Trans-ABySS v1.0.1: User Manual

2 November 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

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 v1.0.1 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

ABySS can be compiled to run on any POSIX-compliant system. Use the following commands to read ABySS man pages:

```
man doc/abyss-pe.1
man doc/ABYSS.1
```

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

Installation

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.

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, 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 <= 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 a set of ABySS assemblies for a range of *k* values. It processes them into a merged assembly, which is then used to generate alignments, identify novel events and perform other analyses. Figure 1 shows an overview of the pipeline.

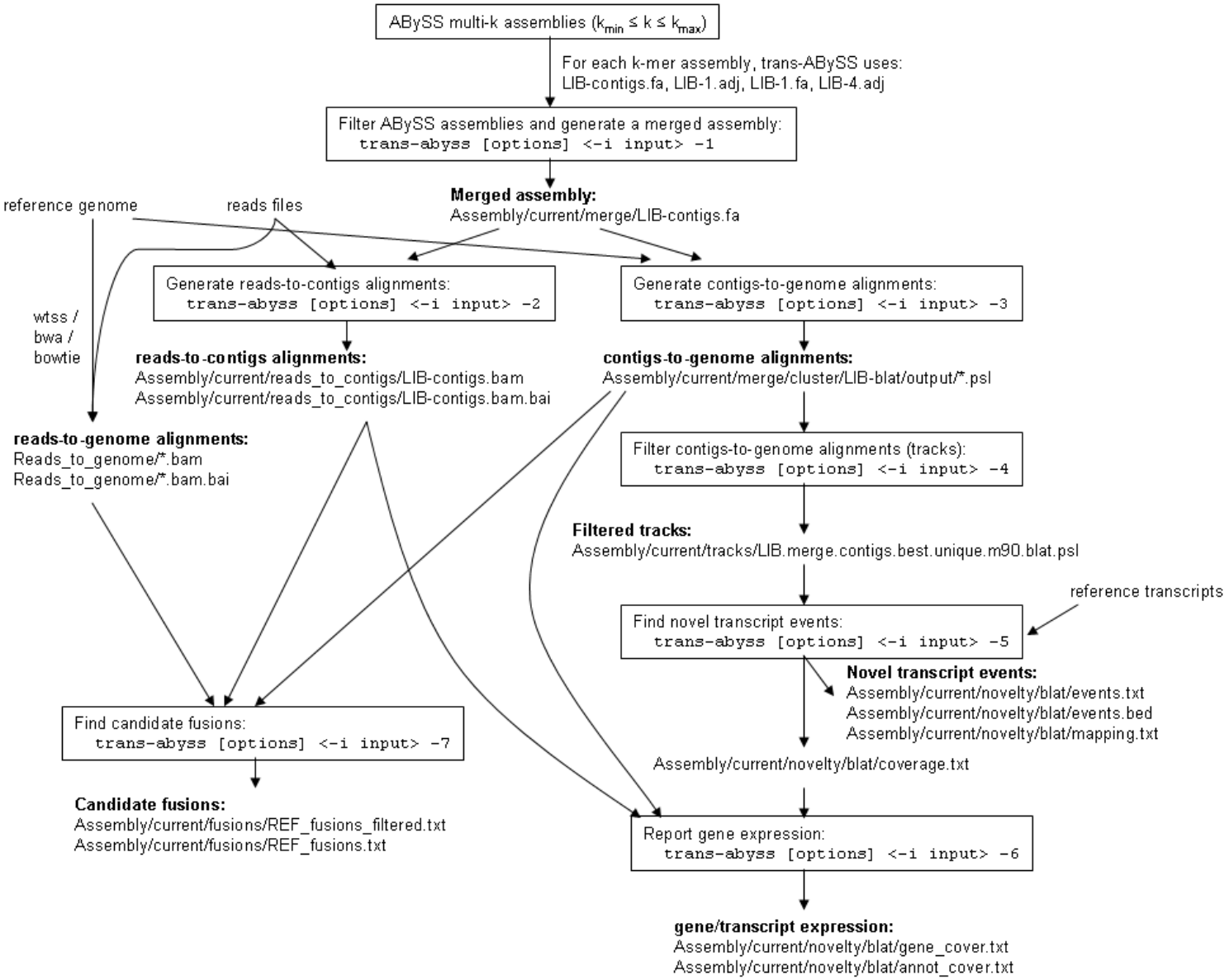

Figure 1. Trans-ABySS pipeline overview.

