SMRT™ Pipe Reference Guide

Introduction

SMRT™ Pipe is Pacific Biosciences' underlying analysis framework for secondary analysis functions. SMRT**™** Pipe is a python-based general-purpose workflow engine, and is lightweight, easily extensible, and provides support for common analysis concerns such as logging, distributed computation, error handling, analysis parameters, and temporary files.

In a typical installation of the SMRT™ Analysis Software, the SMRT™ Portal web application calls SMRT™ Pipe when a job is started. SMRT™ Portal provides a convenient and user-friendly way to analyze PacBio sequencing data through SMRT™ Pipe. Power users will find that there is more flexibility and customization available by instead running SMRT™ Pipe analyses from the command line.

SMRT™ Pipe can also be accessed by way of the Secondary Analysis Web Services API. For details, see the document **Secondary Analysis Web Services API**.

This document describes the underlying command-line interface to SMRT™ Pipe, and is for use by bioinformaticians working with secondary analysis results.

Note: Throughout this document, the path `/opt/smrtanalysis` is used to refer to the installation directory for SMRT™ Analysis (aka `$SEYMOUR_HOME`). Replace this path with the path appropriate to your installation when using this document.

Installation

SMRT™ Pipe is installed as part of the SMRT™ Analysis software installation. For details, see the document **SMRT™ Analysis Software Installation.**

Using the Command Line

In a typical installation of SMRT™ Analysis, SMRT™ Pipe is in your path after sourcing the `setup.sh` file. To do so, enter:

```
. /opt/smrtanalysis/etc/setup.sh
```

(Make sure to replace `/opt/smrtanalysis` with the path to your SMRT™ Analysis installation).

You can check that SMRT™ Pipe is available by entering the following:

```
smrtpipe.py --help
```

This displays a help message describing how to run `smrtpipe.py` and all of the available command-line options.

You invoke SMRT™ Pipe with the following command:

```
smrtpipe.py [--help] [options] --params=settings.xml
xml:inputFile
```

Logging messages are printed to `stderr` as well as a log file (`log/smrtpipe.log`). It is standard practice to pipe the stderr messages to a file using redirection in your shell, for example appending `&> smrtpipe.err` to the command line if running under `bash`.

Command-Line Options

The following table describe the available options for invoking `smrtpipe.py`:

Option	Description
`-D key=value`	Overrides a configuration variable. Configuration variables are key-value pairs that are read from the global `smrtpipe.rc` before starting an analysis. An example is the `NPROC` variable which controls the number of simultaneous processors to use during the analysis. To restrict SMRT™ Pipe to 4 processors, use `-D NPROC=4`.
`--debug`	Activates debugging output in the stderr and log outputs. This flag can be set as a default by setting `DEBUG=True` in the `smrtpipe.rc` file.
`--distribute`	Distributes the computation across a compute cluster. Configuring SMRT™ Pipe for a distributed computation environment is discussed in the document **SMRT™ Analysis Software Installation**.
`--help`	Displays information about command-line usage and options, and then exits.
`--noreports`	Turns off the production of XML/HTML/PNG reports.
`--nohtml`	Turns off the conversion of XML reports into HTML. (This conversion requires that Java be installed.)
`--output=outputDir`	Specifies a root directory to use for all SMRT™ Pipe outputs for this analysis. SMRT™ Pipe will place outputs in this directory, as well as in data, results, and log subdirectories.
`--params=paramsXml`	Specifies a settings XML file for running the pipeline analysis. If this option is **not** supplied, SMRT™ Pipe will print the usage message and then exit.
`--totalCells`	Specifies that if the number of cells in the job is less than `totalCells`, the job is **not** marked complete when it finishes. Data from additional cells will be appended to the outputs, until the number of cells reaches `totalCells`.
`--recover`	Attempts to rerun a SMRT™ Pipe analysis starting from the last successful stage. The same initial arguments should be specified in this case.
`--version`	Displays the version number of SMRT™ Pipe and then exits.

Specifying SMRT™ Pipe Inputs

The input file is an XML file which specifies the PacBio RS sequencing data to process. Generally, you specify the inputs as URIs (Universal Resource Identifiers) which are resolved by code internal to SMRT™ Pipe. In practice, this logic is most useful to large enterprise users that have a data management scheme and are able to modify the SMRT™ Pipe code to include their own resolver.

The simpler way to specify inputs is to fully resolve the path to each input file, which is almost always a `bas.h5` file. The script `fofnToSmrtpipeInput.py` is provided to convert a file of `bas.h5` file names (a "fofn" file) to the input format expected by SMRT™ Pipe. If `my_inputs.fofn` looks like

```
/share/data/run_1/m100923_005722_00122_c15301919401091173_s0_p0.bas.h5
/share/data/run_2/m100820_063008_00118_c04442556811011070_s0_p0.bas.h5
```

then it can be converted to a SMRT™ Pipe input XML file by entering:

```
fofnToSmrtpipeInput.py my_inputs.fofn > my_inputs.xml
```

Following is the resulting XML file:

```
<?xml version="1.0"?>
<pacbioAnalysisInputs>
 <dataReferences>
    <url ref="run:0000000-0000"><location>/share/data/
    run_1 m100923_005722_00122_c15301919401091173_s0_
    <url ref="run:0000000-0001"><location>/share/data/
    run_2/m100820_063008_00118_c04442556811011070_s0_
 </dataReferences>
</pacbioAnalysisInputs>
```

You can now run an analysis using these two `bas.h5` files as input by entering the following command:

```
smrtpipe.py --params=settings.xml xml:my_inputs.xml
```

The SMRT™ Pipe input format provides for specifying annotations, such as job IDs, job names, and job comments, in a job-management environment. The `fofnToSmrtpipeInput.py` application has command-line options for setting these optional attributes if desired.

Note: To get help for a script, execute the script with the `--help` option and no additional arguments. For example:

```
fofnToSmrtpipeInput.py --help
```

Specifying SMRT™ Pipe Parameters

The `--params` option is the most important SMRT™ Pipe option, and is required for any sophisticated use. The `--params` option specifies an XML file that controls:

- The analysis modules to run.

