PACIFIC BIOSCIENCES®

SMRT® Pipe Reference Guide

Introduction

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

SMRT Pipe is the secondary analysis workflow engine of Pacific Biosciences' SMRT Analysis software suite. The underlying framework is written primarily in the Python® programming language. SMRT Pipe is easily extensible, and supports logging, distributed computing, 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 Pacific Biosciences' sequencing data through SMRT Pipe. Power users 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 using the Secondary Analysis Web Services API. For details, see **Secondary Analysis Web Services API**, available at **https://github.com/PacificBiosciences/SMRT-Analysis/wiki/Secondary-Analysis-Web-Services-API-v2.3.0.**

Installation

SMRT Pipe is installed as an integral component of the SMRT Analysis software suite. Currently, there is **no** supported path for installing only SMRT Pipe components. Administrators who intend to install SMRT Pipe Analysis should **not** start the `smrtportald-initd` (Tomcat and MySQL servers) and `kodosd` (automatic secondary analysis) daemons after completing the installation.

- For details, see **SMRT Analysis Software Installation**, available from the Pacific Biosciences web site.
- The latest version of SMRT Pipe is available from the PacBio® Developer's Network at **http://www.pacbiodevnet.com**.

Note:

- Throughout this documentation, the environment variable `SMRT_ROOT` is used to refer to the installation directory for SMRT Analysis. Replace this path with the path appropriate to your installation when using the commands in this document.

Command Line Usage

Setting up the SMRT Analysis Environment

To better deploy SMRT Analysis in an increasingly complex variety of user environments, we implemented new approaches to invoking an isolated and controlled SMRT Analysis environment for running SMRT Portal and SMRT Pipe commands. These changes alleviate some of the problems with resolving library version dependencies and permissions restrictions for non-privileged users.

Prior to SMRT Analysis v2.3.0, SMRT Pipe command-line users were accustomed to issuing the following command to source the environment for accessing SMRT Analysis commands and internals:

```
source $SMRT_ROOT/current/etc/setup.sh
```

Starting with v2.3.0, to avoid conflict with the system user environment, the above command invokes **new behavior**, summarized below:

- System and user-specific locales are **ignored**.
  - The default locale is set to "C" (POSIX). This affects all code run under `$SMRT_ROOT/current/etc/setup.sh`, including SMRT Portal and command-line usage.
- Most user environment variables are **not** passed through. Exceptions to this are either useful to have or needed explicitly by SMRT Analysis internals. The exceptions are:

  ```
  HOME
  LOGNAME
  MPLCONFIGDIR
  PWD
  TERM
  TERMCAP
  USER
  WORKSPACE
  all SMRT_*
  ```

  To pass additional variables through to the SMRT Analysis environment, define the variable `SMRT_ENV_PASSTHROUGH_VARS` prior to sourcing `$SMRT_ROOT/current/etc/setup.sh`:

```
export SMRT_ENV_PASSTHROUGH_VARS="space-separated list of pass-through variables"
source $SMRT_ROOT/current/etc/setup.sh
```

- `PATH` is unset/set to /`usr/bin:/bin`. In addition, the following variables can be used to prepend or append to `PATH`:

  ```
  SMRT_PATH_PREPEND
  SMRT_PATH_APPEND
  ```

So `PATH` will basically be constructed as:

```
$SMRT_PATH_PREPREPEND:$OUR_INTERNAL_PATHS:$PATH:$SMRT_PATH_APPEND
```

Note: We recommend starting a new subshell/child shell when invoking the smrtpipe environment to avoid changing the parent shell.

To illustrate the behavior of source `$SMRT_ROOT/current/etc/setup.sh` within the [parent]/[child] shell context, see the following example:

```
[parent]$ echo $PATH
          /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
[parent]$ bash
  [child ]$ echo $PATH
          /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
  [child ]$ source $SMRT_ROOT/current/etc/setup.sh
  [child ]$ echo $PATH
          /opt/smrtanalysis/current/miscdeps/basesys/usr/bin:/opt/smrtanalysis/current/
redist/mysql/bin:/opt/smrtanalysis/current/redist/mono/bin:/opt/smrtanalysis/current/
analysis/bin:/opt/smrtanalysis/current/redist/scala/bin:/opt/smrtanalysis/current/
redist/java/bin:/opt/smrtanalysis/current/redist/python2.7/bin:/usr/bin:/bin
  [child ]$ exit
          exit
[parent]$ echo $PATH
          /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
```

