

# SOLiD™ WT Analysis Pipeline Documentation

<b>1</b>	<b>OVERVIEW</b>	<b>3</b>
<b>2</b>	<b>INSTALLATION</b>	<b>4</b>
2.1	Requirements	4
2.2	Installation Procedures	4
2.3	Testing the Installation	4
<b>3</b>	<b>USAGE</b>	<b>5</b>
3.1	Common Parameters	7
<b>4</b>	<b>SPLIT READ-MAPPER PROGRAM</b>	<b>8</b>
4.1	Overview	9
4.2	Usage Examples	12
4.3	Parameters	12
4.4	System Input files	15
4.5	System Output Files	15
<b>5</b>	<b>COUNT TAGS PROGRAM</b>	<b>17</b>
5.1	Overview	17
5.2	Usage	17
5.3	Parameters - Required	18
5.4	System Input Files	19
5.5	System Output Files	19
<b>6</b>	<b>NTR FINDER PROGRAM</b>	<b>20</b>
6.1	Overview	20
6.2	Usage	21
6.3	Parameters	22
6.4	System Input files	25

6.5	System Output files .....	25
A	APPENDIX 1 GTF FILE FOR DEFINING GENE ANNOTATIONS .....	29
B	APPENDIX 2 MAX FILE FORMAT .....	30

1.0	7/10/2009	
-----	-----------	--

## 1 Overview

The SOLiD™ Whole Transcriptome Analysis Pipeline (WTAP) is a collection of programs for analyzing reads obtained from a SOLiD™ Transcriptome experiment. Principle WTAP programs include:

1. `split_read_mapper.sh`: A mapping tool that extends the functionality of mapreads to better map transcript reads.
2. `count_tags.pl`: A counting tool for counting the number of reads within specified genomic features.
3. `ntr_finder.sh`: A transcribed-region detection tool for identifying non-annotated regions of active transcription in a genome.

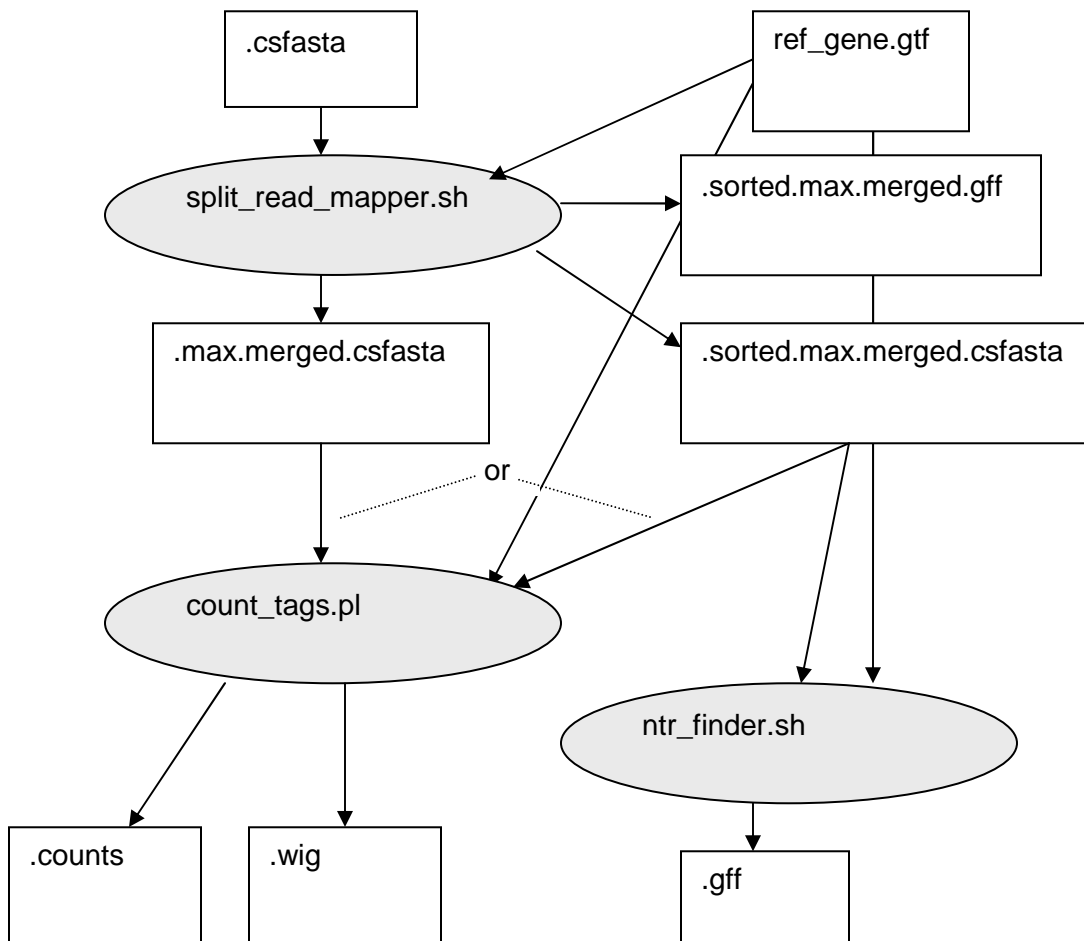


Figure 1 Principal programs (ovals) and files (rectangles).

## 2 Installation

### 2.1 Requirements

This software requires a compute cluster. Each program is executed on a head or staging node. Some programs will send jobs to the scheduler for processing on worker nodes. This cluster must have the following:

- PBS, LSF or SGE Scheduling System
- 64-bit Linux System on every node
- Minimum of 4 GB available RAM on every node.  
The RAM requirement for `split_read_mapper.sh` is determined by the largest sequence in your reference genome. Use this formula to estimate the requirement:

$$1.2 \text{ GB} + 6 \text{ bytes} * \text{sequence length}$$

- Java 1.6 64-bit Virtual Machine on every node
- Python v2.3 on worker nodes. Python 2.3 must be installed as the first 'python' executable in the user's \$PATH.
- Perl v5.8.5 on the head node. Perl 5.8 should be installed as `/usr/bin/perl`. If not, Perl scripts in `/bin` must be edited to change the shebang (`#!`) paths to the location of the Perl 5.8 interpreter.
- 500 or more GB disk space.

### 2.2 Installation Procedures

To install the pipeline follow these steps:

1. Unzip the contents of the `WTAP_<version>.tar.gz` package into the desired location. Enter the following command:

```
$ tar xzvf WTAP_<version>.tar.gz
```

In the rest of this document, this location will be referenced to as `$WTAP_HOME`.

2. Enter the subdirectory created and run make:

```
$ cd WTAP_<version>
$ make
```

The pipeline is now ready for use.

### 2.3 Testing the Installation

We provide a small test dataset on <http://www.solidsoftwaretools.com> with which you may test your WTAP installation.

1. Download and extract the test dataset from <http://www.solidsoftwaretools.com>.
2. Edit the following parameters in `mapping_config.ini` and `ntr_config.ini` for your system.
  - `queue.sys`
  - `queue.sys.queue`

- `queue.sys.resource.string` (if you are using LSF or SGE schedulers).
  - `queue.sys.nodes.tmp.dir`
  - `wt.file.reference *`
  - `wt.output.dir`
3. Edit the following parameters in `mapping_config.ini`:
    - `wt.max.memory.per.job`
    - `wt.mapper.filter.file.reference *`
    - `wt.mapper.file.reads *`
    - `wt.exon.reference *`
  4. Edit the following parameters in `ntr_config.ini`:
    - `wt.ntr.finder.max.file *`
    - `wt.ntr.finder.file.attr.reference *`

\*These parameters point to files in the test dataset. Partial paths are provided in the .ini files. Please complete these paths to the full paths of these files on your system.
  5. Run `split_read_mapper.sh` to map the reads in the test dataset by entering the following command:

```
% split_read_mapper.sh -c <path to mapping_config.ini>
```
  6. The output directory will contain the file `output/alignmentReport.txt`. It should match the file in the test dataset `mapping_output/alignmentReport.txt` file.
  7. Run the `ntr_finder` command by entering the following:

```
% ntr_finder.sh -c <path to ntr_config.ini>
```

The results should match the contents of `ntr_output/` in the test dataset.
  8. Test `count_tags.pl`. The test dataset contains the file `count_tags.sh` which defines the command for testing `count_tags.pl`. You need to change the following file parameters to match file locations on your system:
    - `-max`
    - `-r`
    - `-a`
    - `-counts`
    - `-wig`
    - `-gff_reads`

Compare the results with the contents of `count_tags_output/` in the test dataset.

### 3 Usage

The programs in WTAP require many user definable parameters. The parameters for `split_read_mapper.sh` and `ntr_finder.sh` may be defined as command line switches or in configuration files. Parameters for `count_tags.pl` can **only** be passed as command line switches.

Configuration files define parameters as key-values pairs of the following format:

```
key=value
```

- Some options accept multiple values. Multiple values should be separated with commas.
- Lines starting with “#” are ignored as comments.

Examples of a Configuration file entry:

```
queue.sys=pbs  
my.key=multiple, values
```

Configuration files may occupy multiple locations.

Parameters may be defined in the following files:

1. \$WTAP\_HOME/etc/config.ini
2. ~/.ab\_wtp/config.ini
3. A configuration file passed via the command line with the `-c` switch.