- The order of execution.
- The parameters used by the modules.

The general structure of the settings XML file is as follows:

```
<?xml version="1.0"?>
<smrtpipeSettings>

<protocol>
...global parameters...
</protocol>

<module id="module_1">
...parameters...
</module>

<module id="module_2">
...parameters...
</module>

</smrtpipeSettings>
```

- The `protocol` element allows the setting of global parameters that could possibly be used by all modules.
- Each `module` element defines an analysis module to run.
- The order of the `module` elements defines the order in which the modules execute.

The module IDs correspond to the file name (equivalently class name) of the analysis module. In SMRT™ Pipe v1.1.1, the known module names are:

- `Assembly`
- `HybridAssembly`
- `StrobeScaffolder`
- `AssemblyStats`
- `AssemblyRefinement`
- `S_Filter, S_FilterReports`
- `S_Control, S_ControlReports`
- `S_Mapping, S_MappingReports`
- `S_Consensus, S_ConsensusReports`
- `S_RCCS`
- `S_AnalysisHook`

Parameters are specified by a key-value pair presented in a `param` element. The name of the key is in the name attribute of the `param` element, and the value of the key is contained in a nested value element. For example, to set the parameter named `reference`, you specify:

```
<param name="reference">
  <value>/share/references/repository/celegans</value>
</param>
```

Note: The value of a parameter can be referenced in other parameters by using the notation `${variable}` when specifying a value. For example, a global parameter named `home` can be referenced in other parameters as `${home}`. SMRT™ Pipe supports arbitrary parameters in the settings XML file, so the use of temporary variables like this can help readability and maintainability.

Following is a complete example of a settings file for running filtering, mapping, and consensus steps against the E coli reference genome:

```
<?xml version="1.0" encoding="utf-8"?>
<smrtpipeSettings>
 <protocol>
  <param name="reference">
   <value>/share/references/repository/ecoli</value>
  </param>
 </protocol>

 <module name="S_Filter">
  <param name="minLength">
    <value>50</value>
  </param>
  <param name="readScore">
    <value>0.75</value>
  </param>
 </module>

 <module name="S_FilterReports" />

 <module name="S_Mapping">
  <param name="align_opts" hidden="true">
   <value>--minAccuracy=0.75 --minLength=50 -x </value>
  </param>
 </module>

 <module name="S_MappingReports" />
 <module name="S_Consensus" />
 <module name="S_ConsensusReports" />

</smrtpipeSettings>
```

See below for information about how to specify a reference entry; the parameter named `reference` in this example.

The SMRT™ Pipe configuration file is named `smrtpipe.rc`, found at `/opt/smrtanalysis/analysis/etc/smrtpipe.rc`. This file is formatted similarly to a .ini file:

- Comment lines begin with #.
- Variables are defined using the format `VARIABLE=a_value`.

The following table lists useful configuration variables:

Variable Name	Default Value	Description
CLUSTER_MANAGER	SGE	Specifies the name of a cluster management system for submitting and killing jobs.
DEBUG	False	Setting this to `true` specifies that the default mode of `smrtpipe.py` is to act as if the `--debug` option is specified.
NJOBS	64	Specifies the total number of processors that SMRT™ Pipe uses when called with the `--distribute` option on a cluster.
NPROC	8	Specifies the default number of processors that SMRT™ Pipe uses when calling multi-core code or executing simultaneous single-core processes.
SHARED_DIR	/mnt/secondary/Share/tmp	In distributed analysis, this should be set to the path of a shared writeable directory visible to all nodes.
TMP	/tmp	Specifies the root location for creation of temporary files and directories. For optimal performance this should be a locally mounted file system.

Note: You can override configuration variables at the command-line prompt when running `smrtpipe.py` by specifying the `-D` option.

SMRT™ Pipe Modules and Parameters

Following is an overview of some of the common modules included in SMRT™ Pipe and their parameters. Not all modules or parameters are listed here. Developers interested in even finer control are encouraged to look inside the `validateSettings` method for each python analysis module. By convention, all of the settings known to the analysis module are referenced in this method.

SMRT™ Pipe Outputs

SMRT™ Pipe outputs should use standard open formats as much as possible:

- HDF5 files (`.hdf5`) can be viewed in the freely available HDFView software and accessed with a variety of APIs in common languages.
- CSV and GFF3 files are the preferred standards for representing tabular data and genome-anchored data.

Global Parameters

Global parameters are potentially used in multiple modules. In the SMRT™ Pipe internals they are accessed in the “global” namespace.

The following table lists common global parameters:

Parameter	Default Value	Description
`Reference`	`None`	Specifies the name of a reference repository entry or FASTA file for mapping reads. (**Required** for resequencing workflows.)
`Control`	`None`	Specifies the name of a reference repository entry or FASTA file for mapping spike-in control reads. (**Optional**)
`use_subreads`	`True`	Specifies whether to divide reads into subreads using the adapter region boundaries found by the primary analysis software. (**Optional**)
`num_stats_regions`	`500`	Specifies how many regions to use when reporting region statistics such as depth of coverage and variant density. (**Optional**)

S_Filter Module

This module filters and trims the raw reads produced by Pacific Biosciences' primary analysis software. Options are available for taking the information found in the `bas.h5` files and using this to pass reads and portions of reads forward.

A useful output of the S_Filter module is the `data/filtering_summary.csv` file. This file includes raw metrics and filtering information for each read found in the original `bas.h5` files. The filtering information generated by this module is stored in files named `rgn.h5` files, one for each input `bas.h5` file.

The following table lists the S_Filter module parameters:

Parameter	Default Value	Description
`minLength`	`None`	Reads with a high quality region readlength below this threshold are filtered out. (**Optional**)
`readScore`	`None`	Reads with a high quality region (Read Quality) score below this threshold are filtered out. (**Optional**)
`trim`	`True`	Specifies whether or not to trim reads to the high-quality region. (**Optional**)

S_Mapping Module

This module aligns reads against a reference sequence, possibly a multi-contig reference. If the S_Filter module is run **first**, then **only** the reads which passed filtering will be aligned.

