

Trans-ABySS v1.0: User Manual

13 October 2010

Prepared by:

Readman Chiu, Rong She, Hisanaga Mark Okada, Gordon Robertson, Shaun Jackman, Jenny Qian

On behalf of:

Gordon Robertson, Jacqueline Schein, Readman Chiu, Richard Corbett, Matthew Field, Shaun D Jackman, Karen Mungall, Sam Lee, Hisanaga Mark Okada, Jenny Q Qian, Malachi Griffith, Anthony Raymond, Nina Thiessen, Timothee Cezard, Yaron S Butterfield, Richard Newsome, Simon K Chan, Rong She, Richard Varhol, Baljit Kamoh, Anna-Liisa Prabhu, Angela Tam, YongJun Zhao, Richard A Moore, Martin Hirst, Marco A Marra, Steven J M Jones, Pamela A Hoodless & Inanc Birol

Genome Sciences Centre, BC Cancer Agency
Vancouver, BC, Canada V5Z 4S6

Contact: Readman Chiu (rchiu@bcgsc.ca)

User forum: <http://groups.google.com/group/trans-abyss?hl=en>

Table of contents

ABySS and Trans-ABySS

Licenses

Getting ABySS

Getting Trans-ABySS

 Download

 Unpacking

Configuration

 Trans-ABySS Software

 External software

 Configuration files

 Input file

 Transcript annotations and genome sequence

Assembling and analyzing transcriptome data

 Trans-ABySS pipeline overview

 ABySS assemblies and folder structure

 Trans-ABySS folder structure

 Run Trans-ABySS pipeline

- Setting up contigs for analysis
 - Contig and read alignments
 - Read alignments to contigs
 - Contig alignments to a reference genome
 - Aligning reads to a reference genome
 - Transcriptome assembly analysis
 - Identify candidate novel transcript structures
 - Estimate gene-level expression
 - Identify candidate gene fusion events
 - Identify candidate SNVs and indels
 - Identify candidate polyadenylation sites
- Datasets
 - Large
 - Small
 - Insr_UTR
 - Polyadenylation site analysis
- References

ABySS and Trans-ABySS

ABySS is a de Bruijn graph-based short-read assembler that can process genome or transcriptome sequence data (Simpson et al. 2009, Birol et al. 2009).

Trans-ABySS is an analysis pipeline for post-processing ABySS assemblies of transcriptome sequencing data. It addresses varying transcript expression levels by processing multiple assemblies across a range of k values (Robertson et al. 2010).

The v1.0 pipeline can map assembled contigs to annotated transcripts (e.g. RefSeq, Ensembl,...), and can identify candidate novel splicing events such as exon-skipping, novel exons, retained introns, novel introns, and alternative splicing. It can also extract candidate SNVs, INDELs, and gene fusion events from contig alignment data.

The Trans-ABySS pipeline consists of a) Perl wrapper scripts; b) Python, Perl and bash scripts; and c) command line applications. The pipeline can be run on any Linux platform. Processing large datasets will require a computer cluster.

Licenses

ABySS and Trans-ABySS are released under the terms of the BC Cancer Agency software license agreement. <http://www.bcgsc.ca/platform/bioinfo/>

[license/bcca_2010](#)

Getting ABySS

The Trans-ABYSS pipeline will process outputs from ABySS v1.1.2+. ABySS v1.1.1 was used for the Nature Methods publication. Source code for v1.1.1 and for the most current release of ABySS is available at:

www.bcgsc.ca/platform/bioinfo/software/abyss

The ABySS-users discussion group is available at:

<http://groups.google.com/group/abyss-users>

Getting Trans-ABYSS

1. Download

The pipeline software can be downloaded from:

<http://www.bcgsc.ca/platform/bioinfo/software/trans-abyss>

2. Unpacking

After unpacking, files will be automatically organized into five folders:

analysis	Contains Python modules and Perl scripts that are used for analyzing ABySS-assembled transcriptome assemblies.
annotations	Contains transcript and repeat annotation files used in analysis. It is organized by reference genome assembly (e.g. hg18, mm9, etc).
configs	Contains configuration files (.cfg) that are used for running the trans-ABYSS pipeline.
utilities	Contains Python modules (.py) and ABySS-related binaries that support the analysis modules.
wrappers	Contains Perl scripts (.pl) that are wrappers for running the Trans-ABYSS pipeline.
sample_data	Contains a small sample dataset that can be used for testing

Configuration

1. Trans-ABySS Software

Most of the software is written in Python. Because Trans-ABySS uses Pysam (<http://code.google.com/p/pysam/>) to parse .sam files, Python 2.6 or later is required.

The wrapper scripts for running the pipeline are written in Perl. All Perl5 versions should work. To use the wrappers, you must add to the Perl path a simple custom configuration module for parsing config files. This module is supplied in the “wrappers” folder.

By setting the following environmental variables you should be ready to run the Trans-ABySS software:

```
export TRANSABYSS_PATH=/home/user/trans-ABySS
export PYTHONPATH=.:$PYTHONPATH:$TRANSABYSS_PATH
export PERL5LIB=.:$PERL5LIB:$TRANSABYSS_PATH/wrappers
```

In addition, the reference genomes and their annotations that are used in analysis should be present in the "annotations" folder. The current trans-ABySS package comes with annotation files for two reference genomes: “hg18” and “mm9”. For analysis on other genomes, please set up their annotation folders in the same fashion.

For each reference genome, there should be a "genome.fa" in its corresponding folder. For convenience, a “setup” file is included in the trans-ABySS root folder, which includes the setup of the environmental variables and download genome files from UCSC web site. Change the “TRANSABYSS_PATH” to your own trans-ABySS directory. Then type “source setup” at the command line.