This allows setting site, user and individual command level parameters. The order of precedence among parameter sources is as follows, from highest to lowest:

1. switches
2. `-c` config file
3. ~/.ab\_wtp/config.ini
4. \$WTAP\_HOME/etc/config.ini

Parameters defined in higher precedence sources **override** parameters defined in lower precedence sources.

Example: Suppose the site configuration `$WTAP_HOME/etc/config.ini` defines: `wt.mapper.read.length=50` because most runs with WTAP use 50mer reads. However, you need to analyze some 35mer data. You may override this site setting by specifying a read length of 35 by:

- Entering `-l 35` on the command line, or
- Placing `wt.mapper.read.length=35` in a configuration file to be passed with the `-c` switch.

With parameters defined in several locations, it can be difficult to determine the values of all parameters that will be applied when you actually execute a program. For this reason, we provide the `-p` switch. Running a program with the `-p` switch will simply list the values of the parameters and terminate the program. Use `-p` to verify that your configuration is correct. The values of all parameters are always written to the log file.

Examples of running WTAP tools:

Running `split_read_mapper.sh` using parameters defined in a config file:

```
% split_read_mapper.sh -c myconfig.ini
```

Running `split_read_mapper.sh` with most parameters defined in site or user configuration files and some the command line:

```
% split_read_mapper.sh -A myreads.csfasta -o myoutput.dir
```

### 3.1 Common Parameters

A number of parameters and their corresponding configuration file keys are common between the Split Read Mapper and the NTR Finder applications.

<b>Config File Key:</b>	<code>queue.sys</code>
<b>Command-Line Switch</b>	<code>--queue-sys, -e</code>
Define the scheduling environment, should be <code>pbs</code> , <code>lsf</code> or <code>sge</code> . Formerly: <code>SCHEDULING_ENVIRONMENT</code>	
<b>Config File Key:</b>	<code>queue.sys.queue</code>
<b>Command-Line Switch</b>	<code>--queue-sys-queue, -q</code>
Name of the scheduler queue to use. Formerly: <code>NAME_OF_QUEUE</code>	
<b>Config File Key:</b>	<code>queue.sys.resource.string</code>
<b>Command-Line Switch</b>	<code>--queue-sys-resource-string, -u</code>
Specifies the resource requirement string for the scheduling command. This corresponds to the <code>qsub -l</code> option in PBS and the <code>bsub -R</code> option in LSF.  The memory requirement portions of this parameter can be calculated dynamically by the pipeline. The following tokens are substituted with calculated values when used with the specified scheduler:  <b><u>PBS scheduler (-l option)</u></b> <code>\${pbs.mem}</code> : A value for the mem directive <code>\${pbs.vmem}</code> : A value for the vmem directive  <b>Example:</b> <code>nodes=1:ppn=1,mem=\${pbs.mem},vmem=\${pbs.vmem}</code>  <b><u>LSF scheduler (-R option)</u></b> <code>\${lsf.select.mem}</code> : A value for mem in the select <code>\${lsf.select.swp}</code> : A value for swp in the select <code>\${lsf.rusage.mem}</code> : A value for mem in the rusage <code>\${lsf.rusage.swp}</code> : A value for swp in the rusage  <b>Example:</b> <code>select[mem&gt;\${lsf.select.mem} &amp;&amp; swp&gt;\${lsf.select.swp}]</code> <code>rusage[mem=\${lsf.rusage.mem}:swp=\${lsf.rusage.swp}]</code>  <b>Note:</b> This parameter is passed directly to your scheduler's job submission command. Syntax or other errors in this parameter will cause the pipeline to fail. The requirements for this parameter are determined by your scheduler's configuration. See your scheduler's admin for guidelines in setting this parameter.  Formerly: <code>SCHEDULER_RESOURCE_REQUIREMENTS</code>	
<b>Config File Key:</b>	<code>queue.sys.options</code>
<b>Command-Line Switch</b>	<code>--queue-sys-options, -a</code>
Additional command line options to pass to the scheduling command.	

<b>Example:</b> <code>-n 1</code>	
<b>Note:</b> This parameter is passed directly to your scheduler's job submission command. Syntax or other errors in this parameter will cause the pipeline to fail. See your scheduler's admin for guidelines in setting this parameter.	
Formerly: <code>ADDITIONAL_SCHEDULER_OPTIONS</code>	
<b>Config File Key:</b>	<code>queue.sys.nodes.tmp.dir</code>
<b>Command-Line Switch</b>	<code>--queue-sys-nodes-tmp-dir, -J</code>
Path to a temporary folder local to each compute node. Formerly: <code>FOLDER_FOR_TEMPORARY_FILES_ON_COMPUTE_NODES</code>	
<b>Config File Key:</b>	<code>wt.max.memory.per.job</code>
<b>Command-Line Switch</b>	<code>--max-memory-per-job, -j</code>
Specifies the maximum bytes of RAM that any one job will request; some will request less. We recommend that you choose as high a value as possible, not to exceed 8 <sup>9</sup> or the RAM available on each compute node. We recommend 6.5GB for Human Genome Build 18. Formerly: <code>MAX_MEMORY_PER_JOB_IN_BYTES</code>	
<b>Config File Key:</b>	<code>wt.memory.requirement.adjustment.factor</code>
<b>Command-Line Switch</b>	<code>--memory-requirement-adjustment-factor, -n</code>
The memory requirement for jobs handled by the scheduler are calculated dynamically. This option adjusts the memory requirement by a constant factor. Formerly: <code>MEMORY_REQUIREMENT_ADJUSTMENT_FACTOR</code>	
<b>Config File Key:</b>	<code>wt.file.reference</code>
<b>Command-Line Switch</b>	<code>--file-reference, -f</code>
Path to the multi fasta file containing the reference nucleotide sequences (must be accessible on compute nodes). Formerly: <code>FILE_REFERENCE_FASTA</code>	
<b>Config File Key:</b>	<code>wt.output.dir</code>
<b>Command-Line Switch</b>	<code>--output-dir, -o</code>
Path to the folder where all the files generated by this tool will be found. Formerly: <code>FOLDER_FOR_OUTPUT_FILES</code>	

## 4 *Split Read-Mapper Program*

**Program Name:** `split_read_mapper.sh`

**Program Version:** 1.0

**Development Languages:** Java

**PBS Is Required:** No – LSF can be used as an alternative scheduler.

**Supports AB kit or protocol or sample prep method:** N/A

Date: June 15, 2009

## 4.1 Overview

`split_read_mapper.sh` is used to align reads generated from a transcriptome experiment to a genome. The pipeline creates files containing alignments of reads that can be further processed by downstream tools.

A .csfasta file from a transcriptome experiment is the primary input. The pipeline first removes reads that match a user-defined set of sequences, such as ribosomal RNA. The remaining reads are aligned against a user-provided reference genome. The pipeline can also make use of user-provided gene annotations to detect reads that map across known and putative splice junctions.

The pipeline interfaces with PBS, LSF or SGE schedulers, optimizing the memory usage for the RAM available on the nodes in the queue.

The mapping pipeline can be summarized as 3 principal steps:

1. **Filtering:** Reads that match a user-provided file are removed. This step can be used to filter ribosomal RNA, contaminant sequences and other uninteresting sources of RNA.
2. **Alignment:** Reads are aligned against a reference genome. Jobs are distributed across the cluster in an efficient manner and the reference genome is split into chunks sized for the amount of memory available on each node.
3. **Merging & Sorting:** Results generated on different nodes are combined. Results are ordered by chromosome and position on chromosome to aid processing by downstream modules.

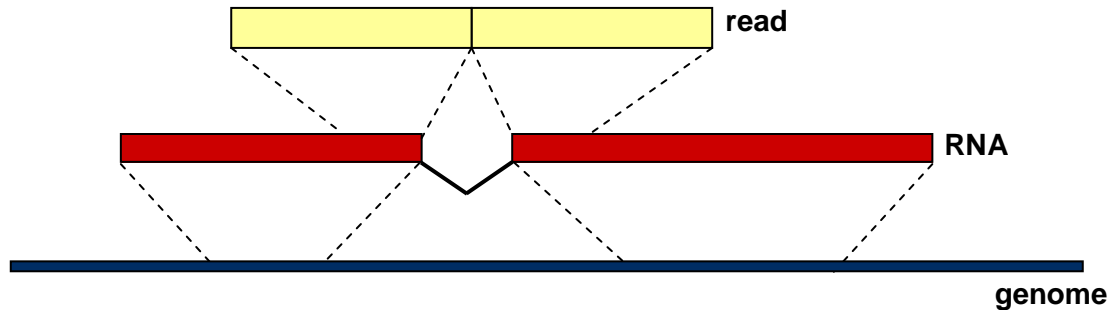
### 4.1.1 Filtering

Many reads from a transcriptome experiment are from ribosomal RNA and other uninteresting sources of RNA. To keep these reads out of the results files, `split_read_mapper.sh` will identify reads that map to a 'filter reference'. These reads are not reported in the final mapping files. The filter reference is intended to contain uninteresting sequences such as vector and adapter sequences, ribosomal RNAs, tRNAs and single base repeats.

We provide a human filter database with the WTAP test dataset available on <http://www.solidsoftwaretools.com>. Filtering is applied by mapping the 'anchor' regions of reads (see below for definition of anchor region) to the filter reference using `mapreads`. A user setting (`wt_mapper.filter.mode`) determines the effect of one or more mappings to the filter reference. Reads may be 'filtered' if one or two of the anchors maps to the filter reference. Generally, we recommend filtering a read if one or more anchor maps to the filter reference.