The main outputs of the S_Mapping module are:

- `data/aligned_reads.cmp.h5`: The pairwise alignments for each read.
- `data/alignment_summary.gff`: Summary information.

The following table lists the S_Mapping module parameters:

Parameter	Default Value	Description
`align_opts`	`Empty String`	A parameter sent to the underlying `compareSequences.py` script to pass options. (**Optional**)
`--useCcs=`	`None`	A parameter sent to the underlying `compareSequences.py` script. Values = {`denovo`|`fullpass`|`allpass`} (**Optional**) • `denovo`: Maps just the *de novo* called sequence and report. (Does **not** include quality values.) • `fullpass`: Maps the *de novo* called sequence, then aligns full passes to the sequence that the *de novo* called sequence aligns to. • `allpass`: Maps the *de novo* called sequence, then aligns all passes (even ones that don't span the length of the template) to the sequence the *de novo* called sequence aligned to.
`load_pulses`	`True`	Specifies whether or not to load pulse metric information into the `cmp.h5` file. (**Optional**)
`maxHits`	`None`	Attempts to find sub-optimal alignments and report up to this many hits per read. (**Optional**)
`minAnchorSize`	`None`	Ignores anchors smaller than this size when finding candidate hits for dynamic programming alignment. (**Optional**)
`output_bam`	`False`	Specifies whether or not to output a BAM representation of the `cmp.h5` file. (**Optional**)
`output_sam`	`False`	Specifies whether or not to output a SAM representation of the `cmp.h5` file. (**Optional**)
`writeBED`	`False`	Specifies whether or not to output a BED representation of the depth of coverage summary. (**Optional**)

S_Consensus Module

This module takes the alignments generated by the mapping module and calls the consensus sequence across the reads.

The main outputs of S_Consensus are:

- `data/aligned_reads.cmp.h5`: The consensus sequence.
- `data/variants.gff.gz`: A gzipped GFF3 file containing variants versus the reference.

Other useful information about variants can be found in `data/alignment_summary.gff` and `data/variants.vcf`.

The following table lists the S_Consensus module parameters:

Parameter	Default Value	Description
`minCov`	`4`	The algorithm will only be invoked for contigs in the reference where the average depth of coverage is greater than or equal to this threshold. A faster plurality algorithm is used for low-coverage reference regions. (**Optional**)
`minVariantQual`	`0`	Only variants with a quality above this threshold will be called by the consensus-calling algorithm. (**Optional**)
`writeBED`	`True`	Specifies whether or not to output a BED representation of the variants. (**Optional**)
`writeVCF`	`True`	Specifies whether or not to output a VCF representation of the variants. (**Optional**)

S_RCCS Module

This module aligns circular consensus sequencing (CCS) reads against a reference sequence and attempts to find the best consensus call for each molecule. The module can be used for more sensitive SNP detection and to find rare variants.

- This module has **no** exposed parameters.

Following is a description of the report files generated by the module, which can be used for tertiary analysis.

`data/RCCS.cmp.h5`

- Contains the alignment of subreads for RCCS. The file includes **only** the consensus read aligned to the reference. It uses a set of parameters that penalizes indels more than in the file `RCCS_sr.cmp.h5`, as different alignment parameters are used.

`data/RCCS_count.gff`

- Contains reference indexed variant calls, in GFF format. The file has a summary of the consensus results for each base in the reference sequence. The file contains the count of different alleles called at a given location.

`data/RCCS.fasta`

- Contains RCCS calls (the reconstructed consensus sequence information), one per ZMW, in FASTA format. This file should be filtered by extracting all RCCS calls which used 3 or more subreads.
  - `rCCS` means the read is a RCCS consensus sequence.
  - `CCC =1` means that the circular consensus coverage is 1. That is, 1 subread was used to produce this RCCS subread.

  - `Chr1` refers to the target sequence used for the “reference”.

`data/RCCS.fastq`

- Contains RCCS calls (the reconstructed consensus sequence information), one per ZMW, in FASTQ format. This file should be filtered by extracting all RCCS calls which used 3 or more subreads.
  - `rCCS` means the read is a RCCS consensus sequence.
  - `CCC =1` means that the circular consensus coverage is 1. That is, 1 subread was used to produce this RCCS subread.
  - `Chr1` refers to the target reference used for the “reference”.

`data/RCCS.lqv`

- This file is meant for internal sanity checking and should **not** be used.
  - Column 1 is the log-likelihood ratio binned to integer. (For more information, see **http://nar.oxfordjournals.org/content/early/2010/06/22/nar.gkq543.abstract**.)
  - Column 2 is the total amount of basecall in each bin.
  - Column 3 is the total amount of “miscall”, assuming that the reference and the sample agree with each other.
  - Column 4 is the empirical phred scale QV calculated using Columns 2 and 3.

`data/RCCS_pi.gz`

- A compressed tab-delimited file that contains detailed information for each base in each molecule used in the analysis. Contains a list of variant calls in detail. `pi` stands for “per-base info”. This is the major file from which all the other files are derived.
  - The columns in the file are: `readId`, `referenceId`, `CCC`, `targetPosition`, `reference base`, `read position`, `read base`, and `log-likelihood`.

`data/RCCS.sam`

- Contains the subreads used by RCCS.

`data/RCCS_sr.cmp.h5`

- Contains the alignments of the subreads used to generate the consensus aligning to the reference sequences. Includes all the “raw-subread” aligned to the reference. This file is more permissive for indel detection than the `RCCS.cmp.h5` file as different alignment parameters are used.

S_AnalysisHook Module

This module allows you to call executable code as part of a SMRT™ Pipe analysis. S_AnalysisHook can be called multiple times in a settings XML file, allowing for an arbitrary number of calls to external (non-SMRT™ Pipe) code.