The ‘External software’ section (below) lists other required software.

2. External Software

In addition to Python and Perl, Trans-ABySS requires the following:

1. Blat (<http://users.soe.ucsc.edu/~kent/src/>)

Blat is used for:

1. Merging: pairwise alignment of contigs to remove redundant contigs.
2. Aligning contigs to a reference genome.

2. Pysam (<http://code.google.com/p/pysam/>)

Pysam is used for parsing .bam files for parsing read-to-contig alignments.

3. BioPython (<http://www.biopython.org/wiki/Download>)

Biopython is used in two parts of Trans-ABYSS analysis:

1. Translating DNA sequence into peptide sequence for identifying potential open reading frames.

2. The "NCBIStandalone.py" module is used for parsing Blast-format output from Blat to extract candidate single nucleotide variants (SNVs) and insertion-deletions (indels). After downloading the module, edit the following line in so that HSPs of all scores will be parsed:

```
r"Score =\s*([0-9.e+]+) bits \(([0-9]+)\)", line,
```

should be changed to:

```
r"Score =\s*([0-9.e+-]+) bits \(([0-9-]+)\)", line,
```

4. Samtools (<http://samtools.sourceforge.net/>)

Samtools is used for merging and indexing read alignment files.

5. Bowtie (<http://bowtie-bio.sourceforge.net/index.shtml>)

Bowtie is used in single-end alignment for aligning reads to contigs.

6. The CPAN Perl module Config::General and IO::Compress

<http://search.cpan.org/~tlinden/Config-General-2.49/General.pm>

<http://search.cpan.org/~pmqs/IO-Compress-2.030/lib/IO/Uncompress/Gunzip.pm>

IO::Compress is only required if the input reads are gzipped or bzipped. These Perl modules are used by the polyadenylation site scripts.

7. BWA (<http://bio-bwa.sourceforge.net/bwa.shtml>)

BWA is used to align PAM and EJ reads to known transcript sequences in the polyadenylation site analysis.

3. Configuration files

Trans-ABYSS analyses are performed on individual libraries, i.e. short-read sequencing datasets. However, to support work on a project that involves multiple related libraries (e.g. tens of patients for a disease), sets of libraries can

be organized under a common project directory, and can share common run configuration settings. Settings are specified in configuration files, as follows.

The wrapper scripts use the following configuration files to run the pipeline:

- **projects.cfg**
For each project, the user specifies the reference genome and the top-level 'project' directory. A 'project' directory will contain a subdirectory for each of its libraries. In `projects.cfg`, default parameters for each script are specified in the "default" section. Defaults can be overridden by values set with each project. Uppercase words (e.g. `MERGINGDIR`) are used by calling scripts as templates that will be automatically replaced with appropriate values during a pipeline run.

```
[default]
merge.pl: VERDIR LIB contigs MERGINGDIR
align_parser.py: BLAT_DIR blat -n 1 -u -m 90 -d -k
TRACK_NAME -o PSL -f CONTIGS
...

[projectA]
topdir: /projects/projectA
reference: hg18

[projectB]
...
```

Figure 1. Example organization of "projects.cfg"

- **binaries.cfg**
Paths to external software are specified in "software: path" format, with one line for each executable. An example file is provided in the distribution. Note in the example file, two versions of "python" are specified: "python" points to the executable of the correct version of python that runs on GSC's cluster, and "python_xhost" points to the executable of the version of python that runs locally. Please replace all paths to point to proper binaries in your own computing environment. Do not change the name of software (the part before ":").

```
[binaries]
python: /gsc/software/linux-x86_64/python-builder-2.6.4/
bin/python
python_xhost: /home/rshe/bin/bin/python
perl: /usr/local/bin/perl5.8.3
blat: /home/pubseq/BioSw/blat/blat34/blat
exonerate: /home/pubseq/BioSw/exonerate/exonerate-2.2.0-
x86_64/bin/exonerate
```

```

bwa: /home/pubseq/BioSw/bwa/bwa-0.5.6/bwa
bowtie: /home/rchiu/bin/bowtie-0.12.5/bowtie
bowtie_build: /home/rchiu/bin/bowtie-0.12.5/bowtie-build
samtools: /home/pubseq/BioSw/samtools/0.1.6/samtools
export2fq: /home/pubseq/BioSw/Maq/maq-0.7.1_x86_64-linux/
scripts/fq_all2std.pl export2std
biopython: /home/rchiu/python/biopython-1.52
mqsub: /opt/mqtools/bin/mqsub

```

Figure 2. Example of “binaries.cfg”

- cluster.cfg
Specify cluster job settings including the memory requirement for running different scripts on cluster, and the genome file used for each reference genome. This file is required when running jobs on cluster.

```

[memory]
merge.pl: 1G
fusion.py: 1G
model_matcher.py: 10G
reads_to_contigs.py: 1G
align_parser.py: 1G
cluster_align.py: 5G
gene_coverage.py: 1G

[genomes]
hg18: /var/tmp/genome/lymphoma/ucsc-hg18.fa
mm9: /var/tmp/genome/mouse/mm9_build37_mouse.fasta

```

Figure 3. Example of “cluster.cfg”

- align.cfg
Specify parameters used by each aligner when contigs are aligned to the reference genome.
- model_matcher.cfg
Specify settings of annotations for “model_matcher.py”, for finding novelties in the transcriptome. Each section specifies the annotation parameters for a reference genome. See 7.1 under “Assembly and Analyzing Transcriptome Data” for more details.

```

[hg18]
path: /home/mapper/trans-ABYSS/annotations/hg18
k: knownGene_ref.txt
e: ensGene_ref.txt
r: refGene.txt
a: acembly_ref.txt
x: ensG.txt
order: k,e,r,a