### 4.1.2 Alignment

Mapping transcriptome reads to a genome introduces complexities not found in traditional SOLiD™ genomic resequencing applications. Traditional read mapping for genomic resequencing applications identifies global, ungapped alignments between a read and reference having a maximum number of mismatches. Transcriptome reads present the additional possibility that a read may cross an exon-intron boundary.

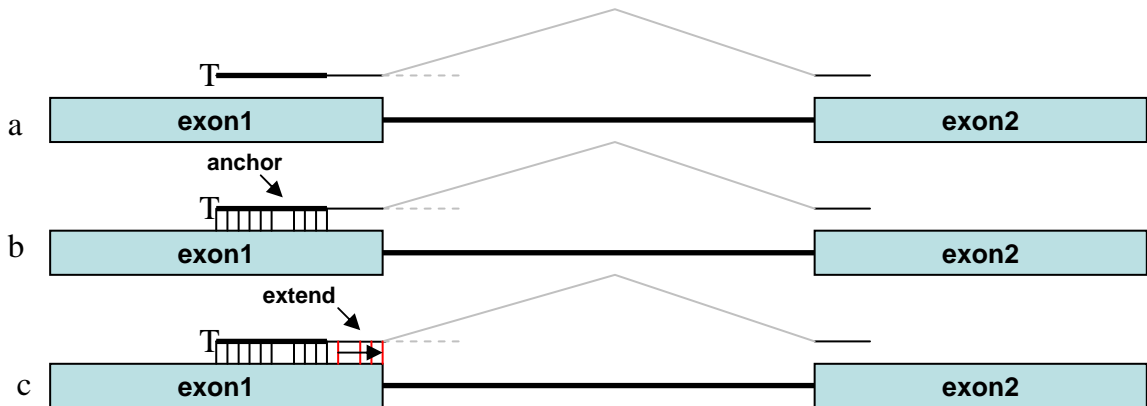


**Figure 2. Transcriptome reads that cross splice junctions do not map contiguously to the genome**

Alignments that extend into intronic sequence will have many mismatches and are therefore unlikely to be detected using the global mapping method applied in genomic resequencing applications (Fig 3a). `split_read_mapper.sh` applies two methods to map reads originating in genomic sequence near exon boundaries.

The first is the ‘anchor-extend’ method. In ‘anchor-extend’ 1 or 2 subsequences of the read are selected as ‘anchors’. Typically, one chooses anchors near the ends of the read. These ‘anchors’ are then mapped to the reference; finding global alignments with a maximum number of mismatches. (Fig 3b). The **mapreads** program is used to perform this step efficiently.

The next step of the ‘anchor-extend’ method is to ‘extend’ the global alignment of the anchor as a local alignment of the entire read with the reference. A simple ungapped alignment algorithm is applied to find the shortest alignment with the maximum score. (Fig 3c.) The alignment score is the sum of scores at each position in the alignment. The score at each position is +1 for a match and -1 for a mismatch. With the default settings, valid-Adjacent mismatches have the same effect on score as isolated mismatches. This score is recorded for use later in the Merging and Sorting step.

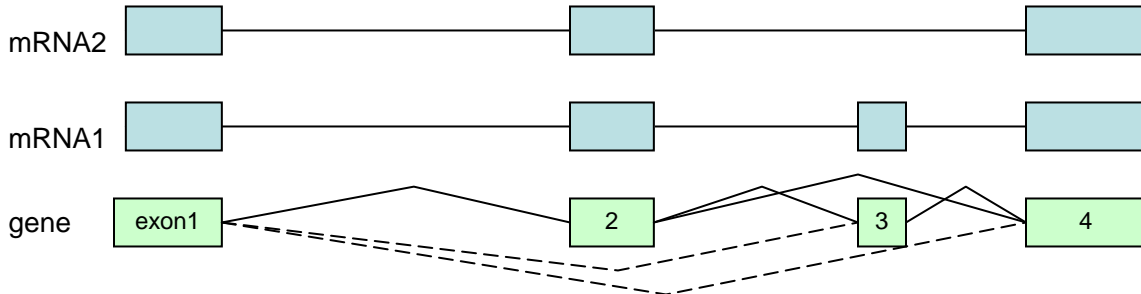


**Figure 3. Anchor Extend method. 1) A read that spans a splice junction will have colors from non-continuous regions in the reference genome. 2) An anchor region of the read is used for efficient mapping to the reference. 3) The alignment is extended as far as possible.**

The second method to map reads near exon boundaries is to use known transcript annotations to map reads to the sequences flanking known and putative splice junctions.

For the purpose of this document, known and putative junctions are defined as follows: A **known junction** is a junction that is observed in a known transcript.

A **putative junction** is a junction that could potentially occur as the result of alternative splicing of exons in a single gene (Fig 3.)



**Fig. 3. Definition of known and putative junctions. A hypothetical gene model is shown with 2 mRNA transcripts. Angled lines represent splice junctions between exons. Solid angled lines represent known junctions. Dashed angled lines represent putative junctions.**

To perform the junction mapping, a database of junctions is prepared using the reference genome sequence and the gene model. Mapping to the junction reference is performed in parallel with the mapping to the genomic reference. In this step, 'anchor-extend' is applied using only the left anchor of the read. Otherwise, mapping parameters are identical to those used in the genomic 'anchor-extend' method.

In the `split_read_mapper.sh` implementation, extension is performed after all reads have been anchored. Mapping and extension jobs are performed in parallel for the different anchors and for different regions of the reference genome, hence the mapping results of a single read will be distributed among several sub-results.

#### 4.1.3 Merging and Sorting

With parallel processing of left and right anchors on different genomic locations and splice junctions, the mapping step produces multiple result sets. The Merging and Sorting step consolidates these results into a single result, identifying uniquely mapped reads and then sorting the uniquely mapped reads by genomic location.

A read may have duplicated results due to anchoring. For instance, if the read is entirely within an exon, the alignment anchored on the right will be identical to the alignment on the left. In the merge step, these alignments are merged as a single alignment. Reads mapping to splice junctions are likely to have shorter genomic mappings in the same location. In this case the junction mapping replaces the partial genomic mapping.

Once the technical task of merging is complete, each read has a non-overlapping set of genomic mappings. The alignment score calculated during the extension step is used to determine if a mapping is unique. A score penalty is applied to splice junction mappings. Typically, no penalty is applied for mappings to known junctions (`wt.mapping.penalty.known.junction`) while a small penalty is applied to mappings to putative junctions (`wt.mapping.penalty.putative.junction`). Alignments that do not exceed a

minimum score (`wt.mapping.score.min`) are discarded. A user definable parameter (`wt.mapping.score.uniqueness.gap`) is used to determine if the read is uniquely mapped. If the highest scoring mapping exceeds all other mappings by the value of this parameter, it is called 'unique'.

Finally, the unique reads are sorted by genomic location and results are written to files. Note that the MAX format files written do not contain mappings to splice junctions. The GFF format file written contains all the uniquely mapped reads, including those mapped to junctions.

## 4.2 Usage Examples

```
split_read_mapper.sh -c myconfig.ini
```

## 4.3 Parameters

See Section 3.1 for parameters shared by Split Read Mapper and NTR Finder.

<b>Config File Key:</b>	<code>wt.delete.intermediate.files</code>
<b>Command-Line Switch</b>	<code>--delete-intermediate-files, -d</code>
Turn this option on to save more space without incurring the cost of time spent compressing files. Turning this option on means that you won't be able to restart the pipeline from an arbitrary point, but only from the point after the last successfully completed task. Values are 'true' or 'false'. Formerly: <code>DELETE_INTERMEDIATE_FILES</code>	
<b>Config File Key:</b>	<code>wt.exon.reference</code>
<b>Command-Line Switch</b>	<code>--exon-reference, -g</code>
Exon/Gene reference in GTF format.	
<b>Config File Key:</b>	<code>wt.mapper.task.skip.to</code>
<b>Command-Line Switch</b>	<code>--task-skip-to, -t</code>
Bypass certain pipeline tasks by skipping directly to a task. This is helpful when a run fails and you want to resume without repeating successfully completed steps. Value: <code>splitting, filtering, ref_partitioning, mapping, extension, or merge</code> . Formerly: <code>RUN_READ_SPLITTING, RUN_READ_FILTERING, RUN_REFERENCE_PARTITIONING, RUN_MAPPING, RUN_EXTENSION, RUN_MERGE</code>	
<b>Config File Key:</b>	<code>wt.mapper.filter.mode</code>
<b>Command-Line Switch</b>	<code>--filter-mode, -i</code>
Controls the way filtering functions: <ul style="list-style-type: none"> <li>• <code>OFF</code>: Filtering is off.</li> <li>• <code>ONE_OR_MORE</code>: A read is filtered if one or more splits maps to the filter reference.</li> <li>• <code>BOTH</code>: A read is filtered if both splits map to the filter reference.</li> </ul> Formerly: <code>FILTERING_MODE</code>	



