|   |   |-- LOG
|   |   `-- fusions.tsv
|   `-- ...
|-- fusions.tsv
|-- fusions_filtered.tsv
|-- local.tsv
`-- LOG
```

See the next section for the description and format of output files.

-i Call indels

Example output:

```
indels/
|-- cluster/
|   |-- 1/
|   |   |-- LOG
|   |   `-- events.tsv
|   `-- ...
|-- events.tsv
|-- events_concat.tsv
|-- events_exons.tsv
|-- events_exons_novel.tsv
|-- events_filtered.tsv
|-- events_filtered_novel.tsv
|-- filter_debug.tsv
`-- LOG
```

See the next section for the description and format of output files.

-x Call novel splicing events and calculate coverage of known isoforms

This stage is only applicable to transcriptome libraries.

Example output:

```
splicing/
|-- cluster/
|   |-- 1/
|   |   |-- LOG
|   |   |-- coverage.tsv
|   |   |-- events.tsv
|   |   |-- log.txt
|   |   `-- mapping.tsv
|   `-- ...
|-- coverage.tsv
|-- events.tsv
|-- events_filtered.tsv
|-- events_summary.tsv
```

```
`-- mapping.tsv
```

See the next section for the description and format of output files.

Output format of analyses results files
Fusion (fusion.py)

Output	Description
fusions.tsv	unfiltered fusion events captured by split contig alignments
fusions_filtered.tsv	filtered events, where: num_read_pairs >= *--min_read_pairs (default: 4)* AND <= *--max_read_pairs (default: 2000).* *<num_read_pairs>* = *<flanking_pairs>* + minimum*(<breakpoint_pairs>)* <spanning_reads> >= *--min_span_reads (default: 2)*
local.tsv	“local” events, when: alignment target regions overlap, or alignment target regions overlap same gene, or transcripts mapped by target regions overlap
LOG	run log recording command run and parameters used

Content of “fusion_filtered.tsv”:

Column	Name	Description
1	id	event ID. Each line represents an event captured by an individual contig. Identical events will be linked by the first number of of 'id'. Example: 2.1, 2.2, 2.3 represent the same event captured by 3 different contigs. Events are grouped by type of rearrangement (<rearrangement>) and breakpoint (<breakpoint>). Reciprocal (<reciprocal>) events are indicated by 'a' and 'b' attached at the end. Example: '89a' and '89b' are reciprocal events.
2	contig	contig ID
3	contig size	size(or length) of contig (<contig>)
4	genomic_regions	the 2 genomic regions the contig aligns to. Format: chromosomeA:start1-end1,chromosomeB:start2-end2 (chromosome names are the same as in the FASTA file used for contig alignments). Order of regions is sorted by the chromosome names.
5	contig_regions	the corresponding contig coordinates of the 2 genomic regions. Format: start1-end1,start2-end2 (regions in the same order of genomic regions)
6	strands	relative orientation of the 2 alignments in relation to the genome. Format:[+|-],[+|-]
7	flanking_pairs	number of read pairs from reads-to-genome alignments with both mates flanking the breakpoint, both pointing towards each other.

8	breakpoint_pairs	number of read pairs from reads-to-genome alignments with one mate spanning the breakpoint and the other mate flanking it, both pointing towards each other. This is useful for read-support when reads lengths are long compared to fragment size. Pairs up- and down-stream of the breakpoint are reported in a 2-member tuple.