```

```

splice: /home/mapper/trans-ABySS/annotations/
shared/splice_motives.txt

[mm9]
path: /home/mapper/trans-ABySS/annotations/mm9
k: knownGene_ref.txt
e: ensGene_ref.txt
r: refGene.txt
a: acembly_ref.txt
order: k,e,r,a
splice: /home/mapper/trans-ABySS/annotations/
shared/splice_motives.txt

```

Figure 4. Example of “model_matcher.cfg”

- **submitjobs.sh**
 The “submitjobs.sh” script in “utilities” folder is used to run jobs on cluster. The GSC cluster is currently a 400+ core (CPUs) Beowulf-style cluster running Red Hat Enterprise Linux 4. The infrastructure follows the notion of having a headnode which is the point of entry for users to submit and test their jobs – the rest of the cluster are composed of compute nodes which are only involved in the computational side of things. The headnode is called “apollo” and running OSCAR 5.0pre with Sun Grid Engine 6.1u3. To submit jobs internally to GSC cluster, use “submitjobs.sh apollo mqs sub <job_dir> <job_file> <job_name> <job_memory_requirement>”. The script “submitjobs.sh” can be tailored by users to their own way of submitting cluster jobs.

4. Input file

The wrapper scripts take an input file as the argument. The input file specifies the libraries that need to be processed. Each library is specified by four fields in one line: “<library> <ABySS-version> <ABySS-assembly-location> <project>”. The fields are separated by spaces. For example:

```

LIB0001 1.2.1 /projects/ABySS/assemblies/LIB0001 projectA
LIB0002 1.2.1 /projects/ABySS/assemblies/LIB0002 projectA
LIB0003 1.2.1 /projects/ABySS/assemblies/LIB0003 projectA

```

Figure 5. Example input file

Note that library names should be unique in the same input file. To process a library with different parameters (e.g. with different ABYSS-versions), each run should be put in a different input file.

5. Transcript annotations and genome sequence

Trans-ABYSS compares genome alignments of assembled contigs to known annotations to discover novel transcript variants.

Transcript annotation files are downloaded from the UCSC genome browser. Annotation files are organized by reference genome (Fig. 6). Currently trans-ABYSS comes with “hg18” and “mm9” annotations that include Ensembl, UCSC, Aceview and Refseq transcripts.

```
annotations/  
  hg18/  
    genome.fa -> ucsc-hg18.fa  
    knownGene.txt  
    knownGene_ref.txt  
    knownGene_ref.idx  
    knownGene_ref.txt.map  
    ...  
  mm9/  
    ...  
  shared/  
    splice_motifs.txt  
  README
```

Figure 6. Organization of the “annotations” folder.

5.1 Reference Genome

The reference genome fasta file is expected to be either copied or linked to genome folder as “genome.fa”.

5.2 Transcript Annotations

Transcript annotation files (“knownGene.txt”, “ensGene.txt”, “acebmly.txt”) (downloaded from UCSC) need to be modified slightly to include the common gene names at the end of each record (“knownGene_ref.txt”, “ensGene_ref.txt”, “acebmly_ref.txt”). The “refGene.txt” file (also downloaded from UCSC) does not require such processing. The “README” file in the “annotations” folder describes how to do this.

5.3 Indexes (“.idx” files)

The transcript annotation files are indexed by genomic locations to expedite the searching and matching of contigs. Indexing is achieved by running the scripts in the “analysis/annotations” folder, one for each transcript model. For example, to index Ensembl transcripts, run the following command:

```
python ~mapper/trans-ABYSS/analysis/annotations/  
ensembl.py ~mapper/trans-ABYSS/annotations/hg18/
```

```
ensGene.txt -i ~mapper/trans-ABYSS/annotations/hg18/  
ensGene.idx
```

Currently, the following four scripts are supplied: `ensembl.py`, `knownGene.py` (for ucsc known genes), `aceview.py` (aceview genes), `refGene.py` (for refseq genes).

Transcript model files with other formats can be used if you create a custom parser. This can be easily done by modifying any of the existing parsers in the “analysis/annotations/” folder.

5.4 Gene to Transcript Mapping (“.map” files)

A gene-to-transcript mapping file provides simple correspondence between genes and transcripts. It was generated by cutting the 2 columns of “gene name” and “transcript name” from transcript annotation file into a tab-delimited text file (gene<tab>transcript). The “README” file in the “annotations” folder also describes how to generate this file. This mapping file is used to calculate gene coverage (see 7.2 under “Assembling and analyzing transcriptome data”).

5.5 Shared files (under “shared” folder)

This folder contains a “splice_motives.txt” that specifies motives of known splice sites for mammalian genomes. It is used by “align_parser.py” and “model_matcher.py” (see 7.1 under “Assembling and analyzing transcriptome data”) for determining whether a splice site is novel or known.

Assembling and analyzing transcriptome data

1. Trans-ABYSS Pipeline Overview

Because transcriptome samples typically contain transcripts with a wide range of expression levels, and assemblies generated with different k-mer lengths perform differently in capturing transcripts expressed at different levels (see manuscript), we recommend using a wide range of k-mer values to assemble read data from an RNA-seq library (Robertson et al. 2010). Currently, for a read length L , we typically use a range from $L/2$ to $L-1$ for libraries with $L \leq 50$ bp, and a range from $L/2$ to $L-1$, using every other k , for libraries with $L > 50$ bp.

Trans-ABYSS starts with ABYSS multi-k assemblies and processes them into a merged assembly, which is then used to generate alignments and identify novel events and perform various other analyses. Figure 7 shows the overview of trans-ABYSS pipeline.