The following table lists the S_AnalysisHook module parameters:

Parameter	Default Value	Description
`scriptDir`	`None`	All executables in this directory are called serially with the command line `exeCmd jobDir`, where `jobDir` is the root of the SMRT™ Pipe output for this analysis. (**Optional**)
`script`	`None`	Path to an executable which will be called with the command line `exeCmd jobDir`, where `jobDir` is the root of the SMRT™ Pipe output for this analysis. (**Optional**)

Assembly Module

This module takes the trimmed reads which pass filtering and attempts to assemble them into contiguous sequences (contigs).

The main outputs of the Assembly module are:

- `data/assembled.fsta`: A FASTA file containing the assembled contig consensus sequences.
- `data/assembled_reads.cmp.h5`: The pairwise alignments for each read against its assembled contig consensus.
- `data/assembled_summary.gff`: Summary information about each of the contigs.

Optional outputs include:

- `data/assembled.ace`: The assembly, in ACE format.
- `data/assembled.bnk.tar.gz`: The assembly, as a compressed AMOS bank.

The following table lists the Assembly module parameters:

Parameter	Default Value	Description
genomeSize	100,000 bases	A rough estimate of the expected size of this genome. This is used to provide an estimate of expected coverage and to modulate parameters based on genome size. (**Optional**)
maxIterations	10	Specifies the maximum number of iterations for progressive assembly. (**Optional**)
minIterations	4	Specifies the minimum number of iterations for progressive assembly. (**Optional**)
outputAce	False	Specifies whether or not to output an ACE representation of the assembly. (**Optional**)
outputBank	False	Specifies whether or not to output an AMOS bank representation of the assembly. (**Optional**)
overlapScoreThreshold	700	The score threshold for accepting an overlap between two reads. The suggested range is between 300-1500, with 700 being a typical threshold. (**Optional**)

HybridAssembly Module

This module scaffolds high-confidence contigs, such as that from Illumina data, using PacBio strobe or long reads. It produces two outputs: A FASTA file containing the scaffolded contigs and a GraphML file containing the scaffold graph.

The HybridAssembly module takes two arguments:

- `params.xml`: Specifies the parameters to run.
- `input.xml`: Specifies the inputs.

The module produces two main outputs:

- `data/scaffold.gml`: A GraphML file that contains the final scaffold. This file can be readily parsed in python using the `networkx` package.
- `data/scaffold.fsta`: A FASTA file with a single entry for each scaffold.

To run `HybridAssembly.py`, enter the following:

```
smrtpipe.py --params=params.xml xml:input.xml >&
smrtpipe.err
```

Unlike other SMRT™ Pipe modules, `HybridAssembly.py` uses two kinds of input instead of one:

- A FASTA file of high-confidence sequences which are to be scaffolded. These are typically contigs assembled from Illumina short read sequence data.

- PacBio strobe or long reads, in HDF5 or FASTA format. These are used to join the high-confidence contigs into a scaffold.

Following is a sample `input.xml` file:

```
<?xml version="1.0"?>

<pacbioAnalysisInputs>

<dataReferences>
<!-- High-confidence sequences fasta file -->
<url ref="assembled_contigs:test_contigs.fsta"/>

<!-- PacBio reads, either in fasta or in bas.h5
format. -->
<url ref="file:test_reads.fsta" />

</dataReferences>

</pacbioAnalysisInputs>
```

The HybridAssembly module can now be run in two modes, using long or strobe reads. Here is a sample `params.xml` file for **long** reads, with only customer-facing parameters:

```
<?xml version="1.0"?>
<smrtpipeSettings>

  <!-- HybridAssembly 1.2.0 parameter file for long reads -->
  <module name="HybridAssembly">

  <!-- General options -->
  <!-- Parameter schedules are used for iterative hybrid assembly. They are
       given in comma delimited tuples separate by semicolons. The fields
       in order are:

       - Minimum alignment score (aka Z-score). Higher is more stringent.
       - Minimum number of reads needed to link two contigs. (Redundancy)
       - Minimum subread length to participate in alignment.
       - Minimum contig length to participate in alignment.

       If a tuple contains less than 4 fields, defaults will be used for
       the remaining fields. -->
  <paramSchedule>6,3,75;6,3,75;6,2,75;6,2,75</paramSchedule>

  <!-- Untangling occurs after the main scaffolding step. Valid values
       are "bambus" and "pacbio" (recommended and the default). -->
  <untangler>pacbio</untangler>

  <!-- Gap fillin can be turned on by leaving the option out, or setting it
       to False. -->
  <dontFillin>True</dontFillin>

  <!-- These options allow long reads -->
  <longReadsAsStrobe>True</longReadsAsStrobe>
  <exactQueryIds>True</exactQueryIds>
  <rm4Opts>-minMatch 6 -minPctIdentity 60 -bestn 10 \
  -noSplitSubreads</rm4Opts>

  <!-- Parallelization options -->
  <numberProcesses>16</numberProcesses>
  </module>
</smrtpipeSettings>
```

Note: If cutting and pasting the line starting with `<rm4Opts>`, you may get a newline character after `-bestn 10`. **Delete** the newline character, or the code may not work correctly.

Here is a sample `params.xml` file for **strobe** reads, with only customer-facing parameters:

```
<?xml version="1.0"?>
<smrtpipeSettings>

    <!-- HybridAssembly 1.2.0 parameter file for *strobe* reads -->

    <module name="HybridAssembly">
    <untangler>pacbio</untangler>
    <paramSchedule>6,3,75;6,3,75;6,2,75;6,2,75</paramSchedule>
    <dontFillin>True</dontFillin>
    <rm4Opts>-minMatch 6 -minPctIdentity 60 -bestn 10 \
     -noSplitSubreads</rm4Opts>

    <!-- Parallelization options -->
    <numberProcesses>16</numberProcesses>

    </module>

</smrtpipeSettings>
```

Known Issues

- There is a known bug in gap fillin in that it can exceed the recursion limit of python. Gap fillin can be turned off by adding the following option to the `params.xml` file:

  ```
  <dontFillin>True</dontFillin>
  ```

- Depending on the repetitive content of the high-confidence input contigs, a large fraction of the sequence in the contigs can be called repeats. To avoid this, turn off the split repeats step by setting the minimum repeat identity to a number greater than 100, for example:

  ```
  <minRepeatIdentity>1000</minRepeatIdentity>
  ```

SMRT™ Pipe Tools

Tools are programs that run as part of SMRT™ Pipe. A module, such as S_Mapping, can call several tools (such as the mapping tools `summarizeCoverage.py` or `compareSequences.py`) to actually perform the underlying processing.