9	spanning_reads	number of reads spanning junction from reads-to-contigs alignments
10	rearrangement	underlying genome rearrangement deduced by relative contig alignment orientations. Can be “translocation”, “deletion”, “inversion”, or “duplication”
11	breakpoint	junction breakpoint. Format: chrA:coordinate1| chrB:coordinate2. Chromosome names are in same format as in FASTA file used for contig alignments
12	size	size (bp) of the event
13	genes	gene1,gene2 of the genes involved in the fusion. 'gene1' correspond to the first coordinate in 'breakpoint'; 'gene1' the second
14	transcripts	transcript1,transcript2 of the transcripts picked for the fusion event
15	senses	'+' indicates the contig aligns in the same direction of the gene strand, '-' indicates the contig aligns in opposite direction of the gene strand
16	exons/introns	exon/intron number ([exon|intron]N), '5utr', or '3utr' where the breakpoints lie
17	exon_bounds	whether breakpoint is within exon boundaries ('yes') or not ('no')
18	reciprocal	breakpoint coordinate (<breakpoint>) of reciprocal event captured in same library
19	descriptor	'conventional' nomenclature of rearrangement e.g. t(11;17)(q12.2;q25.1)
20	orientations	indicates which parts of the chromosomes are joined together. 'L' == chromosome upstream of breakpoint coordinate; 'R' == chromosome downstream of breakpoint coordinate
21	5' gene	gene name of the 5' transcript in 'sense_fusion' cases where the 5' and 3' transcripts can be unambiguously discerned ('-' otherwise)
22	3' gene	gene name of the 3' transcript in 'sense_fusion' cases where the 5' and 3' transcripts can be unambiguously discerned ('-' otherwise)
23	5' exon	exon number of the 5' gene where the breakpoint lies. If breakpoint lies in an intron, the downstream exon number will be reported. If breakpoint lies in an UTR, '5utr' or '3utr' will be

		indicated
24	3' exon	exon number of the 3' gene where the breakpoint lies. If breakpoint lies in an intron, the downstream exon number will be reported. If breakpoint lies in an UTR, '5utr' or '3utr' will be indicated
25	frame	
26	alignment_params	alignment details, mainly for debug purpose. Format: TO:,CO:,CC:,I1:,I2:,AF1:,AF2:, where TO : target overlap fraction = overlap(target_region1, target_region2)/ total_target_region_length CO : contig overlap fraction = overlap(query_region1, query_region2)/ total_query_region_length CC : contig coverage = (match_length1 + match_length2 – overlap) /query length I1 : percent identity of alignment 1 I2 : percent identity of alignment 2 AF1 = alignment fraction of alignment 1: match_length1/query_length AF2 = alignment fraction of alignment 2
27	type	can be: “sense_fusion” - if the breakpoints reside in 2 transcripts, and the orientations of the contig relative to the 2 transcripts are the same “antisense_fusion” - if the breakpoints reside in 2 transcripts, and the orientations of the contig relative to the 2 transcripts are NOT the same “LSR” - any fusion event not of the above types
28	dbsnp	dbSNP entries for deletion events that are already annotated in dbSNP
29	dgv	DGV entries for deletion and inversion events that are already annotated in DGV

SNV/INDEL (snv_caller.py)

Output	Description
events.tsv	unfiltered snv/indel events captured by gapped contig alignments
events_filtered.tsv	filtered events, <event_reads> >= *--min_reads_contigs (default: 3)*
events_filtered_novel.tsv	filtered events not annotated in dbSNP
events_exons.tsv	filtered, non-synonymous events residing in gene exons
events_exons_novel.tsv	filtered, non-synonymous events residing in gene exons not annotated in dbSNP
LOG	run log recording command run and parameters used

Content of “events_filtered.tsv”:

Column	Name	Description
1	id	event ID. Each line represents an event captured by an individual contig. Identical events will be linked by the first number of of 'id'. Example: 2.1, 2.2, 2.3 represent the same event captured by 3 different contigs. Events are grouped by event type (<type>), coordinate (<chr> + <chr_start> + <chr_end>), and the alternative allele (<alt>)
2	type	event type. Can be 'snv', 'ins', 'del', 'inv'
3	chr	chromosome name (as in the chromosome name in the FASTA file used for contig alignments)