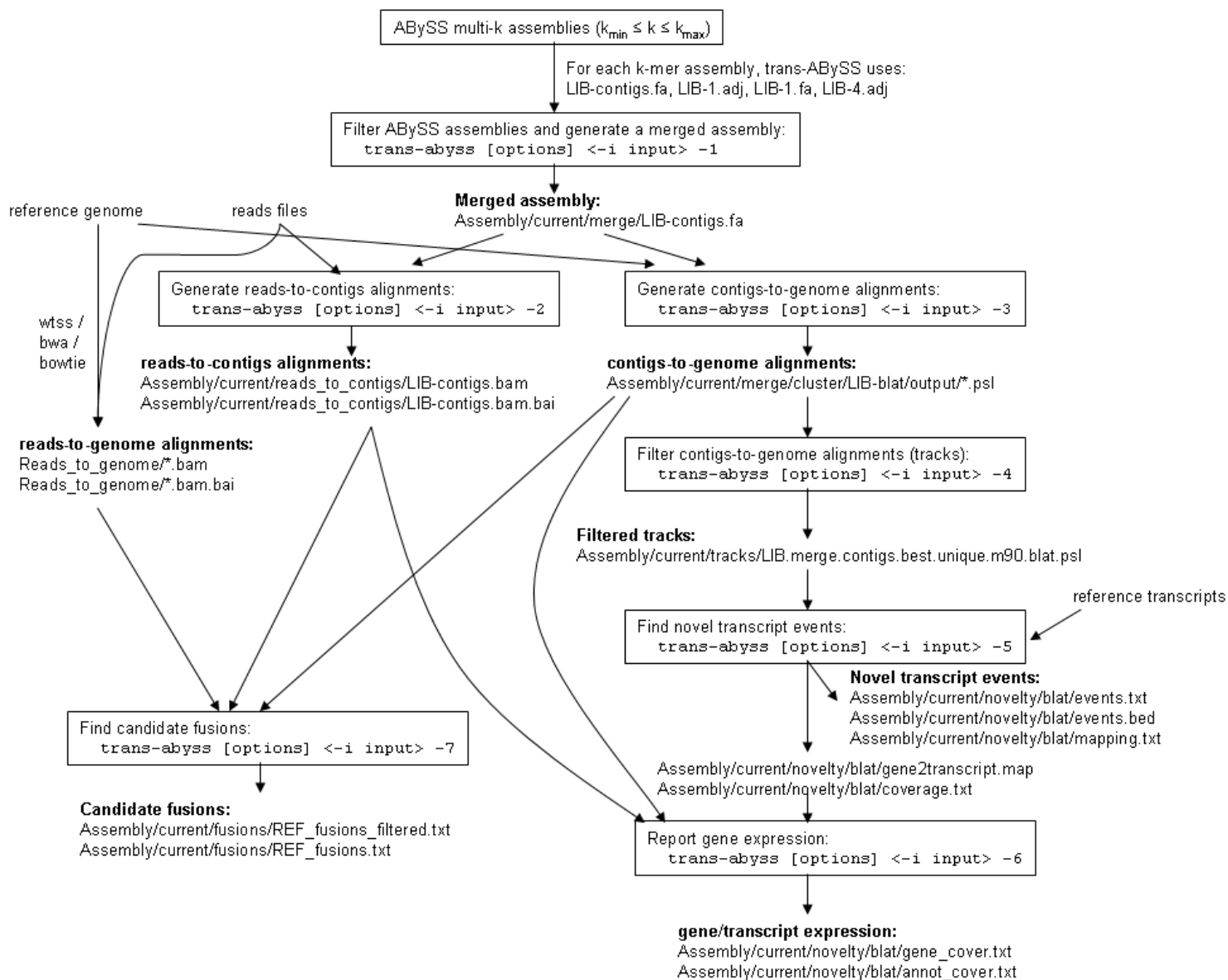


Figure 7. Trans-ABYSS overview

2. ABySS assemblies and folder structure

Trans-ABYSS expects the output from ABySS assemblies of a particular library to be organized as a set of subfolders, each of which is named with “k” followed by the k-mer length (e.g. k26) and contain the ABySS output files from an assembly

using this value of k.

```
Library/  
  k1/  
    [ABySS output files]  
  k2/  
    [ABySS output files]  
  ...  
  in
```

Figure 8. The ABySS assembly folder structure that Trans-ABySS expects. Each 'k' folder holds the output of an ABySS assembly that was generated using that k value. Here, schematic folder 'k' names are shown; typical names might be: k26, k27,

In the above ABySS output parent folder, "in" is a simple text file that list all the paths of all input read files. The read files can be in any of the following formats: bam, qseq, export, or fastq; they can be compressed using gzip or bzip2 (with ".gz" or ".bz2" extensions). The "in" file must be present in the ABySS assembly folder. Its purpose is to facilitate a later step of aligning reads to contigs ("reads_to_contigs.py"). An example "in" file follows.

```
/archive/solexa1_4/analysis2/HS1136/3153YAAXX_2/  
3153YAAXX_2_1_export.txt.gz  
/archive/solexa1_4/analysis2/HS1136/3153YAAXX_2/  
3153YAAXX_2_2_export.txt.gz  
/archive/solexa1_4/analysis3/HS1136/42HVVAAXX_1/  
42HVVAAXX_1_1_export.txt.gz  
/archive/solexa1_4/analysis3/HS1136/42HVVAAXX_1/  
42HVVAAXX_1_2_export.txt.gz  
/archive/solexa1_4/analysis3/HS1136/42HVVAAXX_2/  
42HVVAAXX_2_1_export.txt.gz  
/archive/solexa1_4/analysis3/HS1136/42HVVAAXX_2/  
42HVVAAXX_2_2_export.txt.gz
```

Figure 9. An example "in" file that specifies paths to all input read files.