<b>Command-Line Switch</b>	<code>--valid-adjacent-mismatches-count-as-one, -v</code>
<p>Specifies whether or not to count valid adjacent mismatches as one mismatch, instead of two.            Values: true or false            Formerly: <code>VALID_ADJACENT_MISMATCHES_COUNT_AS_ONE_FOR_MAPPING_AND_EXTENSION</code></p>	
<b>Config File Key:</b>	<code>wt.mapper.match.iub</code>
<b>Command-Line Switch</b>	<code>--match-iub, -H</code>
<p>Specifies whether or not to count alignments to consistent IUB degenerate symbols as a match, instead of as a mismatch.            Values: true or false            Formerly: <code>MATCHES_TO_IUPACS_VALID_FOR_MAPPING_AND_EXTENSION</code></p>	
<b>Config File Key:</b>	<code>wt.mapper.split.left.length</code>
<b>Command-Line Switch</b>	<code>--split-left-length, -L</code>
<p>Specifies the length of the first part of the read sequence split. Be aware that performance issues can arise if the read split is not long enough to result in a small number of hits.            Formerly: <code>LENGTH_OF_FIRST_PART_OF_READ</code></p>	
<b>Config File Key:</b>	<code>wt.mapper.split.left.mismatches</code>
<b>Command-Line Switch</b>	<code>--split-left-mismatches, -E</code>
<p>Specifies the number of mismatches tolerated in the first part of the read sequence when mapping to the reference. Increasing this number may significantly decrease performance.            Formerly: <code>MAX_MISMATCHES_IN_FIRST_PART_OF_READ_FOR_MAPPING</code></p>	
<b>Config File Key:</b>	<code>wt.mapper.split.right.length</code>
<b>Command-Line Switch</b>	<code>--split-right-length, -R</code>
<p>Specifies the length of the last part of the read sequence split. (We have not yet tested values other than 25). Performance issues can arise if the read split is not long enough to result in a small number of hits. Formerly: <code>LENGTH_OF_LAST_PART_OF_READ</code></p>	
<b>Config File Key:</b>	<code>wt.mapper.split.right.mismatches</code>
<b>Command-Line Switch</b>	<code>--split-right-mismatches, -I</code>
<p>Specifies the number of mismatches tolerated in the last part of the read sequence when mapping to the reference. Increasing this number may significantly decrease performance.            Formerly: <code>MAX_MISMATCHES_IN_LAST_PART_OF_READ_FOR_MAPPING</code></p>	
<b>Config File Key:</b>	<code>wt.mapper.score.min</code>
<b>Command-Line Switch</b>	<code>--score-min, -s</code>
<p>Minimum score required for an alignment to be reported in the final output file.            Formerly: <code>MIN_ALIGNMENT_SCORE_FOR_REPORTING_ALIGNMENT</code></p>	

<b>Config File Key:</b>	<code>wt.mapper.score.uniqueness.gap</code>
<b>Command-Line Switch</b>	<code>--score-uniqueness-gap, -G</code>
Identifies unique mapping reads. When a read has multiple mappings, it is unique if it has a mapping with a score that exceeds the next highest score by an amount greater than or equal to this value. Formerly: <code>MIN_GAP_IN_ALIGNMENT_SCORE_TO_SECOND_BEST_ALIGNMENT_FOR_UNIQUENESS</code>	
<b>Config File Key:</b>	<code>wt.mapper.file.reads</code>
<b>Command-Line Switch</b>	<code>--file-reads, -A</code>
Path to the multi-fasta file of reads in color space (The file must be accessible on compute nodes.) Formerly: <code>FILE_FULL_LENGTH_READS_CSFASTA</code>	
<b>Config File Key:</b>	<code>wt.mapper.penalty.known.junction</code>
<b>Command-Line Switch</b>	<code>--penalty-known-junction, -B</code>
Score penalty for mapping to an known splice junction.	
<b>Config File Key:</b>	<code>wt.mapper.penalty.putative.junction</code>
<b>Command-Line Switch</b>	<code>--penalty-putative-junction, -C</code>
Score penalty for mapping to a putative splice junction.	

## 4.4 System Input files

### 4.4.1 Reads file

A csfasta file containing the reads to be mapped

### 4.4.2 Reference sequence file

A multi-fasta file containing the reference sequences to which the reads are mapped.

### 4.4.3 Filter reference sequence file

A multi-fasta file containing the filter reference. These sequences are tRNA, mitochondrial, and other transcripts that are not of interest in typical experiments. A human filter reference file is available for download.

### 4.4.4 GTF genes file

An optional file in GTF format describing genes in the reference. See Appendix 1 GTF file for defining gene annotations for details of this file format.

## 4.5 System Output Files

A required parameter for `split_read_mapper.sh` is the location of the output directory (`wt.mapper.output`). This directory contains the following:

#### 4.5.1 Mapper.log

A file useful for troubleshooting; it should be included in any support inquiries. Note that log files are never overwritten. Should you re-run the pipeline using the same output directory, the old log file will be renamed with a serially assigned number.

#### 4.5.2 tmp/

The local working directory for `split_read_mapper.sh`. This is different than the file specified by the `queue.sys.nodes.tmp.dir`, which specifies a local working directory on the compute nodes. `tmp/` may be deleted after successful execution of the pipeline.

#### 4.5.3 scripts/

This directory contains some shell scripts for batch submission and batch termination of scheduled jobs. These are useful if you jobs need to be resubmitted or if you want to cancel several submitted jobs.

#### 4.5.4 Alignment Report

A summary of the mapping results, including score distributions.

File name pattern: `output/alignmentReport.txt`

#### 4.5.5 Score distribution plots

Plots of the overall score distributions.

File name pattern:

`output/alignmentsByScore.histograms.cumulative.pdf`,  
`output/alignmentsByScore.histograms.pdf`

#### 4.5.6 Filter report

A summary of the filtering process.

File name pattern: `output/filterReport.txt`

#### 4.5.7 MAX file

A MAX file containing all reads that map to genomic reference. Does not include mappings to splice junctions. File sort order is determined by the sort order of the input csfasta file. See Appendix 2 for details on the Max File format.

File name pattern: `output/*.max.merged.filtered.csfasta`

#### 4.5.8 Reads without minimum alignment score

A csfasta file containing the reads that did not map to the genomic reference with a sufficiently high score.

File name pattern: `output/*.readsWithoutMinScoringAlignment.csfasta`

#### 4.5.9 Sorted MAX file

A MAX file containing all the reads that were uniquely mapped to the reference. Does not include mappings to splice junctions. File is sorted by the genomic location of the mapping. See Appendix 2 for details on the Max File Format.

File name pattern: `output/*.sorted.max.merged.filtered.csfasta`

#### 4.5.10 GFF file

A GFF version 2 file containing all the reads that were mapped uniquely, including those mapped to splice junctions. See Appendix 1 for details on this file format.

File name pattern: `output/*.sorted.max.merged.filtered.gff`

## 5 Count Tags Program

**Program Name:** `count_tags.pl`

**Program Version:** 1.0

**Development Languages:** Perl

**Supports AB kit or protocol or sample prep method:** N/A

**Date:** June 15, 2009

### 5.1 Overview

`count_tags.pl` starts with the MAX output file of the pipeline and generates tag counts and coverage files for annotated regions, or for the entire reference genome.

There are two type of formats used for output files: gff (used for counts files) and wiggle (used for coverage files) that can be visualized in the UCSC Genome Browser.

The tag information contained in MAX output file produced by the Whole Transcriptome pipeline is used to generate different types of counts. We generate either count of reads matching predefined regions (such as exons) provided by the user in a gff format file, or base coverage counts (in a wiggle format) of either the same regions of interest or for the entire reference genome. Optionally, this program can reformat the MAX file in a gff format.

There are two main steps:

1. Screening the MAX file to gather the required information. For some options this information is stored in temporary files on the hard drive.
2. Printing the output files. Depending on the parameters used the printing is done simultaneously with MAX file reading.

### 5.2 Usage

```
count_tags.pl -max /path/sample_F3.max.merged.filtered.csfasta
-r /share/reference/human.fa
-s no
-a /share/reference/exonReference.gtf
-t 50
-read_alignment_type top_score
-alignment_cut 23
-counts /path/73_20080930_1_sample_F3.counts
-wig /path/wigdir
-produce_count yes
-produce_wig yes
```

-produce\_wig\_genome yes  
-produce\_gff\_reads no

### 5.3 Parameters - Required

Parameter	Description	Type/Range/Example
-max	MAX file output of the WT pipeline. It can be unsorted (extension <code>.max.merged.filtered.csfasta</code> ) or sorted (extension <code>sorted.max.merged.filtered.csfasta</code> ) based on reads matching locations.	73_20080930_1_sample_F3.max.merged.filtered.csfasta, 73_20080930_1_sample_F3.sorted.max.merged.filtered.csfasta
-s	Whether or not the max file is sorted: Default is no.	yes, no
-r	The reference used in the WT pipeline (under the <code>wt.file.reference</code> field of the configuration file).	/share/reference/genomes/chromFa/human.fa
-a	Annotation file in gtf format.	For human samples, use the <code>refGene.gtf</code> file provided on <a href="http://solidsoftwaretools.com">solidsoftwaretools.com</a> .
-t	Length of the reads.	35
-read_alignment_type	The type of read alignment that will be considered for both counts and coverage. The choices are: <ul style="list-style-type: none"> <li><code>top_score</code> - Only the location(s) giving the highest alignment score is considered.</li> <li><code>unique</code> - Only reads matching in a unique location are considered.</li> <li><code>all</code> - All matching locations are used.</li> <li><code>random</code> - From each read, the script chooses randomly a matching location used for counts and coverage.</li> </ul>	<code>top_score</code> , <code>unique</code> , <code>all</code> , <code>random</code>
-min_score	Threshold for filtering matching locations. Only locations with an alignment score at or above this threshold are considered.	24
-counts	Full path to the output counts file that will be generated.	/path/73_20080930_1_sample_F3.counts
-wig	Full path to the directory where the output coverage file(s) will be generated.	/path/wigdir