4	chr_start	chromosome start coordinate. If <type> == 'ins', <chr_start> = coordinate immediately upstream of insertion. If <type> == 'del' or 'inv', <chr_start> = first base of deletion or inversion. If <type> == 'snv', <chr_start> == <chr_end>, the base of substitution
5	chr_end	chromosome end coordinate. If <type> == 'ins' or 'snv', <chr_end> = <chr_start>. If <type> == 'del' or 'inv', <chr_end> = last base of deletion or inversion
6	ctg	contig ID
7	ctg_len	length of contig (<ctg>) that captures event
8	ctg_start	contig start coordinate. If <type> == 'ins', <ctg_start> = coordinate immediately upstream of insertion. If <type> == 'del' or 'inv', <ctg_start> = first base of deletion or deletion. If <type> == 'snv', <ctg_start> == <ctg_end>, the base of substitution
9	ctg_end	contig end coordinate. If <type> == 'ins' or 'snv',

		<ctg_end> = <ctg_start>. If <type> == “del” or 'inv', <ctg_end> = last base of deletion or inversion
10	len	length (or size) of event
11	ref	reference allele. If <type> == 'ins', <ref> = 'na'
12	alt	alternative allele. If <type> == 'del', <ref> = 'na'
13	event_reads	total number of reads spanning *event* from reads-to-contig alignment
14	contig_reads	number of reads spanning event in *contig* (<ctg>) from reads-to-contig alignment
15	genome_reads	total number of reads spanning *event* from reads-to-genome alignment
16	gene	gene in affected locus. Format: gene:transcript:[intron|exon]number|effect on open reading frame (see below) If the event size is bigger than 1, the output is a pairing of the above format on both coordinates, i.e.: geneA:transcriptA:[intron|exon]numberA| geneB:transcriptB:[intron|exon]numberB|effect on open reading frame, where A and B may be the same (small event within the same transcript) or different (bigger events)
17	repeat-length	length of repeat in alternative allele, e.g. AAAA = 4, CAGCAG = 2
18	ctg_strand	query strand of alignment in relation to reference
19	from_end	shortest distance (bases) of event to end of contig.
20	confirm_contig_region	contig coordinate range (start, end) used for checking event support in reads-to-contig alignments
21	within_simple_repeats	overlap with simple repeats. Name of tandem repeat reported if overlap is True e.g. TRF_SimpleTandemRepeat_CATC. '-' if overlap is False.
22	repeatmasker	overlap with RepeatMasker annotations. Type of repeat reported if overlap is True e.g. AluSx, LTR47A . '-' if overlap is False.
23	within_segdup	overlap with segmental duplication (segdup). Chromosome:Start_coordinate of segdup partner reported if overlap is True, e.g. chr1:17048246. '-' if overlap is False.
24	at_least_1_read_opposite	if at least 1 supporting read is aligned in opposite orientation to rest of supporting reads. Can be “true” or “false”
25	dbsnp	dbSNP entries if event is already annotated in dbSNP e.g. rs12028735,rs71510514

Novel Splicing (model_matcher.py)

Output	Description
events.tsv	unfiltered novel splicing events not observed in annotations specified in *model_matcher.cfg*
events_filtered.tsv	filtered events. See below for filtering criteria.
events_summary.tsv	tally of filtered events by <type>
coverage.tsv	transcript coverage
mapping.tsv	mapping of contig to annotated transcripts
log.txt	detailed block-by-block mapping of alignments to exons
LOG	run log recording command run and parameters used

Contents of “events_filtered.tsv”:

Column	Name	Description
1	id	event ID. Each line represents an event captured by an individual contig. Identical events will be linked by the first number of <id>. Example: 2.1, 2.2, 2.3 represent the same event captured by 3 different contigs. Events are grouped by event type (<type>) and genome coordinate <genome_coord>.
2	type	event type. Can be: AS3: novel 3' splice site AS5: novel 5' splice site AS53: novel 5' and 3' splice site (on the same alignment block) novel_exon: novel exon novel_intron: novel intron novel_transcript: novel transcript, when contig cannot be mapped to any known transcript novel_utr: novel UTR, when novel alignment blocks exist beyond annotated 5' and 3' exons of mapped transcript retained_intron: retained intron skipped_exon: skipped exon
3	contig	contig ID
4	transcript	transcript name
5	gene	gene name
6	exons	exon number(s), relative to transcript strand, start from 1
7	align_blocks	alignment block numbers, counted in ascending order of coordinate, start from 1.