3. Trans-ABySS folder structure

Trans-ABySS and ABySS have similar working directory structures. The Trans-ABySS folder structure can be set up by running "trans-abyss" or "setup.pl", which is in the "wrappers" folder (see Section 3.3 below). Each k-mer sub-folder should initially be empty, and will be populated by the "setup.pl" script to hold the processed assembly file from the corresponding ABySS k-mer assembly.

```
Project/
```

```

Library/
  Reads_to_genome/
    Reads_to_genome.bam

  Assembly/
    Abyss-1.2.1/
      source -> ABySS assembly path
      k1/
        Library-contigs.fa
      k2/
        Library-contigs.fa
      ...
      merge/
        Library-contigs.fa
      fusions/
      novelty/
      reads_to_contigs/
      tracks/

```

Figure 10. A typical Trans-ABYSS working folder.

The “Reads_to_genome” folder should contain the reads-to-genome alignments in .bam format. See Section 6.3 below for more details.

4. Run Trans-ABYSS Pipeline

The current pipeline can be run with a wrapper script: “trans-abyss” (in the “wrappers” folder). It carries out analysis work in seven stages:

1. generate transcriptome assembly:
 - 1.1 set up Trans-ABYSS folder structure (as in Section 3 above);
 - 1.2 process each ABySS k-mer assembly (see Section 5.1 below);
 - 1.3 merge all k-mer assemblies into one assembly (see Section 5.2 below);
2. align reads to contigs (see Section 6.1 below);
3. align contigs to genome (see Section 6.2 below);
4. generate filtered tracks (see Section 7.1 below);
5. find novel transcript events (see Section 7.1 below);
6. report gene expression levels (see Section 7.2 below).
7. find fusions (see Section 7.3 below).

Each stage depends on the completion of some previous stages. Please refer to the pipeline overview for the workflow (see Figure 7).

To run “trans-abyss”, use the following command:

```
trans-abyss [options] <-i input-file> <-1|-2|-3|-4|-5|-6|-7>
```

where “input-file” is the input file described in Section 4 under “Configuration”. Specify the stage number to run trans-ABySS in corresponding stage: “-1”, “-2”, “-3”, “-4”, “-5”, “-6” or “-7”.

The options are as follows:

- c <CLUSTER_HEAD>
name of the cluster head node to submit jobs (if applicable). For large datasets, it is necessary to run the jobs on cluster.
- s <START_LIB>
start library, i.e. the name of the first library to be processed in the list of libraries in the input file, can be used in combination with “-num” option (see below)
- n <NUM>
number of libraries to process starting from “start library”. Use this option in combination with “-start” option to specify the libraries that need to be processed.
For example, given an input file as shown in [Figure 5], use “-start LIB0002 -num 2” to start from library “LIB0002” and process 2 libraries, i.e. LIB0002 and LIB0003.
On the other hand, if only “-start” option is specified without “-num” option, then all libraries from the start library onwards will be processed.
- l <LIB>
library that needs to be processed (for processing single library). If only a single library needs to be processed, use this option instead of “-start” and “-num” combination.
Running “setup.pl” without “-start”, “-num”, or “-lib” options will process all libraries in the input file.
- h | --help
Print help message.
- version
Print version message.

5. Setting up contigs for analysis

5.1 Process ABySS contigs for each k-mer assembly

For each assembly, a working set of contigs will comprise the following:

- all paired-end contigs;
- all junction contigs; “Junction contigs” are single-end contigs that has exactly two neighbours in the ABySS graph, one at each side. The neighbouring contigs and the junction contig will be merged into one longer contig.
- single-end contigs of length greater than 150bp;

- single-end contigs that are not “islands” and between (2k-1)bp and 150bp. “Islands” are contigs that do not have any neighbours in the ABySS graph.

The contig set can be generated by running “assembly.py” in the “utilities” folder:

```
assembly.py library -d assembly_path/k50 -o k50/library-  
contigs.fa -k 50
```

For each assembly, this subset of contigs for analysis should be stored in the assembly’s k-mer directory and should be named “LIBRARY-contigs.fa”.

5.2 Create the merged assembly

After each assembly is processed, multiple k-mer contig sets are then merged to create a smaller non-redundant contig set. The merging algorithm (“merge.pl”), which is described in the manuscript, uses Blat to perform iterative pairwise alignments between assemblies. The final non-redundant contig set should be in the “merge” sub-folder and should be named, again, “LIBRARY-contigs.fa”.

5.3 Using the wrapper

For convenience, the wrapper script “trans-abyss” is set up to call “assembly.py” and “merge.pl” automatically in its stage “-1” as follows:

```
trans-abyss [options] <-i input-file> -1
```

The options are specified in Section 4 above.

6. Contig and read alignments

6.1 Read alignments to contigs

Read alignments to contigs are required for providing evidence for novel transcript events and for estimating gene-level expression. Trans-ABYSS currently uses Bowtie in single-end mode to perform read-contig alignments. Because contigs can overlap, we allow multi-mapping, but require exact match alignments.

The wrapper script “trans-abyss” can be used to perform reads-to-contigs alignments as follows:

```
trans-abyss [options] <-i input-file> -2
```

6.2 Contig alignments to a reference genome

All the analyses described below (except for polyadenylation sites) require that assembled contigs be aligned to the reference genome. Trans-ABYSS currently supports Blat and Exonerate aligners. However, outputs from other aligners that can generate .psl outputs (e.g. GMAP) can be treated as Blat outputs (by specifying the aligner as “blat” when required) and so can be processed by Trans-ABYSS.