BLASR

Description

BLASR is a SMRT™ Pipe tool that maps reads to positions in a genome by clustering short exact matches between the read and the genome, and then scoring clusters using alignment.

The matches are generated by searching all suffixes of a read against the genome, using a suffix array. Global chaining methods are used to score clusters of matches. It is very useful to have read filtering information, and mapping runtime may decrease substantially when you specify a precomputed suffix array index on the reference sequence.

Read filtering information is contained in the `.bas.h5` input files, as well as generated by other post-processing programs with analysis of pulse files and read in from a separate `.rgn.h5` file. The current set of filters applied to reads are high quality region filtering, and adapter filtering.

- Regions **outside** high-quality regions are ignored in mapping.
- Reads that contain regions annotated as adapter are split into non-adapter (template) regions, and mapped separately.

If you specify a suffix array index of a genome, the suffix array is built **before** producing alignment. This may be prohibitively slow when the genome is large (for example, human). It is best to precompute the suffix array of a genome by using the program `sawriter`, and then specifying the suffix array by entering `-sa genome.fa.sa`.

You can specify several parameters to speed up alignments, at the expense of possibly decreasing sensitivity.

Syntax

```
blasr [reads.fasta|reads.bas.h5|reads.pls.h5]
genome.fasta [options]
```

Input Files

The only required inputs to BLASR are a file of reads and a reference genome, or a file containing the names of reads files.

- `reads.fasta` is a multi-fasta file of reads.

- `reads.pls.h5|reads.bs.h5` is the native output format in Hierarchical Data Format of SMRT reads. This is the preferred input to BLASR as rich quality value information (insertion, deletion, and substitution quality values) is maintained. The extra quality information improves mapping speed, and produces higher quality variant detection.

Options - Extra Input Files

`-sa suffixArrayFile`

- Use the array file `suffixArrayFile` for detecting matches between the reads and the reference. You create the suffix array using the `sawriter` program.

`-ctab tab`

- This is a table of tuple counts used to estimate match significance, generated by the `printTupleCountTable` program. It is useful to precompute the ctab if there are many invocations of BLASR.

`-regionTable table`

- Load a read-region table in HDF format for masking portions of reads. This may be a single table if there is just one input file, or a “fofn” file if there are multiple input files.
- If you specify a region table, those region tables inside the `reads.pls.h5` or `reads.bas.h5` files are ignored.

Options - Masking Reads

Ancillary information about substrings of reads is stored in a region table for **each** read file. The region table may be part of the `.bas.h5` or `.pls.h5` file, or a separate file.

A contiguously read substring from the template is a **subread**; any read may contain multiple subreads. The boundaries of the subreads can be inferred from the region table either directly or by definition of adapter boundaries. Typically, region tables also contain information for the location of the high and low quality regions of reads. Reads produced by spurious reads from empty ZMWs have a high quality start coordinate equal to high quality end, making no usable read.

`-noSplitSubreads` (Default value = `false`)

- Do **not** split subreads at adapters.

`-ignoreRegions` (Default value = `false`)

- Ignore any information in the Region table.

`-ignoreHQRegions` (Default value = `false`)

- Include the **full** read in an alignment, not just the regions annotated as high quality.

Options - Alignment Output

`-bestn n` (Default value = `10`)

- Report the top `n` alignments.

`-printFormat t`

- Modify the output of the alignment.
  - `t 0`: Print human-readable alignments.
  - `t 1`: Print only a summary: score and position.
  - `t 2`: Print in Compare.xml format.
  - `t 3`: Print in Vulgar format.
  - `t 5`: Print an easily parsed version of the alignment for `compareSequences`.
  - `t u`: Print in tabular format with many values.

`-out`

`OUT` (Default value = `terminal`)

- Write output to OUT.

`-noSortRefinedAlignments` (Default value = `false`)

- Once candidate alignments are generated and scored via sparse dynamic programming, they are rescored using local alignment that accounts for different error profiles. Resorting based on the local alignment may change the order the hits are returned.

`-titleTable tab` (Default value = `null`)

- Create a table of reference sequence titles. The reference sequences are enumerated by row, 0,1,… The reference index is printed in the alignment results instead of the full reference name. This makes output concise, especially when reference names are long.

`-minPctIdentity p` (Default value = `0`)

- Only report alignments if they are greater than `p` percent identity.

`-unaligned UNALIGNEDFILE`

- Output reads that are not aligned to the file `UNALIGNEDFILE`.

`-metrics METRICSFILE`

- Output alignment metrics to the file `METRICSFILE`. This is used for debugging and measuring speed.

Options - Anchoring Alignment Regions

These options will have the greatest effect on speed and sensitivity. The default anchoring parameters are optimal for small genomes and samples with up to 5% divergence from the reference genome.

`-minMatch m` (Default value = `10`)

- Minimum seed length is `m`. This is the **main** option governing speed and sensitivity. For human genome alignments, a value of 11 or higher is recommended.

`-maxExpand M` (Default value = `1`)

- Perform no more than `M` iterations of searches through the suffix array for matches. At each iteration, all matches of length LCP[i]-M are found, where LCP[i] is the length of the longest common prefix between the string at i and anywhere in the genome. The number of matches grows as `M` increases, and can become very large with `M` > `3`.
- If the genome is highly repetitive or divergent from the read sequences, the value of `-maxExpand` should be increased. This option controls how much the search for anchors is expanded past a simple greedy search. A value for `-maxExpand` of 1 is sufficient for non-repetitive genomes. Values greater than 5 are **not** recommended.