Parameter	Description	Type/Range/Example
-gff_reads	(Optional) Full path to the output file that converts MAX into gff format.	/path/73_20080930_1_sample_F3.gff
-produce_count	Whether or not a counts file needs to be generated. Default: yes.	yes, no
-wig	Whether or not a coverage file of annotated region needs to be generated. Default: yes.	Yes, no
-produce_wig_genome	Whether or not a genome coverage files need to be generated. Default: yes	Yes, no
-produce_gff_reads	Whether or not to convert the MAX file to gff format. Default: no	Yes, no
-coverage_filter_threshold	Specifies the minimum amount of coverage required for a position to be reported in the wig file.	10
-split_coverage_by_chromosome	Specifies whether wig output should be written to a separate file for each chromosome.	Yes, no

## 5.4 System Input Files

### 5.4.1 MAX file

MAX file output of the WT pipeline. It can be unsorted (extension `.max.merged.filtered.csfasta`) or sorted (extension `sorted.max.merged.filtered.csfasta`) based on reads matching locations.

### 5.4.2 Reference fasta

The reference fasta file to which the reads in 'MAX file' have been mapped.

### 5.4.3 GTF Annotation file

A GTF-formatted exon reference. This file must define features of the sequences in 'Reference fasta'. This file can contain any feature type but only features of type 'exon' are used.

## 5.5 System Output Files

### 5.5.1 Counts file (\*.counts)

A file in gff format, identical to the gff annotation file (parameters gff). For each region contained in the gff annotation file, the number or reads hitting the corresponding region are reported in the "quality" column (8).

### 5.5.2 Annotation wig file (chr?.annotation.wig)

A coverage file in wiggle format for regions contained in the annotation file, one for each genomic region (chromosome). The coverage file can be loaded in the UCSC Genome Browser.

### 5.5.3 Wig file (chr?.wig)

A coverage file in wiggle format for regions contained in the annotation file, one for each genomic region (chromosome). The coverage file can be loaded in the UCSC Genome Browser.

## 6 NTR Finder Program

**Program Name:** ntr\_finder.sh

**Program Version:** 1.0

**Development Languages:** Java

**Supports AB kit or protocol or sample prep method:** N/A

**Date:** June 15, 2009

### 6.1 Overview

NTR Finder is one of three programs of the Whole Transcriptome Analysis Pipeline. It works with the other two programs, `split_read_mapper.sh` and `count_tags.pl`. NTR Finder reads `.max` files as input and produces `.gff` files as output.

A **Novel Transcribed Region (NTR)** is a segment of genomic sequence that is transcribed but is not currently annotated as an exon in a database. The Whole Transcriptome Pipeline features a tool for identifying such regions. You can use these regions to supplement the exon reference that is used in the counting script:

`count_tags.pl`.

You perform NTR finding by using the `ntr_finder.sh` program. NTR finding requires one or more sorted `.max` files from the program `split_read_mapper.sh`.

NTRs are identified by the following method: At each position in the reference, the **coverage** is the number of reads with alignments that span the position. Only those reads that are mapped with a score above a threshold score (`min-alignment-score`) are considered in calculating the coverage. A window of length `window-size` is scanned through the reference sequence. The `window-coverage` is the average coverage across the positions in the window. When the `window-coverage` within a window exceeds a threshold value called `min-window-coverage`, the window is considered to be part of a **predicted transcribed region, PTR**. Successive window positions with `window-coverage` exceeding the minimum base count (`min-base-count`) constitute a PTR.

When the end of PTR is identified, the boundaries of the PTR are trimmed. A minimum coverage cutoff is calculated as the average coverage within the PTR multiplied by the **trimming fraction**. Trimming is performed by removing positions from the ends of the PTR until the coverage at the ends equals or exceeds the minimum coverage cutoff. Finally, the **average coverage** of the PTR is calculated by summing the coverage at every position in PTR and dividing by the length of the PTR.

After a PTR has been identified, a set of **annotated TRs (ATR;** for example, annotated RefSeq exons) is searched for a match. The **overlap** between a PTR and an ATR is the number of shared positions divided by the length of the PTR or the ATR, whichever fraction is greater. A PTR matches an ATR when the overlap is greater than or equal to a minimum fraction (termed `min-overlap`).

The results of NTR finding are reported in GFF format. Each line specifies the boundaries and strand of a transcribed region. The attributes field reports the average coverage across the PTR and overlaps between the PTR and all overlapping ATRs.

Note that **all** PTRs are reported, even if they overlap with an ATR. PTRs that do not overlap an ATR are likely to represent **novel transcribed regions (NTRs)**.

Overlap is reported as two numbers: the PTR overlapping ATR, and the ATR overlapping PTR. See Figure 4 - the length of bars represents sequence length. PTR overlapping ATR is  $b/(b + c)$ . ATR overlapping PTR is  $b/(a + b)$ .



**Figure 4. Calculation of overlap.**

The parameters `window-size`, `min-score`, `min-window-coverage`, `trimming-fraction`, and `overlap` are configurable. The optimal values for `window-size` and `min-window-coverage` are highly dependent on the samples and application. For this reason, Applied Biosystems recommends optimizing the parameters by selecting a range of values for `window-size` and `min-window-coverage`. To reduce compute time, you can do this optimization on reads that map to a subset of the reference genome. When you choose parameters, consider the number of TRs reported, the fraction of annotated exons found, and the overlap between annotated exons and TRs. Three plots are produced to help you select values for `window-size` and `min-window-coverage`. The exact choice of parameters depends on your goals.

For example, if you want to be conservative and predict few false positive NTRs, choose larger values for `window-size` and `min-window-coverage`. When you choose the `window-size`, note that there is a trade-off in overlap between PTRs and ATRs. Larger `window-size` values tend to lengthen PTRs and thus increase overlap when you consider the ATR; however, this tends to reduce overlap when you consider the PTR. After selecting optimal parameters, apply the selected parameters to the entire reference genome.

### 6.1.1 Optimizing NTR Finding Parameters

Choosing values for `window-size` and `min-window-coverage` is sample- and application-specific. The relationship between coverage and NTR finding sensitivity and specificity is not well understood. It is also likely that optimal parameters depend on the size of the TRs of interest. For example, the optimal parameters for finding very small exons are different than those for long non-coding RNAs. While these issues are being worked out, apply an optimization process to select parameters.

`ntr_finder.sh` enables the user to find TRs with multiple values for `window-size` and `min-window-coverage`. The resulting `NTR_report.txt` file and plots help you to specify parameter values. Refer to Section 6.5.3 and the discussion there.

## 6.2 Usage

```
% ntr_finder.sh -c myconfig.ini
```

### 6.3 Parameters

<b>Config File Key:</b>	<code>wt.ntr.finder.max.file</code>
<b>Command-Line Switch</b>	<code>--max-file, -m</code>
<p>Paths to sorted max files to use in NTR Finding. These must be sorted max files. Each of these files must be mapped to the same reference file. Multiple values may be specified.</p> <p>Formerly: <code>NTR_MAX_FILE</code> (<code>-m, --sorted-max-file</code>)</p>	
<b>Config File Key:</b>	<code>wt.ntr.finder.genomic.region</code>
<b>Command-Line Switch</b>	<code>--genomic-region, -r</code>
<p>Region of genome to consider for NTR finding. Specify regions in the following format <code>seq_id[start-end][, seq_id[start-end]...]</code></p> <p>Examples:  <code>chr2</code> - Chromosome 2  <code>chr2 0-1000000</code> - First 1000000 base of Chromosome 2  <code>chr2:0-1000000,chr5:0-1000000</code> - First 1000000 bases of Chromosomes 2 and 5</p> <p><code>seq_id</code> must match the entries in <code>FILE_REFERENCE_FASTA</code>. If the description line of the fasta file contains whitespace, the <code>seq_id</code> is first string of non-whitespace characters.</p> <p>Formerly: <code>NTR_GENOMIC_REGION</code></p>	
<b>Config File Key:</b>	<code>wt.ntr.finder.trimming.fraction</code>
<b>Command-Line Switch</b>	<code>--trimming-fraction, -t</code>
<p>Used to trim the ends of PTRs. See algorithm description. Formerly: <code>NTR_TRIMMING_FRACTION</code></p>	
<b>Config File Key:</b>	<code>wt.ntr.finder.min.alignment.score</code>
<b>Command-Line Switch</b>	<code>--min-alignment-score, -s</code>
<p>Specifies the minimum alignment score of mapped reads to be included in NTR finding.</p> <p>Formerly: <code>NTR_MIN_ALIGNMENT_SCORE</code></p>	
<b>Config File Key:</b>	<code>wt.ntr.finder.min.overlap</code>
<b>Command-Line Switch</b>	<code>--min-overlap, -v</code>
<p>Specifies the minimum amount of overlap between PTR and ATR to identify a match.</p> <p>Formerly: <code>NTR_MIN_OVERLAP</code></p>	
<b>Config File Key:</b>	<code>wt.ntr.finder.min.window.coverage</code>
<b>Command-Line Switch</b>	<code>--min-window-coverage, -b</code>
<p>Specifies the minimum window coverage to use in the NTR finding process. Multiple values may be specified. When multiple values for <code>NTR_MIN_WINDOW_COVERAGE</code> are specified, a run is performed for every pair of values. Formerly: <code>NTR_MIN_WINDOW_COVERAGE</code></p>	
<b>Config File Key:</b>	<code>wt.ntr.finder.min.window.size</code>