As noted in the manuscript, to minimize the time required to review candidate novel events, it’s important that a contig aligner have a low error rate, and that its error rate be addressed.

Because even after merging there will typically be a large number of contigs, contig alignments are usually performed in parallel on a computer cluster. However, it is inevitable that computing architecture at different laboratories will be different. The “submitjobs.sh” script in “utilities” can be tailored by users for their own way of submitting cluster jobs.

To run BLAT alignments on a cluster, we split the merged assembly file into many smaller files, and run each job independently on the cluster. The current default is to separate the assembly into 1,000 contigs per file. These files and their corresponding cluster job scripts and output can be found in the “merge/cluster/<LIB-blat-dir>” subdirectory in the Trans-ABYSS working directory (please refer to [Figure 9]):

```
merge/cluster/<LIB-blat-dir>/input
merge/cluster/<LIB-blat-dir>/jobs
merge/cluster/<LIB-blat-dir>/output
```

The wrapper script “trans-abyss” can be used to perform contigs-to-genome alignments as follows:

```
trans-abyss [options] <-i input-file> -3
```

It may be possible that some jobs are not finished successfully on the cluster. To check whether all BLAT alignment jobs for a certain library are finished completely, use the tool “check_complete_blat.pl” (supplied in the “utilities” folder) as follows:

```
check_complete_blat.pl <LIB-blat-dir>
```

<LIB-blat-dir> is the name of the directory that holds the inputs, job scripts, and outputs of the blat jobs.

6.3 Aligning reads to a reference genome

In trans-ABYSS v1.0, read alignments to a reference genome are directly used as supporting evidence only to rank fusion gene candidates. A fusion candidate that is well supported by mate-pair read alignments is prioritized for manual review. The v1.0 pipeline does not include code for handling exon-exon junctions for such read alignments. For the results in the publication, we make a BWA-aligned .bam format file, and a .bigWig file derived from it, available for download from the Trans-ABYSS software download page. These were generated with an internal GSC pipeline (unpublished).

7. Transcriptome assembly analysis

Trans-ABYSS offers the following functionality:

7.1 Identify candidate transcript structures that are novel relative to one or more sets of annotated transcript models (e.g. RefSeq, Ensembl, ...).

We recommend filtering contig alignments to retain only the best alignment that is unique (i.e. a contig cannot align to multiple genomic locations with the same score) and covers the majority of the length of the contig (90%):

```
python align_parser.py blat_output_dir/blat_output_file
blat -n 1 -u m90 -d -k "track name" -o filtered.psl -f
merged_contigs.fa
```

The wrapper script “trans-abyss” can also be used to do this job:

```
trans-abyss [options] <-i input-file> -4
```

After filtering, the resulting “filtered” PSL track can be loaded into the UCSC genome browser for review, and be used for mapping to known transcripts for finding novelties:

```
model_matcher.py filtered_track.psl -s genome -m k,e,r,a
-l -d -o output_dir -S merged_contigs.fa -R genome.fa
```

The wrapper script “trans-abyss” can be used to do this job by specifying stage “-5”:

```
trans-abyss [options] <-i input-file> -5
```

The output directory will contain:

1. “mapping.txt” – details the mappings of contigs to known transcripts

2. “events.txt” – reports novel transcript variants relative to all transcript models (e.g. skipped exons, novel exons, ...)

Each line of the “mapping.txt” file reports the following:

<contig> matches <transcript>(<gene>) model:<model abbreviation> (wt:<model weight> in <number aligned blocks> blocks total blocks=<total alignment blocks> total exons=<total number exons> <match> coord:<contig alignment coordinate> score:<score> events:<number of events> coverage:<coverage>

where:

<contig> = contig id

<transcript> = transcript id

<gene> = gene symbol

<model abbreviation> = gene model abbreviation, as specified in configuration file for [model_matcher.py](#) (e.g. ‘k’ = known genes, ‘e’ = Ensembl, ‘r’ = Refseq, ‘a’ = Aceview)

<model weight> = determined by order of models used for matching, which is specified in configuration file for [model_matcher.py](#) (the first model specified will have higher weight, the second model lower weight). This serves as a tie-breaker when contigs are aligned with the same score to different gene models - gene model with the highest weight will be considered the best match

<number aligned blocks> = number of alignment blocks matching exons

<total alignment blocks> = total number of alignment blocks in contig alignment

<total number exons> = total number of exons in transcript

<match> = “full_match”: all edges of alignment blocks aligned (outermost edges not included); “partial_match”: a subset of the total number of block edges aligned; “non_match”: none of the block edges aligned

<contig alignment coordinate> = coordinate of contig alignment in UCSC genome browser format (chr:start-end)

<score> = total number of edges perfect aligned + 0.5 * number splice site variants

<number of events> = number of novel splicing events

<coverage> = number of bases aligned / transcript length

The “events.txt” file reports an event per contig per line in space-delimited columns:

1. event id: the same event shown by different contigs are grouped using event id, e.g. N.1, N.2, N.3 indicates three contigs captured the same event

2. event type:

‘AS3’ = novel 3’ splice site

‘AS5’ = novel 5’ splice site

‘AS53’ = novel 5’ splice site and novel 3’ splice site

‘skipped_exon’ = exon skipping

‘retained_intron’ = retained intron

‘novel_intron’ = novel intron

‘novel_exon’ = novel exon

- 'novel_utr' = novel UTR
 'novel_transcript' = novel transcript
3. transcript id (gene symbol)
 4. alignment block number
 5. exon number: exons are numbered in ascending order of genome coordinate, regardless of transcript orientation
 6. event coordinate: overall event coordinate from start to end
 7. splice-site info, may differ depending on the type of event
 - a) for AS/novel_utr/novel_intron/novel_exon/novel_transcript:
 <splice site sequence>(<motif name>)
 - b) for retained_intron:
 3x:False/True
 - c) for skipped_exon:
 not applicable
 8. surrounding coordinate: event region masked in "--", surrounded by neighboring coordinates e.g. <upstream neighbour start>, <upstream neighbour end>, --, <downstream neighbour start>, <downstream neighbour end>
 9. longest open reading frame: <start 3 amino acids>...<end 3 amino acids>, <number amino acids>aa, <start base number of contig>-<end base number of contig>, <total number bases translated>nt, <fraction contig translated>, <orientation>

7.2 Estimate gene-level expression

Trans-ABYSS maps contigs to annotated transcripts by default. To estimate gene expression levels, you should provide either a gene-to-transcript mapping file, or use a gene annotation file instead of a transcript annotation file.

There are three parts to the calculations for determining gene expression levels:

1. Align reads to contigs. This is part of the standard pipeline. Alignments can be generated by running the wrapper script "trans-abyss" with stage "-2" or by directly running the Python script "reads_to_contigs.py".
2. Map contig alignments to annotated transcripts to generate a 'coverage' file. This can be done by directly running "model_matcher.py" with the "--coverage" option or running the wrapper "trans-abyss" with stage "-5".
3. Run gene_coverage.py to determine gene coverage:

```
python gene_coverage.py coverage-file-from-model_matcher
reads-to-contigs-bamfile track-file-used-for-model_matcher library-
name -g gene-transcript-mapping
```

If contig alignments were mapped to a gene annotation file in step 2, the gene-transcript-mapping file can be omitted. This step can also be run with the wrapper "trans-abyss" as follows:

```
trans-abyss [options] <-i input-file> -6
```

The output of "gene_coverage.py" reports the following:

1. gene name
2. number of reads mapped to gene
3. total read bases mapped to gene
4. union of contig alignment block lengths
5. normalized coverage (column 4 / column 3).

7.3 Identify candidate gene fusion events.

Split genomic alignments of contigs are reported as candidate gene fusion events. To generate a file with such candidate events, run:

```
fusion.py blat_out_dir output -l library -B genomic_bam -  
b contig_bam
```

To minimize time spent in manually reviewing fusion candidates, we recommend filtering outputs on: a) the minimum number of read pairs from read-to-genome alignments, b) the minimum number of spanning reads (from read-to-contig alignments), c) the minimum percentage of identity in the contig-to-genome alignments, etc.:

```
fusion.py output filtered_output -X -F
```

The output of "fusion.py" takes the following format (one candidate fusion event per line):

```
CTG:<ctg id>(<ctg length>bp) TARGET:<region1 target coordinate>,<region  
2 target coordinate> CONTIG:<region1 contig coordinate>,<region 2 contig  
coordinate> <region1 orientaion>,<region2 orientation> TO:<overlap  
target fraction>,CO:<query overlap fraction>,CC:<contig coverage  
fraction>,I1:<alignment 1 identity>,I2:<alignment 2 identity>,AF1:<alignment  
fraction1>,AF2:<alignment fraction2>
```

where:

<overlap target fraction> = fraction of overlap between target regions over sum of target regions aligned

<query overlap fraction> = fraction of overlap between contig regions over sum of contig regions aligned

<contig coverage fraction> = fraction of contig covered by both alignments

<alignment 1 identity> = identity of alignment 2

<alignment 2 identity> = identity of alignment 1

<alignment fraction1> = fraction of contig aligned to region 1

<alignment fraction2> = fraction of contig aligned to region 2

The wrapper script “trans-abyss” can be used to do this job:

```
trans-abyss [options] <-i input-file> -7
```

7.4 Identify candidate SNVs and indels.

ABySS outputs a “bubbles” file (“bubbles.fa”) that contains bubble contigs that represent potential SNVs. Alignments of the bubble contigs to both the genome and the paired-end contig set can be used to report SNVs:

```
python bubble.py k-len bubbles.fa align-genome.psl align-  
contigs.psl blat -o logfile -n 1 -u -m 90 -b align-genome-  
blast-output
```

Genome alignments of contigs can also be mined to extract potential SNVs and indels. Currently, this requires the contigs genome alignment in both psl and blast formats:

```
python align_parser.py blat_output_psl_file blat -n 1 -u m90  
-d -k “track name” -o filtered.psl -f merged_contigs.fa -b  
blat_output_blast_file -v -w output
```

The .snv file generated from the above commands reports the following columns:

1. type: snv (single or multiple bases substitution), ins (insertion), or del (deletion)
2. chr: chromosome
3. chr_start: start coordinate of event
4. chr_end: end coordinate of event
5. strand: strand of alignment
6. ctg: contig id
7. ctg_len: contig length
8. ctg_start: contig start base
9. ctg_end: contig end base
10. len: length of event (bases)
11. change: e.g. G->A, or bases inserted or deleted
12. from_end: shortest distance of event from contig end

Work is in progress to rank candidate SNVs and indels by using read alignments as evidence.

7.5 Identify candidate polyadenylation sites.

Polyadenylation site candidates are detected using a combination of Perl scripts,

the BWA read aligner and UNIX commands. To perform the basic operations, a configuration file needs to be set up which points to the locations of BWA and the transcript models. The wrapper scripts 'polyareads.pl' and 'polyafinder.pl' can run the necessary commands.

The 'polyascripts' folder under the 'analysis' folder has this structure

```
bin
  Perl scripts
conf
  polyafinder.conf
  DISCLAIMER.txt
  README.txt
```

The wrapper script requires as input: raw Illumina reads, gene transcript models in FASTA format, and contig FASTA sequences from the assembly pipeline.