`-maxLCPLength l`

- Stop mapping a read to the genome when the LCP (longest common prefix) length reaches `l`. Shorter `l` means more matches.

`-skipLookupTable`

- Do **not** use a lookup table to limit suffix array searches.

`-maxAnchorsPerPosition m`

- Do **not** add anchors from a position if it matches to more than `m` locations in the target. Regions that are too repetitive may be ignored during mapping by limiting the number of positions a read maps to using this option. Values between 500 and 1000 are effective in the human genome. For small genomes such as bacterial genomes, the default parameters are sufficient for maximal sensitivity and good speed.

`-advanceHalf` (Default value = `false`)

- This speeds up alignments, at the cost of sensitivity. Rather than clustering matches for every position in the genome, clustering skips forward by half the size of a cluster.

`-advanceExactMatches l` (Default value = `0`)

- Used to speed alignments. Rather than searching for alignments at every position in a read, `m-l` positions are skipped, where `m` is the length of the match.

`-nCandidates n` (Default value = `10`)

- Try up to `n` candidates for the best alignment. A large value of `n` will slow mapping because the slower dynamic programming steps are applied to more clusters of anchors, which can be a rate-limiting step when reads are very long.
  Tip: It may be necessary to increase `nCandidates` when the genome is highly repetitive.

Options - Refining Hits

`-noRefinAlign` (Default value = `false`)

- Do **not** run dynamic programming refinement.

`-indel i` (Default value = `4`)

- Penalty for insertions and deletions.

`-match m` (Default value = `-4`)

- Bonus for matching two nucleotides. This is ignored when aligning `bas.h5` or `pls.h5` files.

`-maxScore m` (Default value = `0`)

- Maximum score to output. A high score is bad, and a negative score is good.

`-sdpTupleSize K` (Default value = `0.6`)

- Use matches of length `K` to seed sparse dynamic programming alignments. This controls accuracy of assigning gaps in pairwise alignments once a mapping has been found, rather than mapping sensitivity itself.

`-fullRead`

- Force alignments of entire reads rather than finding high scoring local substrings. This is useful for extremely repetitive genomes.

Options - Filtering Reads

`-minReadLength l` (Default value = `50`)

- Use a minimum read length of `l`.

`-minSubreadLength l` (Default value = `0`)

- Do **not** align subreads of length less than `l`.

`-minAvgQual q` (Default value = `0`)

- Do **not** align reads if the average base quality is less than `q`.

Options - Parallel Alignment

`-nproc N` (Default value = `1`)

- Align using `N` processes. All large data structures such as the suffix array and tuple count table are shared.

`-start S` (Default value = `0`)

- Index of the first read to begin aligning. This is useful when multiple instances of BLASR are running on the same data, for example when on a multi-rack cluster.

`-stride S` (Default value = `1`)

- Align one read every `S` reads.

Options - Subsampling Reads

`-subsample S` (Default value = `0`)

- Proportion of reads (expressed as a decimal) to randomly subsample and align.

Options - Overlap/ Dynamic Programming Alignments

`-overlap`

- Force alignments that overlap the ends of contigs to extend to the end of the contig.

`-noFrontAlign`

- Do **not** extend alignments beyond the front of the first high scoring anchor.

`-ignoreQuality`

- Ignore quality values when computing alignments.

Options - Debugging

`-v`

- Print verbose information.

`-V i`

- Print more verbose information. As `i` increases, the text becomes more verbose.

`-help`

- Prints a more verbose help page.

`-h`

- Prints a MAN page.

BLASR Examples

- `blasr reads.fasta genome.fasta -sa genome.fasta.sa`
- `blasr reads.bas.h5 genome.fasta`
- `blasr reads.bas.h5 genome.fasta -sa genome.fasta.sa -maxScore -100 -minMatch 15`
- `blasr reads.bas.h5 genome.fasta -sa genome.fasta.sa -nproc 24 -out alignment.out …`

EviCons

Description

Typically, resequencing analysis workflow includes mapping of reads to a reference genome or target region, followed by consensus calling of aligned reads. **Consensus calling** allows researchers to establish variations between the DNA sample and reference sequence, some of which may have important implications for disease research.

EviCons (Evidence-Based Consensus) is a SMRT™ Pipe tool for producing a consensus sequence given an alignment of reads. The consensus is the best estimate of the genomic sequence in the DNA sample. EviCons can analyze human genomes having a maximum of 60 fold coverage.

Algorithm

EviCons uses a hybrid approach for consensus calling. High certainty columns in the MSA, where bases observed tend to agree, are marked as **posts** and called on a per-column basis. The algorithm used for identifying and calling posts is fast considering that this logic will suffice for the vast majority (> 99%) of positions in the alignment. The region between two consecutive posts, designated as an island, requires a more rigorous algorithm as the evidence presented in these regions is more ambiguous.

Input Files

EviCons is a standalone, command-line application. Although it includes many user options and modes, EviCons only requires one parameter to run with default settings: Either an HDF5 file containing mapped reads, or a multiple sequence alignment file.

- If an **HDF5** file is the input, a multiple sequence alignment is constructed from the pair-wise alignments using the Data Access Object (DAO). The DAO translates compressed alignment information and creates a star alignment using the reference as center.
- If a **multiple sequence alignment file** is the input, the representation should describe mapped reads (composed of bases and gaps) in aligned reference coordinates. Ideally, mapped reads should carry a unique name and sequencing orientation. In the absence of orientation information, forward orientation is assumed.

Two possible multiple sequence alignment representations are currently supported by EviCons:

- `[.efa]` This format is considerably more compact by eliminating gaps providing no information gain, which is crucial for larger genome sizes.
- `[.fa]`

Aside from the mandatory alignment information, users can also specify a combination of parameters to better suit their purpose.

Syntax

```
jConnectPost [OPTIONS] [CMPH5FILE]
```

Options

`-h, –help`

- Displays a MAN page, then exits.

`–resultDir=RESULTDIR`

- Specifies the absolute path of the directory where to write consensus results.

`–confFile=CONFFILE`

- Specifies the absolute path to the output file containing the consensus confidence values.