<b>Command-Line Switch</b>	<code>--min-window-size, -w</code>
Specifies the length of the window (in bases) to use in the NTR finding process. Multiple values may be specified. When multiple values for <code>NTR_WINDOW_SIZE</code> are specified, a run is performed for every pair of values. Formerly: <code>NTR_WINDOW_SIZE</code>	
<b>Config File Key:</b>	<code>wt.ntr.finder.max.ptrs.per.megabase</code>
<b>Command-Line Switch</b>	<code>--max-ptrs-per-megabase, -x</code>
Specifies the maximum number of PTRs to report. If a result contains more than this number of PTRs per megabase of reference sequence, a GFF file will not be returned. The X axis of the optimization plots is cut-off at this value. Formerly: <code>MAX_PTRS_PER_MEGABASE</code>	
<b>Config File Key:</b>	<code>wt.ntr.finder.file.atr.reference</code>
<b>Command-Line Switch</b>	<code>--file-atr-reference, -g</code>
Path to a GFF file containing ATRs ('known' exons). These exons are used to match PTRs and calculate overlap. Formerly: <code>FILE_ATR_REFERENCE_GFF</code>	
<b>Config File Key:</b>	<code>queue.sys</code>
<b>Command-Line Switch</b>	<code>--queue-sys, -e</code>
Define the scheduling environment. The value should be <code>pbs</code> , <code>lsf</code> or <code>sge</code> . Formerly: <code>SCHEDULING_ENVIRONMENT</code>	
<b>Config File Key:</b>	<code>queue.sys.queue</code>
<b>Command-Line Switch</b>	<code>--queue-sys-queue, -q</code>
Name of the scheduler queue to use. Formerly: <code>NAME_OF_QUEUE</code>	
<b>Config File Key:</b>	<code>queue.sys.resource.string</code>
<b>Command-Line Switch</b>	<code>--queue-sys-resource-string, -u</code>
Specifies the resource requirement string for the scheduling command. This corresponds to the <code>qsub -l</code> option in PBS and the <code>bsub -R</code> option in LSF.  The memory requirement portions of this parameter can be calculated dynamically by the pipeline. The following tokens are substituted with calculated values when used with the specified scheduler:	
<b><u>PBS scheduler (-l option)</u></b>	
<code>\${pbs.mem}</code> : A value for the mem directive <code>\${pbs.vmem}</code> : A value for the vmem directive	
<b>Example:</b> <code>nodes=1:ppn=1,mem=\${pbs.mem},vmem=\${pbs.vmem}</code>	
<b><u>LSF scheduler (-R option)</u></b>	
<code>\${lsf.select.mem}</code> : A value for mem in the select <code>\${lsf.select.swp}</code> : A value for swp in the select <code>\${lsf.rusage.mem}</code> : A value for mem in the rusage <code>\${lsf.rusage.swp}</code> : A value for swp in the rusage	

<p><b>Example:</b>  <pre>select[mem&gt;\${lsf.select.mem} &amp;&amp; swp&gt;\${lsf.select.swp}] rusage[mem=\${lsf.rusage.mem}:swp=\${lsf.rusage.swp}]</pre></p> <p><b>Note:</b> This parameter is passed directly to your scheduler's job submission command. Syntax or other errors in this parameter will cause the pipeline to fail. The requirements for this parameter are determined by your scheduler's configuration. See your scheduler's administrator for guidelines in setting this parameter.</p> <p>Formerly: SCHEDULER_RESOURCE_REQUIREMENTS</p>	
<b>Config File Key:</b>	queue.sys.options
<b>Command-Line Switch</b>	--queue-sys-options, -a
<p>Additional command line options to pass to the scheduling command.</p> <p><b>Example:</b>  <pre>-n 1</pre></p> <p><b>Note:</b> This parameter is passed directly to your scheduler's job submission command. Syntax or other errors in this parameter will cause the pipeline to fail. See your scheduler's administrator for guidelines in setting this parameter.</p> <p>Formerly: ADDITIONAL_SCHEDULER_OPTIONS</p>	
<b>Config File Key:</b>	queue.sys.nodes.tmp.dir
<b>Command-Line Switch</b>	--queue-sys-nodes-tmp-dir, -J
<p>Path to a temporary folder local to each compute node.</p> <p>Formerly: FOLDER_FOR_TEMPORARY_FILES_ON_COMPUTE_NODES</p>	
<b>Config File Key:</b>	wt.max.memory.per.job
<b>Command-Line Switch</b>	--max-memory-per-job, -j
<p>Specifies the maximum bytes of RAM that any one job will request; some will request less. We recommend that you choose as high a value as possible, not to exceed 8<sup>9</sup> or the RAM available on each compute node. We recommend 6.5GB for Human Genome Build 18.</p> <p>Formerly: MAX_MEMORY_PER_JOB_IN_BYTES</p>	
<b>Config File Key:</b>	wt.memory.requirement.adjustment.factor
<b>Command-Line Switch</b>	--memory-requirement-adjustment-factor, -n
<p>The memory requirement for jobs handled by the scheduler are calculated dynamically. This option adjusts the memory requirement by a constant factor.</p> <p>Formerly: MEMORY_REQUIREMENT_ADJUSTMENT_FACTOR</p>	
<b>Config File Key:</b>	wt.file.reference
<b>Command-Line Switch</b>	--file-reference, -f
<p>Path to the multi fasta file containing the reference nucleotide sequences (must be accessible on compute nodes). Formerly: FILE_REFERENCE_FASTA</p>	
<b>Config File Key:</b>	wt.output.dir

<b>Command-Line Switch</b>	--output-dir, -o
Path to the folder where all the files generated by this tool will be found. Formerly: FOLDER_FOR_OUTPUT_FILES	

## 6.4 System Input files

### 6.4.1 MAX file

MAX file output of the WT pipeline.

## 6.5 System Output files

### 6.5.1 PTR GFF files

The `ptr_results` directory contains the positions of the PTRs. The results are organized into directories based on the `window-size` and `min-window-coverage` parameters used. The results are presented in GFF format. One GFF file is produced per strand. Each line of the GFF file represents a PTR.

Column Name	Description	Example
seqid	Reference sequence id.	chr1
source	Always <code>ntr_finder</code> .	<code>ntr_finder</code>
type	Always <code>transcript_region</code> .	<code>transcript_region</code>
start	Start position of the PTR.	1000000
end	End position of the PTR.	1000100
score	Not used.	
strand	The strand on which the PTR is detected.	+ -
frame	Not used.	
attributes	ID: A unique identifier for the PTR of the form 'PTR_#' where # is a serially assigned number.	<code>ID=PTR_14;</code>
	coverage: The average coverage in the PTR.	<code>coverage=14.1;</code>
	atr_match: A comma-separated list of atr positions from the exon reference that match this PTR.	<code>atr_match=276397-276517,276397-276547;</code>
	atr_overlap: A comma-separated list of overlap values, corresponding to the values in atr_match. These overlaps are calculated as the length of overlap divided by the length of the ATR.	<code>atr_overlap=1.00,0.99;</code>
	ptr_overlap: A comma-separated list of overlap values, corresponding to the values in atr_match. These	<code>ptr_overlap=0.54,0.66;</code>

Column Name	Description	Example
	overlaps are calculated as the length of overlap divided by the length of the PTR.	

### 6.5.2 NTR Report

NTR\_report.txt is a tab-delimited summary of the results. Each line represents a result from one set of parameters. If you specify multiple values for NTR\_WINDOW\_SIZE and NTR\_MIN\_WINDOW\_COVERAGE, you will have multiple lines in this file.

Column Name	Description	Example
window size	The window size that was used.	25
min window coverage	The min window coverage that was used.	10
min overlap	The min-overlap that was used.	0.5
min score	The min-score that was used.	24
trimming fraction	The trimming fraction that was used.	0.1
ptr-count	Total number of PTRs identified.	1591
atr-count	Total number of ATRs in the region analyzed.	143
number-of-ptrs-matching-atr	Number of TRs that overlap an annotated transcribed region (ATR).	161
number-of-atrs-matching-ptr	Number of annotated ATRs that overlap a PTR.	129
avg-frac-atr-overlapping-ptr	The average fraction of ATR overlapping PTR. Fraction of ATR overlapping PTR is calculated as the length of overlap divided by the length of the PTR.	0.8336
avg-frac-ptr-overlapping-atr	The average fraction of PTR overlapping ATR. Fraction of PTR overlapping ATR is calculated as the length of the overlap divided by the length of the ATR.	0.817

### 6.5.3 Optimization plots

Three plots, using values from the NTR Report, are produced to aid in optimizing parameters. Each point in these plots represents a single set of NTR finding parameters.