2. ABySS assemblies and folder structure

Trans-ABySS expects the output from ABySS multi-k assemblies for each library to be organized as follows: a single parent folder is used to hold all k-assemblies, where each subfolder is named "k*n*" (*n* is the value of k, e.g. k35) and stores the ABySS assembly output files for that particular k value (Fig. 2). In addition,

a simple text file named "in" should be present in the ABySS assembly folder, which lists all the paths of all input read files.

```
LIB0001/
      k1/
            LIB-contigs.fa
            LIB-1.adj
            LIB-1.fa
            LIB-4.adj
            [other ABySS output files]
      k2/
            LIB-contigs.fa
            LIB-1.adj
            LIB-1.fa
            LIB-4.adj
            [other ABySS output files]
      …
      in
```

Figure 2. 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, ...

The read files specified by the "in" file 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). Fig. 3 shows an example "in" file.

```
/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 3. An example "in" file that specifies paths to all input read files.

Once the "in" file is created, ABySS can be run with multiple *k* values on the same set of read files. For convenience, an example shell script "run-abyss" is included in Trans-ABySS "utilities" folder, which demonstrates how to generate ABySS multi-k assemblies in the required folder structure:

```
for k in {26..49}; do mkdir k$k; cd k$k; abyss-pe
```

```
in=`paste -sd' ' in` OVERLAP_OPTIONS=--no-scaffold
SIMPLEGRAPH_OPTIONS=--no-scaffold E=0 n=10 v=-v; cd ..;
done
```

The following are example scripts to run ABySS on a computer cluster and generate multi-k assemblies in the required folder structures using qsub:

Script: `utilities/qsub-l50-64`

```
#!/bin/sh
set -eu
qsub -N `basename $PWD` -t 33-49 ../qsub-l50-k64
```

Script: `utilities/qsub-l50-k64`

```
#!/bin/env qsub
#$ -q mpi.q
#$ -pe openmpi-1.3.1 16
#$ -l hostname=qn*
##$ -l mem_used=300M
setenv PATH [PATH-TO-ABYSS-BINARIES]:$PATH
setenv in `paste -sd' ' in`
mkdir k$SGE_TASK_ID && cd k$SGE_TASK_ID && \
     abyss-pe OVERLAP_OPTIONS=--no-scaffold
SIMPLEGRAPH_OPTIONS=--no-scaffold E=0 n=10 v=-v
```

For more help on how to generate ABySS multi-k assemblies or other ABySS-related problems, please refer to the ABySS help group at http://groups.google.com/group/abyss-users.

3. Trans-ABySS folder structure

Trans-ABySS and ABySS have similar working directory structures (Fig. 4). The Trans-ABySS folder structure can be set up by running "`trans-abyss`" or "`setup.pl`", both of which are 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. The other directories are used to hold various Trans-ABySS output files, which will be discussed in detail in the following sections.

```
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 4. A typical Trans-ABySS working folder.

4. Running the Trans-ABySS Pipeline

4.1 Set up 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 (in "`configs`" folder) to run the pipeline:

- projects.cfg
  For each project, the user specifies the reference genome and the top-level 'project' directory (Fig. 5). 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 5. Example organization of “projects.cfg”

- binaries.cfg
  Paths to external software are specified in “software: path” format, with one line for each executable (Fig. 6). 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 6. Example of “binaries.cfg”

- cluster.cfg
  Specify cluster job settings including the memory requirement for running different scripts on cluster, and the file used for each reference genome (FIg. 7). This file is required when running jobs on a 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 7. 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 novel transcripts and transcript events, relative to reference transcript annotations (Fig. 8). Each section specifies the annotation files and their order for a reference genome. See section 4.4.3A for more details.

```
[hg18]
k: knownGene_ref.txt
e: ensGene_ref.txt
r: refGene.txt
a: acembly_ref.txt
x: ensg.txt
order: k,e,r,a

[mm9]
k: knownGene_ref.txt
e: ensGene_ref.txt
r: refGene.txt
a: acembly_ref.txt
order: k,e,r,a
```

Figure 8. Example of "model_matcher.cfg"

- submitjobs.sh
  The "`submitjobs.sh`" script in the "`utilities`" folder is used to run jobs on a computer cluster. The GSC cluster is currently a ~2000+ core (CPUs) Beowulf-style cluster running Red Hat Enterprise Linux 4. The infrastructure consists of a headnode to which users submit jobs, and the rest of the cluster consists of compute nodes that are involved only in computation. The headnode, called "apollo", runs OSCAR 5.0pre with Sun Grid Engine 6.1u3. We submit jobs to apollo with this command:

  ```
  submitjobs.sh apollo /opt/mqtools/bin/mqsub <job_dir>
  ```

```
<job_file> <job_name> <job_memory_requirement>
```

Please adjust "`submitjobs.sh`" to be appropriate to your cluster.