`–debug`

- Enables debugging output.

`–fastMode`

- Specifies that the island size be limited to an upper bound (`length=8`) to keep computation time tractable.

`–randomNumberSeed=RANDOMNUMBERSEED`

- Integer seed used as a starting point for random number generation.

Options - Alignment Refinement

`-refineAln`

- Specifies whether to perform MSA refinement prior to consensus calling. Before consensus calling takes place, this refines the alignment provided by the user or generated by the DAO (if using HDF5 input).
- If set to `true`, the initial alignment is optimized via simulated annealing, which seeks to maximize an objective function that measures the “fitness” of the alignment from the perspective of consensus calling.

Options - Alignment Processing

`–subAlignment`

- Specifies whether to compute consensus for partial alignment.

`–refStart=REFSTART`

- Specifies the start position in a reference (unaligned coordinates, 0-indexing) if doing partial alignment consensus.

—refEnd=REFEND

- Specifies the inclusive end position in a reference (unaligned coordinates, 0-indexing) if doing partial alignment consensus.

By default, EviCons will process the entire alignment from start to finish. However, if `subAlignment` is true, you can specify `refStart` and `refEnd` to generate consensus for a targeted piece of the alignment. Coordinates are in aligned reference space.

Options - Base to Dye Translation

`-baseMap=BASEMAP` (Default value = `1234`)

- Comma-separated base map string: Dye numbers for (A,C,G,T)

The dyes used for base labeling can vary from one experiment to another. It is important to know which dye is associated with which nucleotide as the HMM model represents labeled nucleotides based on their emission frequency.

- If a nucleotide is labeled with the **lowest** frequency dye in the experiment, it is tagged as `1`.
- If a nucleotide is labeled with the **highest** frequency dye, it is tagged as `4`.

You must include an experiment-specific base map (using comma-delimited tags corresponding with [A,C,G,T]) if using anything other than the default `baseMap=1,2,3,4`.

Options - High-Certainty Columns

`-runDecode`

- Specifies whether or not to use the Decode program for post identification.

`-decodeFile=DECODEFILE`

- Specifies the file containing the Decode Probability matrix. EviCons can use two different methods for deciding whether a column can be called without going to a more rigorous algorithm.
- If `runDecode` is `true`, EviCons uses Decode. You need to provide the location of the Decode Probability matrix (`DECODEFILE`), formatted as shown in **Decode Probability Matrix**.
- If `runDecode` is `false`, EviCons uses Plurality, (the default.) Adjustable parameter `postMin` is set at 0.5 by default, indicating that at least half of the bases in a column must agree for it to be called right away.

Decode Probability Matrix

The matrix must be a tab-delimited file in the following format:

```
0.919 0.016 0.024 0.017 0.024
0.105 0.800 0.036 0.026 0.033
0.131 0.030 0.766 0.032 0.041
0.106 0.026 0.033 0.799 0.036
0.130 0.032 0.041 0.030 0.766
```

where the values represent the gap,1, 2, 3, and 4 probabilities of reference (column) vs. observed (row).

Options - Regions of Ambiguity

`-islandLogic=ISLANDLOGIC` (Default value = `steinerMstGA`)

EviCons can use two different algorithms for consensus determination in ambiguous region flanked by high certainty columns.

- If `islandLogic=hmm`, EviCons uses the HMM framework.
- If `islandLogic=steinerMstGA`, EviCons uses the Steiner framework.

Options - Steiner Framework

There are **no** exposed parameters for this framework.

Options - Expected System Errors

`–insPrior=INSPRIOR`

- Prior probability for insertion.

`–delPrior=DELPRIOR`

- Prior probability for deletion.

`–callPrior=CALLPRIOR`

- Prior probability for aligning a base to reference.

EviCons uses the known error rates of the sequencing system for determining the most likely consensus under the HMM model. The algorithm stores a set of default values (`callPrior`, `insPrior`, and `delPrior`) that are expected to reflect the current platform. However, you can specify custom values as platform error rates change.

Options - Alignment Gap Processing

`-preserveAllGaps`

- Specifies whether or not to keep all gaps present in aligned sequences.

For most purposes, the correct behavior is to truncate terminal gaps found in reads within an alignment. However, for certain alignments

you may want to leave all gaps “as-is”. To preserve all gaps, terminal and intervening, set `preserveAllGaps` to `true`.

Options - Computing Resources

`-nProc=NPROC` (Default value = `1`)

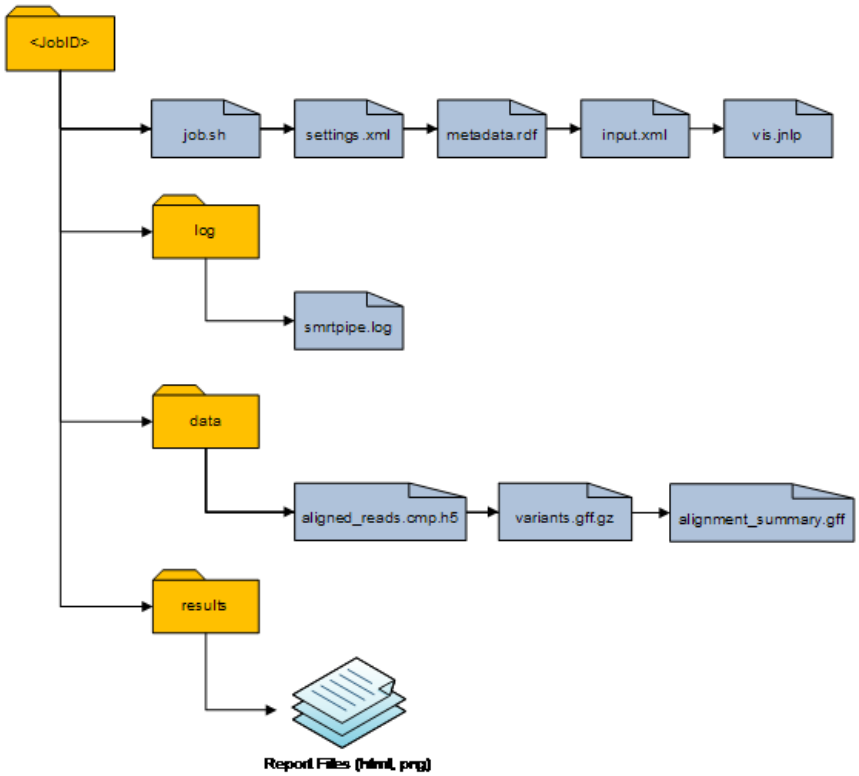
- Specifies the number of processors to use for parallel execution. EviCons is capable of parallel processing if multiple processors are available. If `nProc` exceeds the number of available processors at runtime, EviCons will use 1 processor.