Use these plots to select the optimal parameters for NTR finding. Please note that min-window-coverage is not currently labeled in the plots. However points tend to be distributed with high min-window-coverage on the left side of the plot and low min-window-coverage on the right. We aim to improve these visualizations in a future release. You can determine the min-window-coverage of specific points by looking at NTR\_report.txt.

All three plots have Number of NTRs on the X-axis. Each plot features a curve corresponding to a window-size. These curves help the user determine the relationship between window-size, min-window-coverage and various measures of similarity to the set of exons from the ATR Reference.

The plots present similarity to the exon reference in three ways:

- The number of exons from the reference that are 'found' as PTRs.
- The average overlap as a fraction of exon size.
- The average overlap as a fraction of PTR size.

Generally, the number of PTRs identified increases as window-size and min-window-coverage are decreased. As min-window-coverage is decreased, at a certain point the number of PTRs begins to increase at a high rate and an increasing proportion of these PTRs are false positives. When you specify values for parameters, examine the similarity to the exon reference while controlling the number of PTRs reported. Refer back to Section 6.1.1, Optimizing NTR Finding Parameters.

### 6.5.3.1 ATRs Found

This plot has Fraction of ATRs found on the Y-axis. This value is the number of ATRs from the ATR reference file that match an PTR divided by the total number of ATRs in the ATR reference.

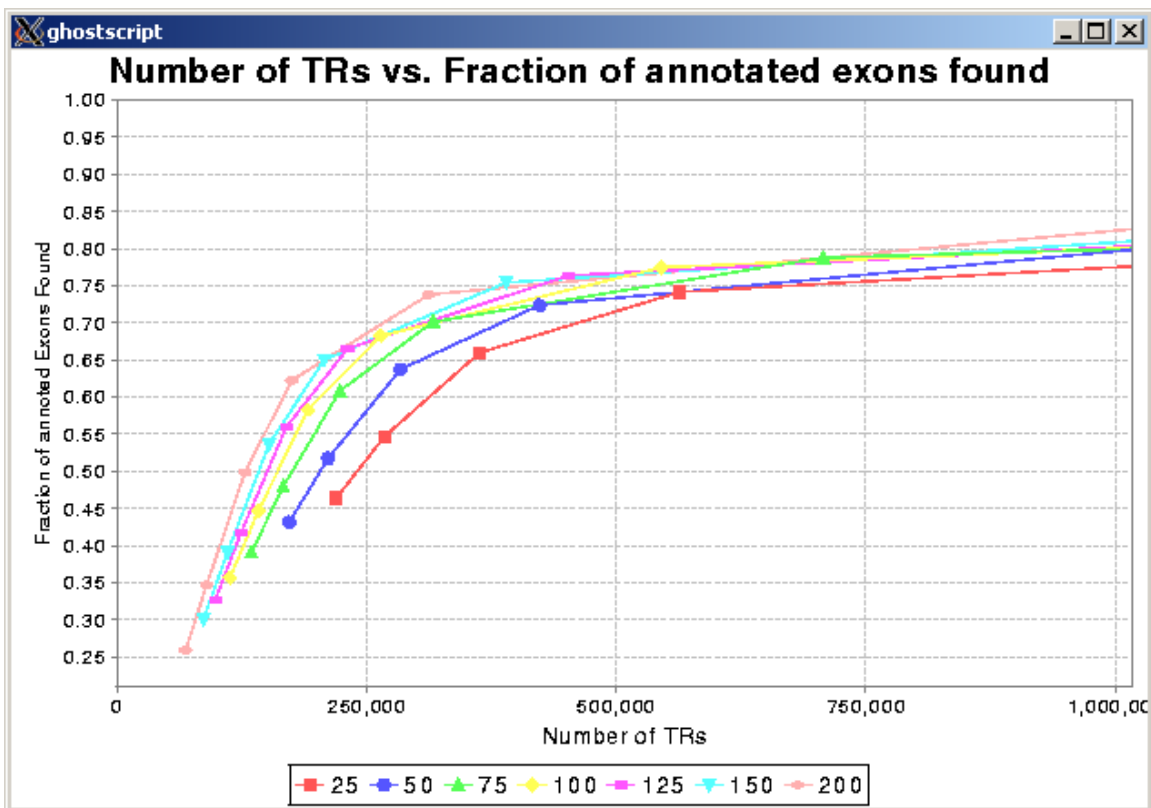


Figure 5. Plot of Number of PTRs vs. Fraction of ATRs found. min-base-coverage is decreasing from left to right.

*6.5.3.2 PTRs overlapping ATRs*

This plot has 'Average fraction of PTR overlapping ATR' on the Y axis. This value corresponds to the column `avg-frac-ptr-overlapping-atr` in `NTR_report.txt`.

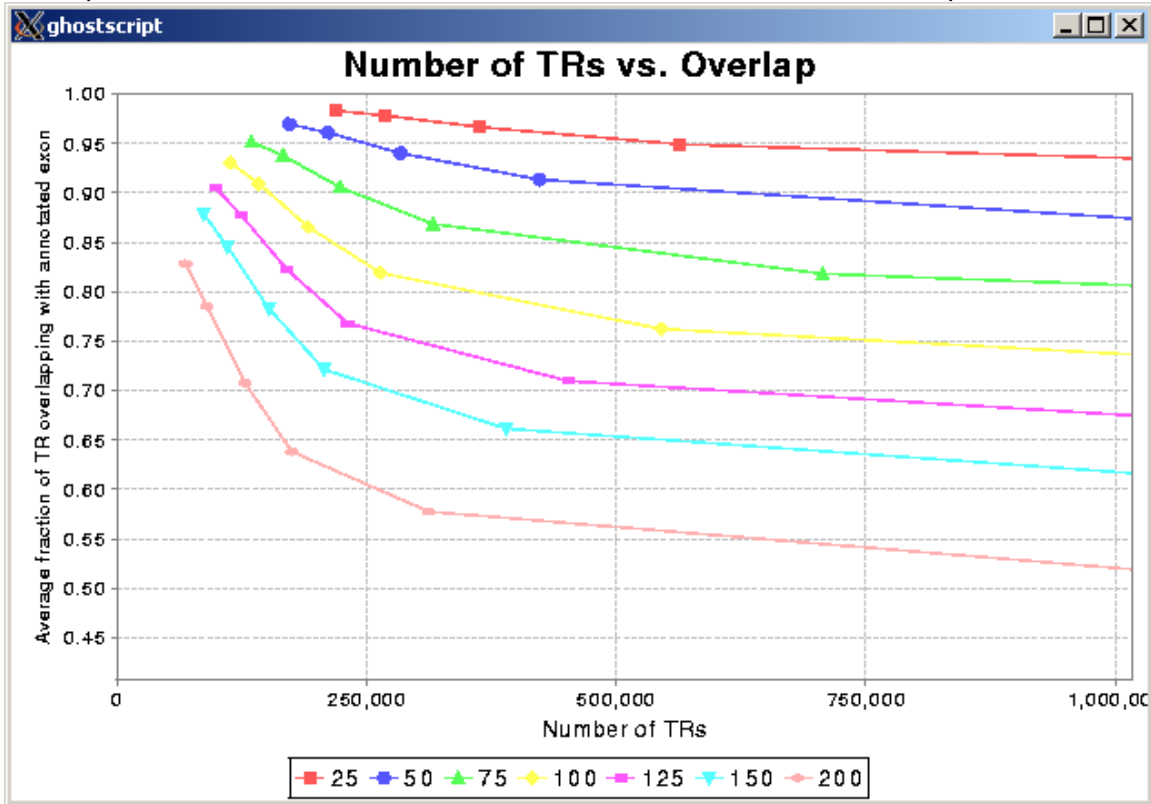


Figure 6. Plot of Number of PTRs vs. average fraction of PTR overlapping with ATR

*6.5.3.3 ATRs overlapping PTRs*

This plot has 'Average fraction of ATR overlapping PTR' on the Y axis. This value corresponds to the column `avg-frac-atr-overlapping-ptr` in `NTR_report.txt`.