4.2 Set up transcript annotations and genome sequence

Trans-ABySS compares genome alignments of assembled contigs to known annotations to discover transcript variants that are novel relative to reference transcript annotations.

Transcript annotation files are downloaded from the UCSC genome browser. Annotation files are organized by reference genome (Fig. 9). Currently trans-ABySS comes with "hg18" and "mm9" annotations (under "`annotations`" folder) that include Ensembl, UCSC, Aceview and Refseq transcripts.

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

Figure 9. Organization of the "annotations" folder.

4.2.1 Reference Genome

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

4.2.2 Transcript Annotations

Transcript annotation files ("`knownGene.txt`", "`ensGene.txt`", "`acembly.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.

4.2.3 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.

4.2.4 Splice motifs

The "`splice_motives.txt`" file specifies motives of known splice sites for each genome. It is used by "`align_parser.py`" and "`model_matcher.py`" (see Section 4.4.3A) for determining whether a splice site is novel or known.

There is also a shared "`splice_motives.txt`" file (under "`shared`" folder) that can be used as common splice motifs, and the "`splice_motifs.txt`" file in each reference genome can be a symbolic link to this file.

4.3 Set up the input file

The wrapper scripts take an input file as the argument (Fig. 10). 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. The <ABySS-assembly-location> is the name of the parent folder that holds all ABySS k-assemblies for that library, where each k-assembly is in a subfolder kn (see Fig. 2).

```
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 10. 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.

4.4 Running trans-ABySS

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 the transcriptome assembly
    1.1 set up the Trans-ABySS folder structure (as in Section 3 above);
    1.2 process each ABySS k-mer assembly (see Section 4.4.1A below);
    1.3 merge all k-mer assemblies into one assembly (see Section 4.4.1B below);
2. align reads to contigs (see Section 4.4.2A below);
3. align contigs to the genome (see Section 4.4.2B below);
4. filter contigs-to-genome alignments and generate a track file that can be loaded to UCSC browser as a custom track (see Section 4.4.3A below);
5. find candidate novel transcript events (see Section 4.4.3A below);
6. report gene expression levels (see Section 4.4.3B below).
7. find candidate fusion genes (see Section 4.4.3C below).

Each stage depends on the completion of previous stages. Please refer to the pipeline overview for the workflow (Fig. 1).

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 4.3 above. 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 a 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 10], use "-start LIB0002 -num 2" to start from library "LIB0002" and process 2 libraries, i.e. LIB0002 and LIB0003.
If only "-start" option is specified, and "-num" is not specified, 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 "trans-abyss" without "-start", "-num", or "-lib" options will process all libraries in the input file.

-h | --help
Print help message.

--version
Print version message.

4.4.1 Setting up contigs for analysis

A. Processing ABySS contigs for each k-mer assembly

For each assembly, a working set of contigs will comprise the following:
- all paired-end contigs; "Paired-end contigs" are contigs that were assembled during the pair-end stage of ABySS (Simpson et al. 2009).
- all junction contigs; "Junction contigs" are single-end contigs with length between *k* and *2k-2*. Each has exactly two neighbours (one on each side) in the ABySS assembly graph. A junction contig and its two neighbouring contigs will be merged into one longer contig.
- single-end contigs of length greater than 150bp;
- single-end contigs that are not "islands" and are between *(2k-1)*bp and 150bp in length. "Islands" are contigs that have no 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
```

Running this command for each ABySS k-mer assembly will generate a single FASTA file (named "LIBRARY-contigs.fa") in the corresponding trans-ABySS k-mer directory. This filtered contig set will be used for all downstream analysis.

