Trans-ABySS v1.3.5: User Manual

September, 2012

Prepared by Readman Chiu, Ka Ming Nip
Contact: rchiu@bcgsc.ca, kmnip@bcgsc.ca
On behalf of: Tony Raymond, Shaun Jackman, Karen Mungall, Inanc Birol

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

Table of Contents

1. Generating Transcriptome Assemblies with ABySS

[1.1] Installing ABySS

The input to Trans-ABySS (TA) are one or more ABySS (1.3.2 or above) assemblies. This section describes only one of the many ways to generate transcriptome assemblies with ABySS 1.3.2.

ABySS can be compiled as described in its README. Should you run into any difficulties in compiling or running ABySS, please contact the ABySS Google Group:
abyss-users@googlegroups.com

[1.2] Choosing k-mer values for your assemblies

Transcriptome (RNAseq) samples are composed of transcripts with a wide range of expression levels. Because it is observed that reconstruction of transcripts tend to be performed with various degrees of completeness at different k-mer values, TA takes the approach of generating assemblies using a range of k-mer values and then merging the different k-assemblies into a single meta-assembly. The choice of k-mer sizes depend on the read length of an RNAseq library and we suggest the following k-mer sizes for the given read-lengths:

read length (bp)	k-mer sizes	total number of assemblies
50	26, 28, 30, … , 46, 48, 50	13
75	38, 40, 42, … , 70, 72, 74	19
100	52, 54, 56, … , 92, 94, 96	23

Please note that the above is just a guideline based on our experience with the given read-lengths. The choice is basically a compromise between performance (level of reconstruction) and practicality (time, computing resources, etc). Users can experiment with their own k-mer sizes to suit their sample characteristics and computing resources.

[1.3] Running ABySS

Before running ABySS to generate the assemblies, set up a directory to store your assemblies for each k-mer size:

```
mkdir $name
```

where `$name` is the library name.

In this newly created directory, make a text file called `$name.in` that lists the paths to the input reads, one read file per line. The input reads can be `.bam`, `.fq.gz` (`.fastq.gz`), Illumina export/qseq files, or any other formats that ABySS can read. This text file will also be required in TA Stage 0. Please read Section "[3.1] Stage 0″ for more details.

Since you will be generating paired-end assemblies, the list of reads files in `$name.in` should be

ordered like so:

```
<first pair, read file 1>
<first pair, read file 2>
<second pair, read file 1>
<second pair, read file 2>
<third pair, read file 1>
<third pair, read file 2>
```

Please read Sections "[3.1] Stage 0" (b) and "[3.3] Stage 2" for more details.

Here is an example of `$name.in` and the name of the reads files:

```
/path/to/reads/s_5_1_concat_qseq.txt
/path/to/reads/s_5_2_concat_qseq.txt
/path/to/reads/s_6_1_concat_qseq.txt
/path/to/reads/s_6_3_concat_qseq.txt
```

This is a sample command to generate one paired-end assembly with ABySS:

```
cd $name && mkdir k$k && cd k$k && \
exec abyss-pe E=0 n=5 v=-v k=$k \
      name=$name \
      in=`<../$name.in` \
      OVERLAP_OPTIONS='--no-scaffold' \
      SIMPLEGRAPH_OPTIONS='--no-scaffold' \
      MERGEPATHS_OPTIONS='--greedy' \
      mp=''
```

where `$k` is the k-mer size, and `$name` is the library name. The assembly output files would be generated in the directory called `k$k`, which is strictly required for TA. You can vary the options `n`, `E`, `c`, `s`, etc. to generate your assemblies, but please keep the `'--no-scaffold'` option for `OVERLAP_OPTIONS` and `SIMPLEGRAPH_OPTIONS` because TA does not deal with scaffolds.

Please execute the above command (in a cluster job script if necessary) for all k-mer sizes in the same directory so you would have one directory per k-mer size. For example, these are the directories and files created if the library `$name` has 50-bp reads:

```
$name/
      k26/
      k28/
      ...
      k48/
      k50/
      $name.in
```

After the assemblies have finished, please check for error messages in the log files and make sure that these files exist in each k-directory:

`$name-1.fa`
`$name-2.adj`
`$name-3.dist`
`$name-3.fa`
`$name-4.fa` (can be empty)
`$name-5.fa` (can be empty)
`$name-5.adj`
`$name-5.path`
`$name-6.fa`
`$name-contigs.fa` (a symbolic link to `$name-6.fa`)
`$name-indel.fa`

If there are no missing output files or error messages, you have generated the ABySS assemblies needed for TA. If you decided to generate the assemblies using your own methods or pipelines, then you must rename any files and/or construct a directory that looks like so:

```
assemblies_parent_directory/
      $name.in
      k26/
            $name-1.fa
            $name-2.adj
            $name-3.dist
            $name-3.fa
            $name-4.fa
            $name-5.fa
            $name-5.adj
            $name-5.path
            $name-6.fa
            $name-contigs.fa
            $name-indel.fa
            ...
      k28/
      ...
      k48/
      k50/
```

Note: `$name` is the name of your library

2. Installing Trans-ABySS

Upon extracting the TA package, you should see the following directories/files:

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

Note: From now on, the directory containing the above files/directories is denoted as `<TA_DIR>`

[2.1] `bin/`

TA requires the following external software packages for various purposes:

Software	Version	(R)equired / (O)ptional	Purpose	Download
BWA	0.5.9-r16 or above	R	Align reads to contigs	http://sourceforge.net/projects/bio-bwa/files/
Bowtie or Bowtie2		O	Align reads to contigs; **an alternative to BWA**	http://bowtie-bio.sourceforge.net/index.shtml
Pysam	0.1.2 or above	R	Samtools Python API for extracting read support in analysis modules	http://code.google.com/p/pysam/downloads/list
Samtools	0.1.18	R	View/create BAM files	http://sourceforge.net/projects/samtools/files/samtools/
ABySS	1.3.2 or above	R	Assembly and stage 0 of TA	http://www.bcgsc.ca/platform/bioinfo/software/abyss/releases/1.3.2
xa2multi.pl	-	R	Convert secondary alignments kept in `XA` tag into individual records	http://sourceforge.net/projects/bio-bwa/files/
Blat	34 or above	R	Align contigs to reference genome	http://users.soe.ucsc.edu/~kent/src/
GMAP		O	Align contigs to reference genome; **an alternative to Blat**	http://research-pub.gene.com/gmap/
Python	2.6	R	For running all analysis modules	http://www.python.org/getit/releases/2.6/
Perl	5.8	R	For running Perl wrappers	http://www.perl.org/get.html
mqsub	-	R	Submission of multiple jobs to cluster	provided in `/bin` in TA v1.3.5

We recommend users put the executables (if any exists) of the above software inside TA's `bin` directory and include the path of the `bin` directory in the `$PATH` variable in the `setup` file. TA stage 0 (FEM) requires two special-purpose modules from ABySS 1.3.2+, `abyss-filtergraph` and `abyss-`

`junction`. These two modules are compiled when ABySS is compiled but are **not** installed by default. Therefore, you must copy the executables for these two modules into either TA's `bin` directory, ABySS's installation directory, or anywhere accessible by TA.

[2.2] `setup`

The purpose of the `setup` file is to define all the proper environment variables needed by TA. To ensure all the dependent software can be accessible, a typical TA job begins with the Unix command:

```
source <TA_DIR>/setup
```

The `setup` file from the download package looks like so:

```
export TRANSABYSS_VERSION=1.3.5
export TRANSABYSS_PATH=/your/trans-abyss/code/directory
export PERL5LIB=$TRANSABYSS_PATH/wrappers:$PERL5LIB:/your/perl/libraries
export PYTHONPATH=/your/python/path:$PYTHONPATH:$TRANSABYSS_PATH
export ABYSSPATH=/directory/containing/abyss/executables
export LD_LIBRARY_PATH=/your/shared/libraries:$LD_LIBRARY_PATH
export PATH=$TRANSABYSS_PATH/bin:$ABYSSPATH:$LD_LIBRARY_PATH:$PYTHONPATH:$PATH
```

Users must update the `setup` file with all the pertinent file paths before attempting to run TA. The following environment variables must be defined because they are referenced in the wrapper scripts:

`TRANSABYSS_VERSION`
`TRANSABYSS_PATH` (note that this is actually `<TA_DIR>`)

After you have installed the required software and configured your `setup` file, you can check the paths with this command:

```
sh <TA_DIR>/check-prereq.sh
```

A few notes on cluster use:
Because of the sheer volume of transcriptome data, TA assumes the use of a cluster for any practical performance. The cluster job shell scripts created by TA are intended for the Sun Grid Engine (version 6.2u5). Users might need to modify the relevant modules (ie. `wrappers/analyze.pl`, `wrappers/setup.pl`, `utilities/submitjobs.sh`) for each stage of TA accordingly to fit your cluster environment. Experience in programming in Perl/Python and submitting jobs to your cluster would be of great value. Please read Section "3. Running Trans-ABySS" for more details on running TA. To ensure seamless submission of cluster jobs, please set up automatic login to your cluster head-node. Ask your system administrator for help or simply do a Google search for "SSH login without password".

[2.3] `configs/`

(a) Setting up `transcriptome.cfg`

The configuration file `transcriptome.cfg` specifies how the different steps of the TA pipeline are run. It contains the following major sections:

`[commands]`
Contains the default command options for running each script.

`[memory]`
Contains the default memory request for cluster jobs.

`[genome]`
Contains the paths to your reference genomes on the cluster.

`[contact]`
Contains the default email address to contact if the cluster jobs failed.

Projects and libraries
TA processes data on a per-library basis. Libraries are grouped under projects; each library must belong to a single project. Projects are configured as individual sections under the above major sections in `transcriptome.cfg`. Each project section contains a `topdir` attribute which specifies the directory under which the output of each library belonging to the project is stored under. The path for `topdir` must exist before running TA for a library under the particular project. Projects can have their own specific running parameters, which is to be applied on all libraries belonging to the same project. Here is an example project:

```
[your_project_name_here]
r2c_sam-mem: 10G
ta-r2c.py-cmd: CONTIGS READSLIST -p LIB -P PROJECT -o PATH/reads_to_contigs -t 8
-x
contact: helloworld@email.com
topdir: /your/working/directory/for/this/project
reference: hg19
```

The `reference` attribute specifies the name of the reference genome and is required if any analysis is to be performed on the library. If your project does not have a reference genome, simply define `reference` as `none` and only stages 0, 2, and 3 can be run. Other attributes are only needed if the default settings need to be overridden for the project. The `-mem` and `-cmd` postfixes distinguish what to override for the project.

(b) Setting up `model_matcher.cfg`

This file specifies the gene models that are used by the module `model_matcher.py` for contig-transcript mapping. The file is organized into sections where each section represents a reference genome. The gene model files which are referenced here are expected to be present in the `annotations` folder. See Section "2.5 annotations/" for instructions to download annotation files.