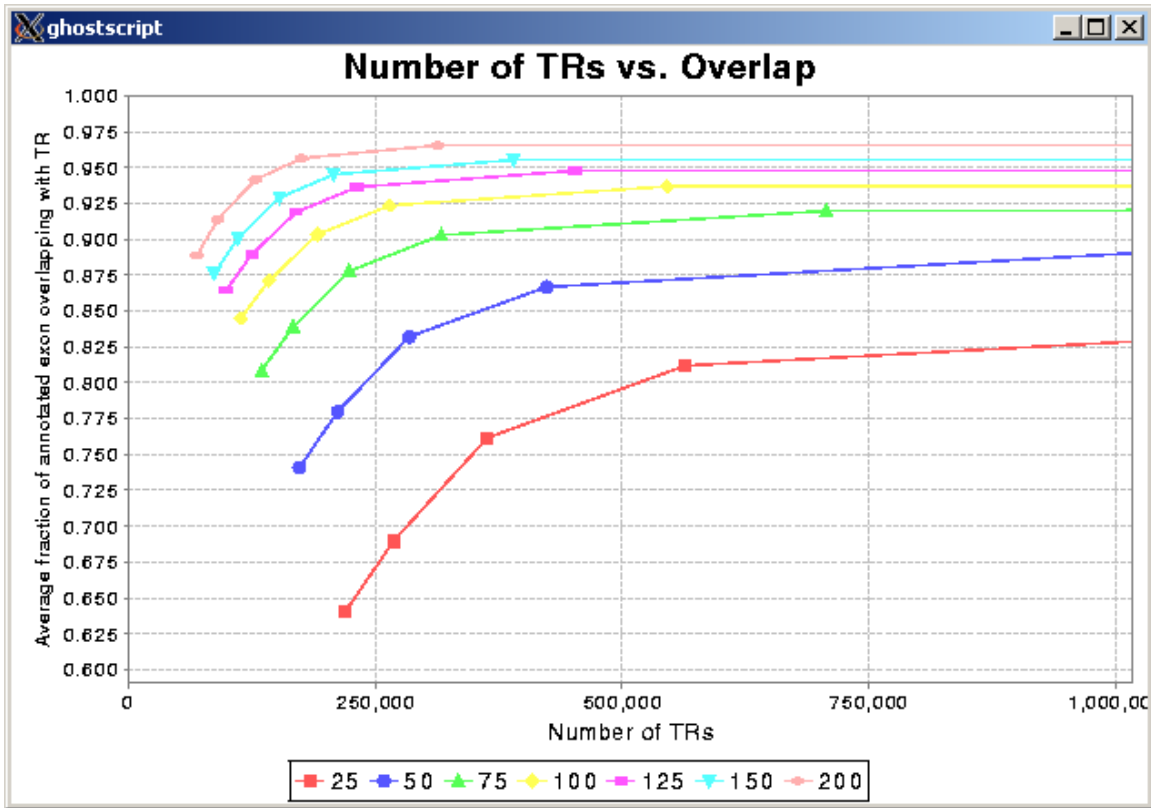


Figure 7. Plot of Number of TRs vs. average fraction of ATR overlapping with PTR

## A Appendix 1 GTF file for defining gene annotations

A GTF file provided to `split_read_mapper.sh` via the `wt.exon.reference` parameter defines the genes and transcripts in the reference genomes. Details on the GTF format can be obtained here: <http://mblab.wustl.edu/GTF2.html>. A GTF file containing the human genes in UCSC Genome Browser's RefGene table is available on <http://www.solidsoftwaretools.com>.

Please note that the GTF files available directly from the UCSC Genome Browser are **not** appropriate for this tool because they do not group features by `gene_id`. WTAP requires this file to group exons from the same gene with common values in the `gene_id` field in the attributes column. We have provided a program, `bin/refgene2gff.sh` for converting the ASCII dump of the RefGene table (`RefGene.txt`) from the UCSC Genome Browser to a GTF format appropriate for use with WTAP tools. `RefGene.txt` is available from <http://hgdownload.cse.ucsc.edu/goldenPath/hg18/database/>

To convert `RefGene.txt` to `RefGene.gtf`:

```
% refgene2gff.sh -i refGene.txt -o refGene.gtf
```

**Note:** We have only tested this program with the UCSC's human `RefGene.txt`. Some have reported that slightly different `RefGene.txt` formats are used for non-human databases at UCSC. In those cases or in cases where a `RefGene.txt` file is not available, a custom script for preparing this GTF file is required.

## **B Appendix 2 MAX file format**

The MAX file format is FASTA-compliant, and is comprised of a header line followed by the color sequence of the read.

The FASTA header line has the following format, made up of fields separated by comma (","), pound ("#"), or period (".") characters:

```
>beadID,<coordinate1>,<coordinate2>,<coordinate3>
```

If the read does not hit the reference sequence, no coordinates are listed.

Each <coordinate> has the following format:

```
<hit info>#<list of valid adjacent mismatches>#<list of insertions in read>#<List of insertions in reference>
```

where:

<hit info> is made up of:

```
<refID>.<referencePosition>.<alignmentStartPosition>.<alignmentLength>.<score>.<numberOfMismatches>
```

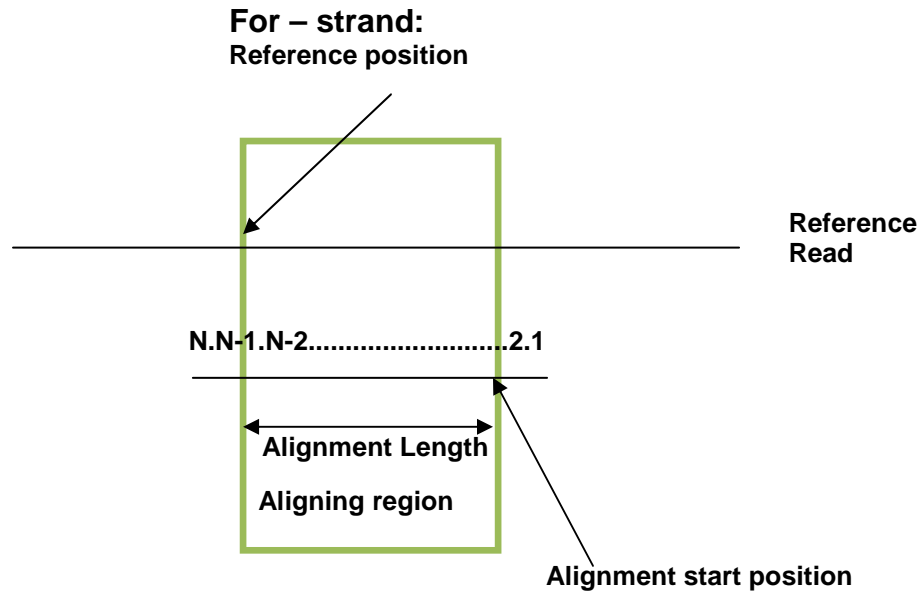
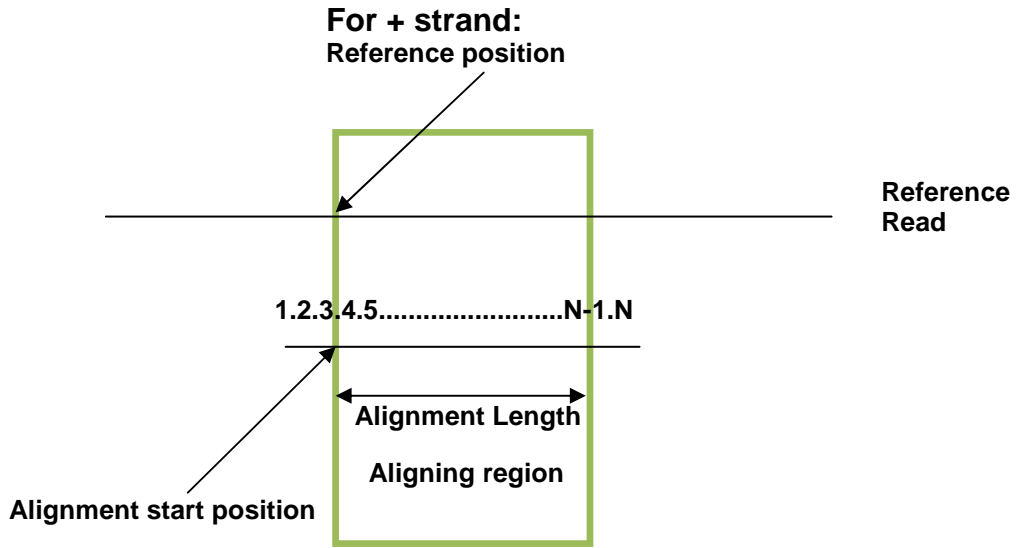
- `refID`  
One-based integer specifying the index of the sequence entry in the reference file.
- `referencePosition`  
A zero-based integer specifying the most 5' prime aligned base on the plus strand of the reference. A value preceded by a minus sign indicates alignment made with the minus strand, but still specifies a base position on the plus strand. (See Fig.8.)
- `alignmentStartPosition`  
One-based integer specifying the position of the first aligned color in the read. (See Fig. 8.)
- `alignmentLength`  
The length of the alignment. (See Fig. 8.)
- `score`  
The score of the alignment.
- `numberOfMismatches`  
The number of single color mismatches in the alignment, including mismatches that are part of a valid adjacent mismatch.

**Note: The following are not currently used:**

- list of valid adjacent mismatches
- list of insertions in read
- list of insertions in reference

See the test dataset posted on <http://www.solidsoftwaretools.com> for an example of this GTF file.

Fig 8. Explanation of alignment terms in the <hit info> section of the max file.



Example:

```
>6_385_1653_F3,11.11229299.1.45.41.2###,12.-51245229.1.50.48.1###
```

In the above example, the read aligns with two places in the reference.

The first alignment is with the plus strand of sequence #11. The alignment begins with base-position 11229299 of sequence #11 and is 45 colors long. It scores 41 and has 2 mismatches.

The second alignment is with the minus strand of sequence #12. The alignment ends with base position 51245229 of the plus strand of sequence #12 and is 50 colors long. It scores 48 and has one mismatch.

### Licensing

This software is being licensed to you under the OSI compliant GNU Public License (GPL V3). The license can be found at the following URL: <http://www.gnu.org/licenses/gpl.html>. Please read the license in its entirety and ensure that you understand the licensing conditions for use. Your use of this software indicates your acceptance of this licensing agreement.

© 2009 Life Technologies Corporation. All rights reserved.

The trademarks mentioned herein are the property of Life Technologies Corporation or their respective owners.