8	geome_coord	genome coordinate of novel block. Format: chromosome:start-end
9	contig_coord	contig coordinate of novel block
10	splice	splice sites adjacent to the novel junction. E.g. gtAG(U2/U12),

		where 'AG' (in capitals) is the novel splice donor/acceptor created by the novel sequence, 'gt' is the partner splice donor/acceptor; 'U2/U12' is name of the splice motif. If a novel block creates two novel splice sites (e.g. a 'skipped_exon' event), 2 splice sites will be reported e.g. gtAG(U2/U12),GTag(U2/U12)
11	multi_3	only applicable to 'retained_intron' events. “True” if the size of the intron retained is a multiple of 3, i.e. retained open reading frame
12	size	size of novel block. Only applicable to AS53, novel_exon, novel_intron, novel_transcript, and novel_utr
13	orf	effect on open reading frame. See below.
14	spanning_reads	number of reads spanning novel junction, gathered from reads-to-contig alignments
15	coverage	number of reads spanning novel block. Applies to 'AS53', 'novel_exon', 'novel_transcript', 'novel_utr', and 'retained_intron'

Contents of “coverage.tsv”:

Column	Name	Description
1	feature	'gene' or 'transcript'
2	model	single-letter initial of gene model used for coverage calculation. The initial is specified in the configuration file 'model_matcher.cfg'
3	transcript	transcript name
4	gene	gene name
5	exon	exon number (currently not relevant as exon-level coverage is not reported)
6	strand	transcript strand
7	coord	coordinate of <feature> chromosome:start-end
8	feature_size	Best contig mapped to <transcript> in terms of bases covered
9	bases_reconstructed	number of exonic bases reconstructed by all contig mapped to <feature>
10	reconstruction	fraction of exonic bases reconstructed by all contig mapped to <feature>
11	num_reads	number of reads spanning feature from reads-to-contigs alignments (currently not reported)
12	bases_reads	total number bases spanning feature from reads-to-contigs alignments (currently not reported)
13	depth	<bases_reads> / <num_reads>

14	contigs	list of contigs mapped to <feature>
15	num_contigs	number of contigs mapped to <feature>
16	best_contig	ID of contig that reconstructs <feature> best
17	best_contig_reconstruction	fraction of <feature> reconstructed by <best_contig>
18	align_blocks	list of alignment blocks used for reconstructing <feature> - only reported in intermediate batch outputs before filtering stage
19	exons	list of exons reconstructed - only reported in intermediate batch outputs before filtering stage

Contents of “mapping.tsv”:

Column	Name	Description
1	contig	contig ID
2	contig_len	length or size of <contig>
3	coord	genome alignment coordinate of <contig>
4	model	single-letter initial of gene model used for coverage calculation. The initial is specified in the configuration file 'model_matcher.cfg'
5	transcript	name of mapped transcript
6	gene	gene name of <transcript>
7	strand	strand of <transcript>
8	coding	'CODING' or 'NONCODING'. 'NONCODING' if start and end coordinate of <transcript> are the same in annotation file
9	intronic	intron number if <contig> is mapped to introns; '-' if otherwise
10	num_align_blocks	number of alignment blocks
11	num_exons	number of exons of <transcript>
12	num_matched_blocks	number of alignment blocks matched to exons. Internal blocks are considered 'matched' when both edges align, terminal blocks are considered 'matched' when internal edges align
13	matched_blocks	list of matched alignment blocks. Blocks are numbered from left to right
14	matched_exons	list of matched exons. Exons are numbered in reference to transcript strand (<strand>)
15	score	number of edges matched (terminal edges count if corresponding internal edges match)

16	coverage	fraction of exonic bases of <trancript> covered/reconstructed by <contig>
17	align_blocks	genome coordinates (start, end) of all alignment blocks with each block separated by ';'

Miscellaneous

Open Reading Frame Effect Descriptors

Throughout the output from TA, a standard nomenclature (used, for example, by the Human Genome Variation Society) is used to denote the effect of an event on a gene at the protein level. The following table describes the changes with an example notation and explanation:

Change	Example
frameshift	A245Sfs (Alanine 235 becomes Serine followed by a frameshift)
deletion	V422_S431del (deletion from Valine 433 to Serine 431)
insertion	Q484_I485insVA (insertion of Valine and Alanine in between Glutamine 484 and Isoleucine 485)
indel	S293_Y294insKS (Serine 293 to Tyrosine 294 becomes Lysine and Serine)
synon	Synonymous/silent
substitution	T327S (Threonine 327 to Serine)