B. Creating the merged assembly

After each assembly has been processed, sets of contigs for assemblies across a range of k values are then merged to create a smaller, non-redundant contig set. The merging algorithm ("`merge.pl`"), which is described in the manuscript (Robertson et al. 2010), uses Blat to perform iterative pairwise alignments between assemblies. The final result is another FASTA file (also named "LIBRARY-contigs.fa", but under the "`merge`" sub-folder) that consists of the non-redundant contig set from all k-mer assemblies.

C. 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 4.4 above. This command will run through all k-mer assemblies and merge the results into the final FASTA file.

4.4.2 Contig and read alignments

A. Read alignments to contigs

Read alignments to contigs are required for providing evidence 'support' 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
```

B. 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 (Robertson et al. 2010), to minimize the time required to review candidate novel transcript events, it is 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. Computing systems at different laboratories will differ, and the "`submitjobs.sh`" script in "`utilities`" should be tailored by users to suit their cluster configuration.

To run BLAT alignments on a cluster, we split the merged assembly file into many smaller files, and run each job independently. 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 (Fig. 4):

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

The "`trans-abyss`" wrapper script can be used to perform contig-to-genome alignments as follows:

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

Some alignment jobs may not finish 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.

C. Aligning reads to a reference genome

Mate-pair read alignments to a reference genome are directly used as supporting evidence to rank fusion gene candidates. A fusion candidate that is well supported by mate-pairs will be prioritized for manual review.
The v1.0.1 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 v1.0 download page. These were generated with an internal GSC pipeline (unpublished).

4.4.3 Transcriptome assembly analysis

Trans-ABySS currently offers the following functionality.

A. 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 (e.g. 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 be used to run this job:

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

After filtering, the resulting PSL-format file can be loaded into the UCSC genome browser for review, and can also be compared to reference transcript model files (e.g. in UCSC gene table format) by the "`model_matcher.py`" script in order to find novel transcripts and transcript events in the contig alignments:

```
model_matcher.py filtered_track.psl genome -l -d -o
output_dir -f merged_contigs.fa -r
```

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, …)
3. "events.bed" – novel transcript variants in bed format
4. "coverage.txt" – transcript coverage statistics

The "mapping.txt" file is a text file where each line reports a match between a contig and a transcript, for example:

k38:11 matches uc009krd.2(Insr) model:k(wt:4) in 1 blocks total_blocks=1 total_exons=21 partial_match coord:chr8:3154550-3154852 score:2.0 events:0 coverage:0.032

The format of each line is:
<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 "configs/model_matcher.cfg" file (the models are specified in the order from highest to lowest weight). This serves as a tie-breaker when contigs are aligned with the same score to different gene models, in which case the 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 perfectly aligned + 0.5 * number of 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, for example:

1.1 novel_exon k40:5-,2+,8- uc009krd.2(Insr) 11,12 12 chr8:3181702-3181737 36 3174889,3181701,--,3181738,3184950 orf:AGV...NPS,1399aa,162-4361,4199nt,0.88,1

The columns in each line are as follows:
1. event id: the same event shown by different contigs are grouped using event id, e.g. for event N, N.1, N.2, N.3 indicates that 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>

The "events.bed" file contains several UCSC-format .bed tracks with each track representing one type of novel event (listed in 2, above). Each line represents one event. For reviewing novel event predictions, it is helpful to load this file into the UCSC genome browser, along with the contig alignments.

The "coverage.txt" is a tab-delimited txt file that reports the coverage of individual transcripts by contigs, for example:

```
InsrandD630014A15Rik.cSep07	InsrandD630014A15Rik	219	547	k26:8	0.043	7	k33:2,k45:18,k42:8,k35:13,k38:11,k26:8,k44:20	0.400	8.2
```

The columns are as follows:
1. transcript
2. gene
3. total_coverage - total number of bases of transcript covered by contig
4. transcript_length - transcript length in base pairs
5. best_contig - best contig covering the transcript in terms of bases covered
6. best_contig_coverage - coverage of best contig
7. nbr_contigs - number of contigs covering the transcript
8. contigs - list of contigs covering the transcript

9. coverage - total bases covered (column 3) divided by transcript length (column 4)
10. normalized_k_coverage_best_contig - k-mer coverage divided by contig length

B. Estimate gene-level expression

Trans-ABySS maps contigs to reference annotated transcripts by default. Gene-level expression is estimated by mapping to the gene the coverage on contigs aligned to a gene's transcripts. For the mouse adult liver data described in the Nature Methods publication (Robertson et al. 2010), trans-ABySS expression values correlated closely to those from ALEXA-seq (Griffith et al. Nat Methods. 2010 7(10):843-7).

There are three parts to the calculations:

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`" 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-contgs-bamfile track-file-used-for-model_matcher libary-
name output-file
```

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`" is stored in "gene_cover.txt" as a tab-delimited text file with the following 5 columns:
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 3 / column 4).