Each gene model is given a one-letter alias for quick referencing. For example, `e` represent the Ensembl gene model file. A comma-separated "order" field is used to specify the priority of the gene models when comparisons are made. Order is also used in breaking ties when the same contig can be mapped to genes from multiple models. An earlier model given in the order will be given precedence over the later ones when a single transcript is assigned to a contig. Here is an example of the contents:

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

(c) Setting up `job_script.cfg`

In TA 1.3.5, we have started creating jobs scripts with templates created by the user. The purpose of creating job scripts with templates is to allow external (non-GSC) users to easily interface the TA pipeline with their high performance computing (HPC) environment.

Currently, **only jobs submitted in Stage 0 (prepare reads) and Stage 2** would use these templates. However, all job scripts will be created with templates starting with the public release of TA 1.4.*.

This is the content of `job_script.cfg`:

```
local: gsc_local.txt
cluster_basic: gsc_sge_basic.txt
cluster_parallel: gsc_sge_parallel.txt
predessors_list_delimiter: ,
qsub_return_string: Your job ${JOBID} .* has been submitted
```

- `local` specifies the template for local job scripts.
- `cluster_basic` specifies the basic template for cluster job scripts.
- `cluster_parallel` specifies the template for multi-threaded cluster job scripts.
- `predessors_list_delimiter` specifies the delimiter for the list of predecessor job ids.
- `qsub_return_string` specifies the string returned when a cluster job has been submitted. `${JOBID}` is the part of the string containing the job id.

We have only provided the templates (`gsc_*.txt`) for the Sun Grid Engine of our HPC cluster. You must 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.
- `${PREDESSORS}` is the list of predecessors' job id. Note that `predessors_list_delimiter` from `jobs_script.cfg` would be used here.

- `${MEM}` is the amount memory to request for the job.
- `${THREADS}` is the number of CPUs for the parallel 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:

- `$TMPDIR` is the prefix for temporary files. Typically, the scheduler of your HPC cluster configures it automatically for each job. Otherwise, please configure it in the template to use the cluster node's local temporary directory along with a unique prefix, ie. `TMPDIR=/tmp/$JOB_ID.$TASK_ID.$QUEUE.`

[2.4] input/

An input file is what initiates the TA pipeline. There is no restrictions on how to name an input file. As discussed, TA analysis is performed on a per-library basis; therefore each line in the input file represents a single library and a single input file can contain multiple lines.

The format of each line in an input file contains 4 space-separated columns:

```
<LIBRARY> <ABYSS VERSION> <ASSEMBLIES DIR> <PROJECT NAME>
```

- `<LIBRARY> is the library name`
- `<ABYSS VERSION>` is the version number of ABySS used for the transcriptome assembly
- `<ASSEMBLIES DIR>` is the path to the directory containing the library's multi-k-mer assemblies
- `<PROJECT NAME>` is the project name, which has to be defined in `transcriptome.cfg`

An example input file:

```
L00001 1.3.2 /abyss/assembly/L00001 your_project_name_here
L00002 1.3.2 /abyss/assembly/L00002 your_project_name_here
L00003 1.3.2 /abyss/assembly/L00003 your_project_name_here
```

It is important that the assembly directories are set up as described in Section "[1.3] Running ABySS".

[2.5] `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 (copied from annotations/shared)
            [rest of annotation files]
      shared/ (provided)
```

```
splice_motifs.txt
```

TA mainly uses the annotation files available from the UCSC genome browser (ftp://hgdownload.cse.ucsc.edu/goldenPath/<genome>/database) for this purpose. 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 downloads and running the following processing steps. This is an example of how to use the provided shell script to get 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

The script uses wget for downloading. 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 user needs to run this command to split up the dbSNP annotation by chromosome:

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

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/transriptome.cfg under [genomes], ie.

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

A <genome>.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 .

[2.6] analysis/ and utilities/

These folders contain the analysis modules written in Python.

[2.7] `sample_output/`

We have provided sample output files for our sample library. We encourage users to run TA on the sample library. This is a great exercise to get familiar with the process for setting up your project and running TA. In addition, this exercise may also help you check whether the required software have installed properly. Otherwise, it is very unlikely that you will get the same output files.

Before you begin, make sure you have installed and set up ABySS 1.3.2 and Trans-ABySS 1.3.5 properly. Please refer to the Sections "[1.1] Installing ABySS", "[2.1] bin/", "[2.5] annotations/" for details.

Step 1. Generate the multi-kmer transcriptome assemblies with ABySS 1.3.2. Please refer to Section "[1.3] Running ABySS" for details.

These are the input reads for the sample library:

```
<TA_DIR>/sample_output/ABySS/SampleProject/abyss-1.3.2/sim0003/reads_1_export.fq
<TA_DIR>/sample_output/ABySS/SampleProject/abyss-1.3.2/sim0003/reads_2_export.fq
```

We used k-mer sizes {62, 64, 66, 68, 70, 72, 74} and the ABySS settings described in Section [1.3]. We called the sample library "sim0003".

Step 2. Set up a new working directory for TA and put the path as `topdir` under `[SampleProject]` in `<TA_DIR>/configs/transcriptome.cfg`:

```
[SampleProject]
topdir: /path/to/your/topdir/here
reference: hg19
```

Please refer to Section "[2.3] configs/" for details.

Step 3. Set up the input file like so:

```
sim0003 1.3.2 /path/to/your/abyss-assemblies/here SampleProject
```

Please refer to Section "[2.4] input/" for details.

Step 4. Run TA from stages 0, and 2 to 8 as described in Section "3. Running Trans-ABySS". After running stage 0, you can skip stage 1 and copy our JAGUAR BAM file and its index

```
<TA_DIR>/sample_output/Trans-ABySS/SampleProject/sim0003/Reads_to_genome/
            output.jag.sorted.bam
            output.jag.sorted.bam.bai
```

to your "Reads_to_genome" directory.

3. Running Trans-ABySS

Before running TA on any new libraries, please check:

1. Your ABySS multi-k-mer transcriptome assemblies have completed successfully. There should be no errors in the logs and all output files are present. The directory structure and names of the output files of your assemblies are adjusted to be compatible for . Please refer to Section "[1.3] Running ABySS" for more details.
2. Your "project" is set up correctly in `config/transcriptome.cfg` and the directory path for `topdir` exists. Note that `topdir` defines where will place its output for the project. Please refer to Section "[2.3] configs/" for more details.
3. Your input file is set up correctly. Particularly, check whether the library name and the path to the assemblies directory are correct. Please refer to Section "[2.4] input/" for more details.

Figure 1 shows an overview of TA. The pipeline is divided into 9 stages (0 to 8). Each stage is described in this section.

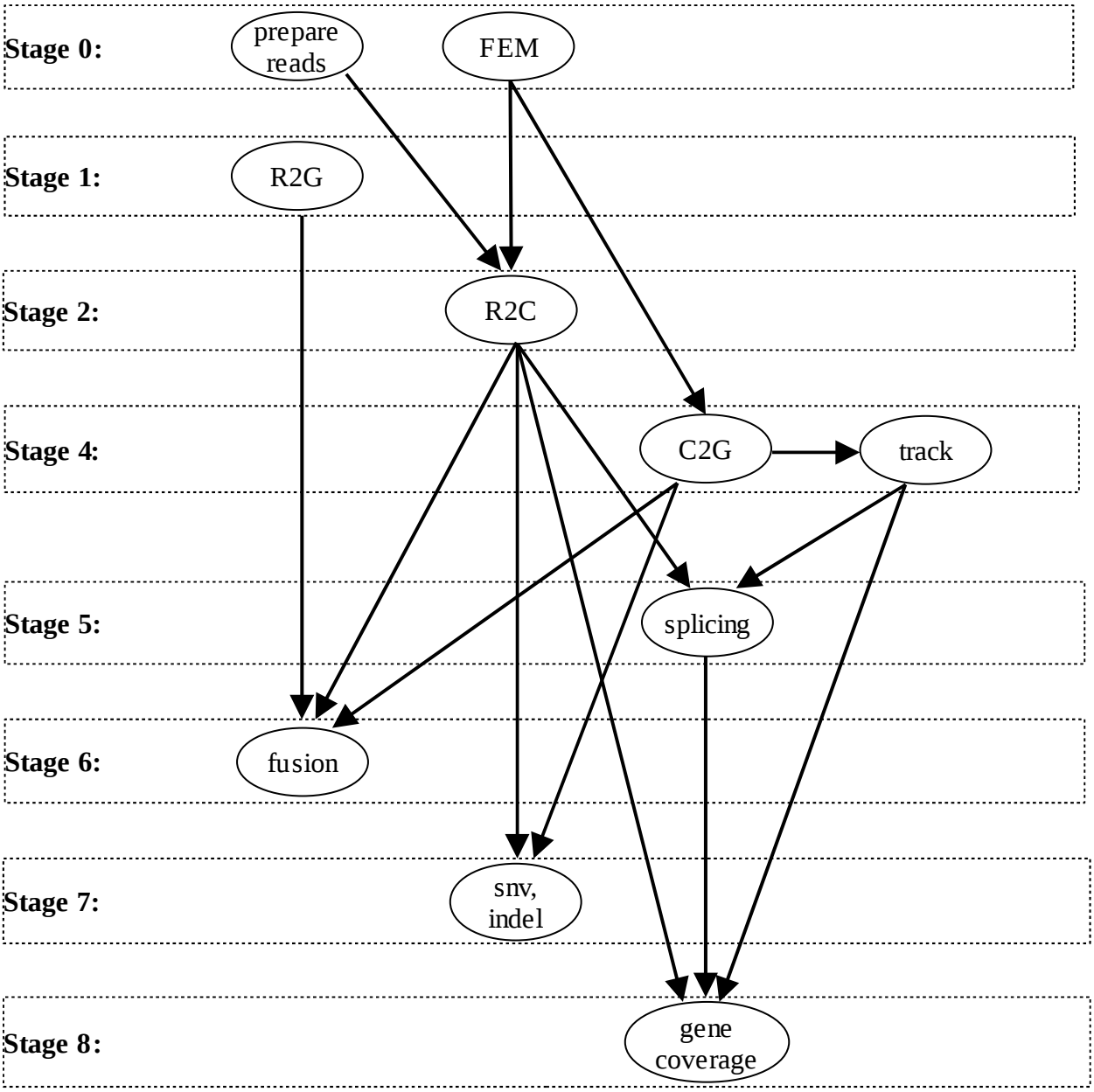

Figure 1. An overview of the Trans-ABySS pipeline

FEM: Filter, extend, merge assemblies
R2G: Reads-to-genome alignments
R2C: Reads-to-contigs alignments
C2G: Contigs-to-genome alignments

Stage 3 is no longer available and has been merged to Stage 2.

Abbreviations used in this section:

`<HEAD_NODE>`	name of the cluster head node
`<INPUT>`	path to the input file
`<TOPDIR>`	path to the project directory that holds the output for each library
`<LIBRARY>`	name of the library, as known as `$name` in previous section
`<ASSEMBLY>`	path to the directory containing the ABySS multi-k-mer assemblies
`<STAGE>`	the stage number

Before running any scripts from TA, you need to set up your environment with this command:

```
source <TA_DIR>/setup
```

To understand the usage and available parameters for TA:

```
<TA_DIR>/wrappers/trans-abyss.sh -h
```

To run each stage on the cluster with the TA wrapper:

```
<TA_DIR>/wrappers/trans-abyss.sh \
      -c <HEAD_NODE> \
      -i <INPUT> \
      -<STAGE>
```

Suppose `<INPUT>` contains multiple libraries and you want to run TA on one particular library called `<LIBRARY>`, you can run the wrapper like so:

```
<TA_DIR>/wrappers/trans-abyss.sh \
      -c <HEAD_NODE> \
      -i <INPUT> \
      -<STAGE> \
      -l <LIBRARY>
```

Now, suppose you want to run TA on `<N>` libraries starting with library called `<LIBRARY>` in `<INPUT>`, you can run the wrapper like so:

```
<TA_DIR>/wrappers/trans-abyss.sh \
      -c <HEAD_NODE> \
      -i <INPUT> \
      -<STAGE> \
      -s <LIBRARY> \
      -n <N>
```

[3.1] Stage 0

Three tasks are preformed in this step.

(a) Set up the working directories for TA.

command:

```
<TA_DIR>/wrappers/setup.pl \
      <INPUT> \
      -make_dir \
      -cluster <HEAD_NODE>
```

TA will set up the directories and symbolic links like so:

```
<TOPDIR>/
      <LIBRARY>/
            Reads_to_genome/
            Assembly/
                  current -> ./abyss-1.3.2
                  abyss-1.3.2/
                        fusions/
                        k$k1/
                        ...
                        k$kn/
                        merge/
                        novelty/
                        reads_to_contigs/
                        snv/
                        source -> <ASSEMBLY>
                        tracks/
```

(b) Prepare reads for alignments to contigs.

command:

```
<TA_DIR>/wrappers/setup.pl \
      <INPUT> \
      -get_reads \
      -cluster <HEAD_NODE>
```

input files:
`<ASSEMBLY>/<LIBRARY>.in`
output files:
`<TOPDIR>/<LIBRARY>/Assembly/abyss-1.3.2/reads_to_contigs/<LIBRARY>.in`

If the input reads for the ABySS assemblies are `.bam` or `.fq.gz` files, then the `reads` directory would not be created under the `reads_to_contigs` directory and `reads_to_contigs/<LIBRARY>.in` would be a symlink to `source/<LIBRARY>.in`. Otherwise, cluster jobs would be submitted to convert/compress read files to `.fq.gz`.

If you did not follow our aforementioned method to generate your paired-end assemblies with ABySS, this part of stage 0 might not work for you. In this case, you must create `reads_to_contigs/<LIBRARY>.in` yourself. `reads_to_contigs/<LIBRARY>.in` is in the same format as described in Section "1.3 Running ABySS".

You can skip this part and Stage 2 if you want to align reads to contigs yourself.

(c) Filter assemblies, extend contigs, merge assemblies (FEM).

command:

```
<TA_DIR>/wrappers/setup.pl \
      <INPUT> \
      -fem \
      -cluster <HEAD_NODE>
```

TA takes the output of the multiple-k ABySS assemblies and performs FEM to generate a single meta-assembly for analysis.

(i) Filter assemblies:
input files:
`<ASSEMBLY>/k*/<LIBRARY>-contigs.fa`
`<ASSEMBLY>/k*/<LIBRARY>-5.adj`
`<ASSEMBLY>/k*/<LIBRARY>-5.path`
output:
`<TOPDIR>/<LIBRARY>/Assembly/abyss-1.3.2/k*/<LIBRARY>-f.fa`

The input assembly (`<LIBRARY>-contigs.fa`) is filtered with the following criteria:

- Removes all contigs less than 2k-1 in size
- Removes island contigs (contigs that have no neighbors in the ABySS adjacency graph) less than or equal to 150bp in size

(ii) Extend contigs and indel bubbles:
input files:
`<ASSEMBLY>/k*/<LIBRARY>-1.fa`
`<ASSEMBLY>/k*/<LIBRARY>-2.adj`
`<ASSEMBLY>/k*/<LIBRARY>-3.fa`
`<ASSEMBLY>/k*/<LIBRARY>-4.fa`
`<ASSEMBLY>/k*/<LIBRARY>-5.fa`
`<ASSEMBLY>/k*/<LIBRARY>-5.dist`
`<ASSEMBLY>/k*/<LIBRARY>-5.adj`
`<ASSEMBLY>/k*/<LIBRARY>-5.path`
`<ASSEMBLY>/k*/<LIBRARY>-indel.fa`
output:
`<TOPDIR>/<LIBRARY>/Assembly/abyss-1.3.2/k*/<LIBRARY>-j.fa`
`<TOPDIR>/<LIBRARY>/Assembly/abyss-1.3.2/k*/<LIBRARY>-b.fa`

ABySS single-end contigs (`<LIBRARY>-3.fa`, `<LIBRARY>-4.fa`, `<LIBRARY>-5.fa`) are extended with the following criteria:

- Extends 1-in 1-out contigs which have read pair support between flanking contigs
- The number of read pairs required is the value of the `n` parameter from the ABySS assembly
- Excludes 1-in 1-out contigs used in the final stage of assembly

ABySS indel bubbles (`<LIBRARY>-indel.fa`) are extended as long as there is no ambiguity in adjacency.

(iii) Combining output from (i) and (ii):
input files:
```
<TOPDIR>/<LIBRARY>/Assembly/abyss-1.3.2/k*/<LIBRARY>-f.fa
<TOPDIR>/<LIBRARY>/Assembly/abyss-1.3.2/k*/<LIBRARY>-j.fa
<TOPDIR>/<LIBRARY>/Assembly/abyss-1.3.2/k*/<LIBRARY>-b.fa
```
output files:
```
<TOPDIR>/<LIBRARY>/Assembly/abyss-1.3.2/k*/<LIBRARY>-contigs.fa
```

(iv) Merge assemblies from (iii):
input files:
```
<TOPDIR>/<LIBRARY>/Assembly/abyss-1.3.2/k*/<LIBRARY>-contigs.fa
```
output files:
```
<TOPDIR>/<LIBRARY>/Assembly/abyss-1.3.2/merge/<LIBRARY>-contigs.fa
```

The filtered and extended assemblies (iii) are merged into a single meta-assembly (iv) where a contig is removed if it has an exact, full length match to its sequence in an assembly of smaller k-mer size.

The four parts of FEM happen within a single cluster job. The cluster job is configured to use 8 threads. To configure the job scripts to fit your cluster environment, please modify `setup.pl` (`sub fem`) and `abyss-rmdups-iterative` (`function rmdups`).

[3.2] Stage 1

Mate-pair reads need to be aligned to a reference genome and exon-exon junction reference for finding evidence for fusion candidates. However, these alignments are not performed as part of TA. We use JAGuaR to align reads to genome. For more information, please read:
http://www.bcgsc.ca/platform/bioinfo/software/jaguar
http://www.bcgsc.ca/platform/bioinfo/docs/jaguar/Butterfield_JAGuaR_Nov2011.pdf

When you run this stage, you would see this message:

```
Please put your code in:
<TA_DIR>/wrappers/setup.pl (sub copy_bam)
for copying JAGUAR's BAM file to the "Reads_to_genome" directory!
```

Obviously, this stage does not do anything. Please modify `setup.pl` to suit your needs.

[3.3] Stage 2

This step uses BWA to aligns reads to the meta-assembly from stage 0.
input files:
`<TOPDIR>/<LIBRARY>/Assembly/abyss-1.3.2/reads_to_contigs/<LIBRARY>.in`
`<TOPDIR>/<LIBRARY>/Assembly/abyss-1.3.2/merge/<LIBRARY>-contigs.fa`
Remember from stage 0:
`<LIBRARY>.in` is a text file that list the paths to input reads files (fastq, fq.gz or bam)

output files:
`<TOPDIR>/<LIBRARY>/Assembly/abyss-1.3.2/reads_to_contigs/<LIBRARY>-contigs.bam`
`<TOPDIR>/<LIBRARY>/Assembly/abyss-1.3.2/reads_to_contigs/<LIBRARY>-contigs.bam.bai`

Alternatively, you may align your reads to contigs with other aligners (such as Bowtie or Bowtie2) and skip stage 2. You must, however, name your output BAM file as `<LIBRARY>-contigs.bam` and put the BAM file and its index in the `reads_to_contigs` directory.

[3.4] Stage 3

Stage 3 has been merged to stage 2 in TA 1.3.5.

[3.5] Stage 4

This step performs two tasks.

(i) Aligns contigs in the meta-assembly from stage 0 to the reference genome with BLAT.
input files:
`<TOPDIR>/<LIBRARY>/Assembly/abyss-1.3.2/merge/<LIBRARY>-contigs.fa`
output files:
`<TOPDIR>/<LIBRARY>/Assembly/abyss-1.3.2/merge/cluster/<LIBRARY>-contigs/output/`
`seq.*.psl`

(ii) Filter the BLAT alignments and generate a UCSC custom track.
input files:
`<TOPDIR>/<LIBRARY>/Assembly/abyss-1.3.2/merge/cluster/<LIBRARY>-contigs/output/`
`seq.*.psl`
output files:
`<TOPDIR>/<LIBRARY>/Assembly/abyss-1.3.2/tracks/`
`<LIBRARY>.merge.contigs.best.unique.m90.blat.psl`

[3.6] Stage 5

This step finds novel transcript splicing events.
input files:
```
<TOPDIR>/<LIBRARY>/Assembly/abyss-1.3.2/tracks/
      <LIBRARY>.merge.contigs.best.unique.m90.blat.psl
<TOPDIR>/<LIBRARY>/Assembly/abyss-1.3.2/reads_to_contigs/<LIBRARY>-contigs.bam
```
output files:
Please refer to Section "[4.3] Novel Splicing (model_matcher.py)".

[3.7] Stage 6

This step finds candidate gene fusions and large structural rearrangements.
input files:
```
<TOPDIR>/<LIBRARY>/Reads_to_genome/*.bam
<TOPDIR>/<LIBRARY>/Assembly/abyss-1.3.2/merge/cluster/<LIBRARY>-contigs/output/
      seq.*.psl
<TOPDIR>/<LIBRARY>/Assembly/abyss-1.3.2/reads_to_contigs/<LIBRARY>-contigs.bam
<TOPDIR>/<LIBRARY>/Assembly/abyss-1.3.2/reads_to_contigs/<LIBRARY>-contigs.bam.bai
```
output files:
Please refer to Section "[4.1] Fusion (fusions.py)".

[3.8] Stage 7

This step finds candidate single nucleotide variants, insertions, and deletions.
input files:
```
<TOPDIR>/<LIBRARY>/Assembly/abyss-1.3.2/merge/cluster/<LIBRARY>-contigs/output/
      seq.*.psl
<TOPDIR>/<LIBRARY>/Assembly/abyss-1.3.2/merge/cluster/<LIBRARY>-contigs/input/
      seq.*.fa
<TOPDIR>/<LIBRARY>/Assembly/abyss-1.3.2/reads_to_contigs/<LIBRARY>-contigs.bam
<TOPDIR>/<LIBRARY>/Assembly/abyss-1.3.2/reads_to_contigs/<LIBRARY>-contigs.bam.bai
```
output files:
Please refer to Section "[4.2] SNV/INDEL (snv_caller.py)".

[3.9] Stage 8

This step reports the coverage for each gene from the reference genome that was detected.
input files:
```
<TOPDIR>/<LIBRARY>/Assembly/abyss-1.3.2/novelty/coverage.txt
<TOPDIR>/<LIBRARY>/Assembly/abyss-1.3.2/tracks/
      <LIBRARY>.merge.contigs.best.unique.m90.blat.psl
<TOPDIR>/<LIBRARY>/Assembly/abyss-1.3.2/reads_to_contigs/<LIBRARY>-contigs.bam
```
output file:
```
<TOPDIR>/<LIBRARY>/Assembly/abyss-1.3.2/novelty/gene_coverage.txt
```
Please refer to Section "[4.4] Gene coverage".

4. Analysis Output

[4.1] Fusion (fusions.py)

Output	Description
fusions.tsv	Unfiltered fusion events captured by split contig alignments
fusions_filtered.tsv	Filtered events, where: 1. <num_read_pairs> >= *--min_read_pairs (default: 4)* AND <= *--max_read_pairs (default: 2000)* *<num_read_pairs> = <flanking_pairs>* + minimum*(<breakpoint_pairs>)* 2. <spanning_reads> >= *--min_span_reads (default: 2)*
local.tsv	"local" events, when: 1. alignment target regions overlap, or 2. alignment target regions overlap same gene, or 3. transcripts mapped by target regions overlap
LOG	Run log recording command run and parameters used

Content of `fusion_filtered.tsv`:

Field	Description
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 <rearrangement> and <breakpoint>
contig	contig ID
contig size	size(length) of <contig>
genomic_regions	The 2 genomic regions the contig aligns to. Format: chromosomeA:start1-end1,chromosomeB:start2-end2
contig_regions	The corresponding contig coordinates of the 2 genomic regions. Format: start1-end1,start2-end2 (regions in the same order of genomic regions)
strands	Relative orientation of the 2 alignments in relation to the genome. Format: [+|-],[+|-]
flanking_pairs	Number of read pairs from reads-to-genome alignments with both mates flanking the breakpoint, both pointing towards each other
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

spanning_reads	Number of reads spanning junction from contig alignment of reads
rearrangement	Underlying genome rearrangement deduced by relative contig alignment orientations. Can be “translocation”, “deletion”, “inversion”, or “duplication”
breakpoint	Junction breakpoint. Format: chrA:coordinate1|chrB:coordinate2
genes	Format: GeneA([+|-])GeneB([+|-]), where [+|-] indicates the relative orientation of the contig alignment to the gene strand, i.e. '+' indicates the contig aligns in the same direction of the gene strand, '-' indicates the contig aligns in opposite direction of the gene strand
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
type	Can be: “gene_fusion” - if a gene resides in both genomic regions “lsr” - large scale rearrangement, any event not a “gene_fusion”

[4.2] SNV/INDEL (snv_caller.py)

Output	Description