Output Files

EviCons produces two files, `consensus.txt` and `confidence.txt`.

- `consensus.txt` contains the consensus sequence.
- `confidence.txt` lists the confidence value of each consensus call made. Confidence measures range from -1 to 10, with 0 being least confident, and -1 signifying an uncallable confidence value.

SMRT™ Pipe File Structure



Note: The output of a SMRT™ Pipe analysis includes more files than described here; interested users are encouraged to explore the file structure. Following are additional details about the major files.

<jobID>/job.sh

Contains the SMRT™ Pipe command line call for the job.

<jobID>/settings.xml
Contains the XML configuration parameters for the SMRT™ Pipe run.

<jobID>/metadata.rdf
Contains the master index of all the outputs of the SMRT™ Pipe analysis. It is formatted as an RDF-XML file using OWL ontologies. See **http://www.w3.org/standards/semanticweb/** for an introduction to Semantic Web technologies.

<jobID>/input.fofn
Contains the file names of the inputs for this analysis. If the original input XML file came from a fofn file, then these files will be quite similar.

<jobID>/vis.jnlp
Contains wrapping information from `cmp.h5`, `alignment_summary.gff`, and variants.gff to be used by SMRT™ View for data visualization.

log/smrtpipe.log
Contains debugging output from SMRT™ Pipe modules. This is typically shown by way of the **View Log** button in SMRT™ Portal.

Data Files

aligned_reads.cmp.h5, aligned_reads.sam, aligned_reads.bam

- Contains mapping and consensus data from secondary analysis.

alignment_summary.gff

- Contains alignment data summarized on sequence regions.

variants.gff.gz

- Contains all sequence variants called from consensus sequence.

toc.xml
Contains a deprecated version of the master index file for the job outputs. Any future extensions to the master index information will only go into the `metadata.rdf` file.

Results/Reports Files

Modules with **Reports** in their name produce HTML reports with static PNG images using XML+XSLT. These reports can be found in the `results` subdirectory. The underlying XML document for each report is preserved there as well; these can be useful files for data-mining the outputs of SMRT™ Pipe.

The Reference Repository

The **reference repository** is a file-based data store used by SMRT™ Analysis to manage reference sequences and associated information. The full description of all of the attributes of the reference repository is beyond the scope of this document, but you will need to use some basic aspects of the reference repository in most SMRT™ Pipe analyses.

Example: Analysis of multi-contig references can only be handled by supplying a reference entry from a reference repository.

It is very simple to create and use a reference repository:

- A reference repository can be **any** directory on your system. You can have as many reference repositories as you wish; the input to SMRT™ Pipe is a fully resolved path to a reference entry, so this can live in any accessible reference repository.

Starting with the FASTA sequence `genome.fasta`, you upload the sequence to your reference repository using the following command:

```
referenceUploader -c -p/path/to/repository -nGenomeName
-fgenome.fasta
```

where:

- `/path/to/repository` is the path to your reference repository.
- `GenomeName` is the name to use for the reference entry that will be created.
- `genome.fasta` is the FASTA file containing the reference sequence to be uploaded.

For a large genome, we highly recommended that you produce the BLASR suffix array during this upload step. Use the following command:

```
referenceUploader -c -p/path/to/repository -nHumanGenome
-fhuman.fasta --Saw='sawriter -welter'
```

There are many more options for reference management. Consult the MAN page entry for `referenceUploader` by entering `referenceUploader -h`.

To learn more about what is being stored in the reference entries, look at the directory containing a reference entry. You will find a metadata description (`reference.info.xml`) of the reference and its associated files. For example, various static indices for BLASR and SMRT™ View are stored in the sequence directory along with the FASTA sequence.

SMRT™ Pipe Examples

Resequencing Analysis of Bacteriophage Lambda

1. Create a working directory and cd into it:
```
mkdir lambda_test_1
cd lambda_test_1
```
2. Copy the test inputs into the working directory:
```
cp /opt/smrtanalysis/common/test/smrtpipe/
lambda_resequencing/* .
```
3. Source the SMRT™ Analysis environment:
```
. /opt/smrtanalysis/etc/setup.sh
```
4. Create an input XML file:
```
fofnToSmrtpipe.py lambda_resequencing.fofn > input.xml
```
5. Run smrtpipe.py:
```
smrtpipe.py --params=settings.xml xml:input.xml &>
smrtpipe.err
```

If everything is installed and configured correctly, the resequencing results can be found in the data and results subdirectories.

An example is the file `results/quality.xml`, which reports various metrics, including the single-pass accuracy and read lengths of the data. The HTML reports can be viewed in a web browser by pointing at the results directory.

De novo Assembly of Bacteriophage Lambda

1. Create a working directory and cd into it:
```
mkdir lambda_test_2
cd lambda_test_2
```
2. Copy the test inputs into the working directory:
```
cp /opt/smrtanalysis/common/test/smrtpipe/
lambda_denovo/* .
```
3. Source the SMRT™ Analysis environment:
```
. /opt/smrtanalysis/etc/setup.sh
```
4. Create an input XML file:
```
fofnToSmrtpipe.py lambda_assembly.fofn > input.xml
```
5. Run smrtpipe.py:
```
smrtpipe.py --params=settings.xml xml:input.xml &>
smrtpipe.err
```

P/N 001-353-082-01