C. 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 wrapper script "`trans-abyss`" can be used to do this job, which is equivalent to running the above two "`fusion.py`" commands:

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

The output of "`fusion.py`" is a space-delimited txt file that reports one candiate fusion event per line, for example:

CTG:k30:253500+,938187+,1140721-(7146bp) TARGET:chr4:13210525-13238093,chr4:13209979-13210251 CONTIG:1-6881,6874-7146 -,+ TO:0.00,CO:0.00,CC:1.00,I1:100.0,I2:100.0,AF1:0.96,AF2:0.04 READPAIRS:1001 SPAN_READS:2

Each line is in the following format:

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

4.4.4 Additional Trans-ABySS Functions

Trans-ABySS provides some additional functions that are currently not handled by the wrapper scripts, but can be run separately.

A. 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 (note that BLAT is able to output in blast format):

```
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. This functionality will be made available in a future version of the pipeline.

B. Identify candidate polyadenylation sites.

Polyadenylation site candidates are detected using a combination of Perl scripts, the BWA short-read aligner and UNIX commands. To perform the basic operations, a configuration file needs to be set up that points to the locations of BWA and the reference transcript models (Refseq, Ensembl, etc. sequence files

in FASTA). The wrapper scripts '`polyareads.pl`' and '`polyafinder.pl`' can run the necessary commands.

Below, we outline the workflow. For more detailed information on each script, please refer to their respective perldocs, and to the README in the "`polyascripts`" folder under the "`analysis`" folder.

The "`analysis`/`polyascripts`" folder has this structure

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

The wrapper script requires as input: raw Illumina read sequence files (or FASTQ files), eference transcript sequences in FASTA format, and contig FASTA sequences from the assembly pipeline.

As described in the publication's Supplementary Information (Robertson et al. 2010), the method uses two types of reads: paired-end mate (PAM) and end-junction (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 to reference transcript sequences:

```
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 to reference transcript sequences:

```
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 that facilitate reviewing predictions at UCSC, use the script `getpolyTmapcoord_extracols.pl`.

Align reads to the genome by either extracting from the raw Illumina read file:

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

or if no raw Illumina reads are available, use FASTQ to align to genome:

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

Then, convert the alignments to .wig:

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

To rank the transcripts by number of PAM/EJ reads mapped 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 number 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.

Data sets

1. Large

MM0472 library at the SRA (Short Read Archive). This 147 M read PE dataset can be downloaded from http://www.ncbi.nlm.nih.gov/sra/SRX017642?report=full

Once downloaded, run ABySS with multiple *k* values (we ran every *k* from 26 to 50) and then use trans-ABySS to process the assemblies and perform analyses. The reads-to-genome (mm9) alignments and bigWig files can be downloaded from Trans-ABySS software v1.0 release web page.

2. Small

2.1 Insr_UTR

We include a dataset that consists of all 15,024 reads that aligned with Bowtie to contigs that the pipeline matched to annotated insulin receptor gene, Insr, including its UTR regions. Trans-ABySS identified an exon in this gene that was novel when we discovered it, but was subsequently (but temporarily) 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_data/Insr_UTR" folder and can be used for testing. You should be able to duplicate the results by running trans-ABySS on your own machine.

2.2 Polyadenylation site analysis

In the data_pam folder, we include a small dataset that contains 25 PAM reads in FASTQ format, 4 genes in FASTA format, and 2 contig sequence in FASTA format. The PAM reads were chosen to illustrate finding novel polyadenylation sites. Specifically, the two 'gene' examples show candidate novel short 3' UTRs while the 'contig' example shows a candidate novel lengthened 3' UTR.
In the data_ej folder, we include 13 EJ reads in FASTQ format, and 1 gene in FASTA format. Similarly, these were chosen to illustrate the detection of novel short polyadenylation site.

The wrapper scripts `polyareads.pl` and `polyafinder.pl` can be used to extract and align the reads to the transcripts. Refer to cmd.txt for commands that can be used. cmd.txt can also be run as a shell script.

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.