As described in the publication's Supplementary Information, the method uses two types of reads: PAM and EJ.

Run the wrapper script to extract PAM reads from the read files:

```
polyareads.pl -p -f <FORWARD_READS> -r <RIGHT_READS> [-F
<FORWARD_READS2> -l ...] [-r <RIGHT_READS2> -r ...] [-conf
<CONFIGFILE>]
```

Perform the PAM read alignment:

```
polyafinder.pl -f <FORWARD_PAM> -r <REVERSE_PAM> -t
<TRANSCRIPT> [-t <TRANSCRIPT>] -a [-cont <CONTIGFILE> -
capp]
```

Extract EJ reads from the read files:

```
polyareads.pl -e -f <FORWARD_READS> -r <RIGHT_READS> [-F
<FORWARD_READS2> -l ...] [-r <RIGHT_READS2> -r ...] [-conf
<CONFIGFILE>]
```

Perform the EJ read alignment:

```
polyafinder.pl -e -a -f <FORWARD_EJ> -r <REVERSE_EJ> -t
<TRANSCRIPT> [-t <TRANSCRIPT>] [-mf <FORWARD_EJ_MATE> -mr
<REVERSE_EJ_MATE>]
```

Optionally, it is possible to create visual images of the reads mapped to the genome by creating .bed files with `getremappedBED.pl` or `samse2bed.pl`, .wig files with `bed2Wig.pl`. Both can be used as viewable tracks within the UCSC

genome browser. To summarize high-hit alignments and generate genome browser URLs, use the script `getpolyTmapcoord_extracols.pl`.

Align reads to genome by either:

```
cat <INPUT_READS> | getremappedBED.pl <OUT_UNMAPPED_READS>
<INPUT_FASTQ > <OUTPUT_BED>
```

or (if no raw reads available)

```
bwa aln <GENOME_FA> <INPUT_FASTQ> > <OUTPUT_SAI>
bwa samse -n 20 <GENOME_FA> <OUT_SAI> <INPUT_FASTQ> |
samse2bed.pl > <OUT_BED>
```

Then, convert to .wig

```
cat <OUT_BED> | bed2Wig.pl > <OUT_WIG>
```

To rank the transcripts and to create UCSC linked URLs:

```
getpolyTmapcoord_extracols.pl -t <IN_TSV> [-b <OUT_BED>]
[-z <ZOOM>] [-c CUTOFF1,C2,C3,...,Cn] [-u 'http://
/genome.ucsc.edu/cgi-bin/hgTracks?
org=<ORGANISM>&db=<DB>&position='] [-o] [-d 500] >
<RANK_OUT>
```

To limit the amount of false positives due to high poly-A and poly-T regions, use the script `calcPolyAinSeqs.pl` for the transcript and contig sequences, and use the script `findgenomicpolya.pl` for the genome.

For more detailed information on each script, please refer to their respective perdocs, and the README in the “polyascripts” folder under the “analysis” folder.

Data sets

1. Large

MM0472 library at SRA (Short Read Archive). This library can be downloaded from <http://www.ncbi.nlm.nih.gov/sra/SRX017642?report=full>

Once downloaded, run ABySS with multi-k (we run every k from 26 to 50) and then use trans-ABYSS to process the assemblies and perform analyses. The reads to genome (mm9) alignments can be downloaded from Trans-ABYSS software release web page.

2. Small

2.1 Insr_UTR

We include a dataset that consists of all 15024 reads that aligned with Bowtie to

contigs that the pipeline matched to annotated insulin receptor gene, *Insr*. Trans-ABySS identified an exon in this gene that was novel when we discovered it, but was subsequently included in 'UCSC gene' transcript models for this gene.

This set of reads can be assembled with ABySS, and processed and analyzed with Trans-ABySS on a single CPU in less than 2 hours. All data/results are stored in "sample_date/*Insr*_UTR" folder and can be used for testing. You should be able to duplicate the results when running trans-ABySS on your own machine.

2.2 Polyadenylation site analysis

We include a small dataset that contains 34 PAM reads in FASTQ format, 2 genes in FASTA format, and a contig sequence in FASTA format. The PAM reads are chosen to illustrate the ability to find novel polyadenylation sites. Specifically, the two gene examples show novel short 3' UTR while the contig shows a lengthened 3' UTR.

The wrapper scripts `polyareads.pl` and `polyafinder.pl` can be used to extract and align the reads to the transcripts.

References

Birol I, Jackman SD, Nielsen CB, Qian JQ, Varhol R, Stazyk G, Morin RD, Zhao Y, Hirst M, Schein JE, Horsman DE, Connors JM, Gascoyne RD, Marra MA, Jones SJ. **De novo transcriptome assembly with ABySS**. *Bioinformatics*. 2009 Nov 1;25(21):2872-7.

Robertson G, Schein J, Chiu R, Corbett R, Field M, Jackman SD, Mungall K, Lee S, Okada HM, Qian JQ, Griffith M, Raymond A, Thiessen A, Cezard T, Butterfield Y, Newsome R, Chan SK, She R, Varhol R, Kamoh B, Prabhu A-L, Tam A, Zhao Y-J, Moore R, Hirst M, Marra MA, Jones SJM, Hoodless PA, Birol B. **De novo Assembly and Analysis of RNA-seq data**. *Nature Methods*, *in press*.

Simpson JT, Wong K, Jackman SD, Schein JE, Jones SJ, Birol I. **ABySS: a parallel assembler for short read sequence data**. *Genome Res*. 2009 Jun;19(6):1117-23.