events.txt	Unfiltered snv/indel events captured by gapped contig alignments
events_filtered.txt	Filtered events, <event_reads> >= *--min_reads_contigs (default: 3)*
events_filtered_novel.txt	Filtered events not annotated in dbSNP
events_exons.txt	Filtered, non-synonymous events residing in gene exons
events_exons_novel.txt	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.txt`:

Field	Description
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> and <alt>
type	Event type. Can be “snv”, “ins”, “del”, “inv”
chr	Chromosome number
chr_start	Chromosome start coordinate. If <type> == “ins”, <chr_start> = coordinate immediately upstream of insertion. If <type> == “del”, <chr_start> = first base of deletion
chr_end	Chromosome end coordinate. If <type> == “ins”, <chr_end> = <chr_start>. If <type> == “del”, <chr_end> = last base of deletion
ctg	Contig ID
ctg_len	Length of <ctg> that captures event
ctg_start	Contig start coordinate. If <type> == “ins”, <chr_start> = coordinate immediately upstream of insertion. If <type> == “del”, <chr_start> = first base of deletion
ctg_end	Contig end coordinate. If <type> == “ins”, <chr_end> = <chr_start>. If <type> == “del”, <chr_end> = last base of deletion
len	Length (size) of event
ref	Reference allele. If <type> == “ins”, <ref> = “na”
alt	Alternative allele. If <type> == “del”, <ref> = “na”

event_reads	Total number of reads spanning *event* from reads-to-contig alignment
contig_reads	Number of reads spanning event in *contig* from reads-to-contig alignment
genome_reads	Total number of reads spanning *event* from reads-to-genome alignment
gene	Gene in affected locus. Format: gene:transcript:[intron|exon]number|effect on open reading frame (see below) (if event spans more than 1 exon/intron, the output becomes: geneA:transcriptA:[intron|exon]numberA|geneB:transcriptB:[intron|exon]numberB|effect on open reading frame)
repeat-length	Length of repeat in alternative allele, e.g. AAAA = 4, CAGCAG = 2
ctg_strand	Query strand of alignment in relation to reference
from_end	Distance (bases) from event to end of contig
confirm_contig_region	Contig coordinate range (start, end) used for checking for event existence in reads-to-contig alignments
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.
repeatmasker	Overlap with RepeatMasker annotations. Type of repeat reported if overlap is True e.g. AluSx, LTR47A . '-' if overlap is False.
within_segdup	Overlap with segmental duplication. Chromosome:Start_coordinate of segdup partner reported if overlap is True, e.g. chr1:17048246. '-' if overlap is False.
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”
dbsnp	dbSNP entries if event is already annotated in dbSNP e.g. rs12028735,rs71510514

[4.3] Novel Splicing (model_matcher.py)

Output	Description
events.txt	Unfiltered novel splicing events not observed in annotations specified in *model_matcher.cfg*
events_filtered.txt	Filtered events. See below for filtering criteria.
events_summary.txt	Tally of unfiltered events by <type>
events_filtered_summary.txt	Tally of filtered events by <type>
coverage.txt	Transcript coverage
mapping.txt	Mapping of contig to annotated transcripts
log.txt	Detailed block-by-block mapping of alignments to exons
events_reads	Directory containing FASTA files of event-spanning reads. Format of file names: *contig::event_type::chromosome:start-end.fa*
events.bed	Unfiltered events in bed format
events_filtered.bed	Filtered events in bed format
LOG	Run log recording command run and parameters used

Contents of `events_filtered.txt`:

Field	Description
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 <type> and <coord>.
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
contig	Contig ID
transcript	Transcript ID

gene	Gene name
exons	Exon number(s), relative to transcript strand, start from 1
align_blocks	Alignment block numbers, counted in ascending order of coordinate, start from 1. Will be multiple values for skipped_exon, novel_intron, novel_utr, and novel_transcript
coord	Coordinate of novel block. Format: chromosome:start-end
splice	Splice site sequence surrounding novel junction e.g. GT-ag(U2/U12), where U2/U12 is name of splice motif
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
size	Size of novel block. Only applicable to AS53, novel_exon, novel_intron, novel_transcript, and novel_utr
orf	Effect on open reading frame. See below.
spanning_reads	Number of reads spanning novel junction, gathered from reads-to-contig alignments
contig_coverage	Number of reads spanning novel block. If size of novel block is small, <contig_coverage> will be equal to <spanning_reads>
contig_neighbor	Number of reads spanning blocks/junctions immediately upstream and downstream. This is to inform relative expression levels
read_support	'passed' if the following: skipped_exon, novel_intron: <spanning_reads> >= *minimum_spanning_reads* AS5, AS3, AS53, novel_exon, novel_utr, retained_intron: <contig_coverage> >= *minimum_spanning_reads* AND <contig_neighbor>/<contig_coverage> >= *maximum_coverage_differential* novel_transcript: <spanning_reads> of each of the novel junction >= *minimum_spanning_reads*
filter	'passed' if read_support == 'passed', and the following: AS5, AS3, AS53, novel_exon, novel_utr, novel_intron, nove_transcript: surrounding splice sequences are canonical splice sites retained_intron: <multi_3> is True

Contents of `coverage.txt`:

Field	Description
transcript	Transcript name
gene	Gene name
total_coverage	Number of exonic bases covered by contig

transcript_length	Length of <transcript>
best_contig	Best contig mapped to <transcript> in terms of bases covered
best_contig_coverage	Coverage of <transcript> by <best_contig>
nbr_contigs	Number of contigs mapped to <transcript>
contigs	List of contigs mapped to <transcript>
contig_coverage	Total coverage of <transcript> by <contigs>

Sample line of `mapping.txt`:

```
<A> matches <B>(<C>) <D> model:<E>(wt:<F>) in <G> blocks total_blocks=<H>
    total_exons=<I> <J> coord:<K> score:<L> events:<M> coverage:<N>
```

Field	Description
A	Contig ID
B	Transcript name
C	Gene name
D	CODING or NONCODING (of transcript)
E	Gene model initial (specified in *model_matcher.cfg* e.g. e=Ensembl, r=Refseq)
F	Weight of gene model in matching (first model's wt = # models used, second model's wt = # models used – 1, etc)
G	Number of alignment blocks mapped to exons
H	Total number of blocks in alignment
I	Total number of exons in transcript
J	“partial_match” or “full_match”. “full_match” if both edges of internal alignment blocks match and internal edges of outermost blocks match; “partial_match” otherwise
K	Coordinate of alignment
L	Score = Number of edges matched. AS5 and AS3 junctions considered “matched”
M	Number of novel splicing events
N	Coverage of transcript

[4.4] Gene coverage

Field	Description
gene	Gene name
nreads	Total number of reads covering gene
total_read_length	Sum of length of reads covering gene
union_aligned_block_length	Total length of union of alignment blocks mapped to gene
normalized_coverage	<total_read_length> / <union_aligned_block_length>

5. Miscellaneous

[5.1] 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)

6. Technical Support

Please direct your bug reports, questions, and suggestions to the Trans-ABySS Google Group:
trans-abyss@googlegroups.com

You can also read and search existing discussions on the Google Group at:
http://groups.google.com/group/trans-abyss

- End of User Manual -