To simplify accessing the SMRT Analysis environment, users can spawn a child shell, and set up a new isolated environment with a single command, `smrtshell`. This essentially mimics the commands:

```
[parent]$ bash
[child ]$ source $SMRT_ROOT/current/etc/setup.sh
```

Example usage:

```
$ $SMRT_ROOT/smrtcmds/bin/smrtshell
(smrtshell-2.3.0) $ smrtpipe.py --version
smrtpipe.py v1.87.139427
(smrtshell-2.3.0) $
```

SMRT Pipe Command

To invoke the `smrtpipe.py` script, enter:

```
. $SMRT_ROOT/current/etc/setup.sh && smrtpipe.py [--help] [options] --
params=settings.xml xml:input.xml
```

This is the same way `smrtpipe.py` is invoked in SMRT Portal with the `job.sh` script.

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

Following are some of 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 file `$SMRT_ROOT/current/analysis/etc/smrtpipe.rc` before starting an analysis. An example is the `NPROC` variable which controls the number of simultaneous processors to use during the analysis. To restrict SMRT Pipe to 4 processors, use `-D NPROC=4`.
`--debug`	Activates debugging output in the `stderr` and log outputs. To set this flag as a default, specify `DEBUG=True` in the `$SMRT_ROOT/current/analysis/etc/smrtpipe.rc` file.
`--distribute`	Distributes the computation across a compute cluster. For information on configuring SMRT Pipe for a distributed computation environment, see 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 places outputs in this directory, as well as in `data`, `results`, and `log` subdirectories.
`--params=params.xml`	Specifies a settings XML file for running the pipeline analysis. If this option is **not** specified, SMRT Pipe prints a message and then exits.
`--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 is appended to the outputs, until the number of cells reaches `totalCells`.
`--version`	Displays the version number of SMRT Pipe and then exits.
`--kill`	Kills a SMRT Pipe job running in the current directory. This works with `output`.

`--examples`	Displays the SMRT Pipe example jobs. These are a useful reference for how different workflows are configured and run through SMRT Pipe.
	`$ $SMRT_ROOT/current/smrtcmds/bin/smrtshell`
	`(smrtshell-2.3.0) $ smrtpipe.py --examples`
	`Name Directory`
	`1 smrtpipe_resequencing_barcode /opt/smrtanalysis/current/doc/examples/smrtpipe_resequencing_barcode`
	`2 smrtpipe_assembly_hgap3 /opt/smrtanalysis/current/doc/examples/smrtpipe_assembly_hgap3`
	`3 smrtpipe_basemods /opt/smrtanalysis/current/doc/examples/smrtpipe_basemods`
	`4 smrtpipe_resequencing /opt/smrtanalysis/current/doc/examples/smrtpipe_resequencing`
	`Exiting smrtpipe.py.`

Utility Scripts

Several utility scripts to run common workflows can be created:

run_smrtpipe_singlenode.sh

```
SMRT_ROOT=/path/to/smrtanalysis/

. $SMRT_ROOT/current/etc/setup.sh && smrtpipe.py  --
params=settings.xml   xml:input.xml
```

run_smrtpipe_distribute.sh

```
SMRT_ROOT=/path/to/smrtanalysis/

. $SMRT_ROOT/current/etc/setup.sh && smrtpipe.py  --distribute -
-params=settings.xml   xml:input.xml
```

run_smrtpipe_debug.sh

```
SMRT_ROOT=/path/to/smrtanalysis/

. $SMRT_ROOT/current/etc/setup.sh && smrtpipe.py  --debug --
params=settings.xml   xml:input.xml
```

Specifying SMRT® Pipe Inputs

The input file is an XML file specifying the 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 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 `bax.h5` file. For more information, see the document **bas.h5 Reference Guide**, available from the PacBio® Developer's Network at **http://www.pacbiodevnet.com**.

Use the script `fofnToSmrtpipeInput.py` to convert a FOFN (a "file of file names" file) to the input format expected by SMRT Pipe. If `my_inputs.fofn` looks like

```
/mnt/data/2770276/0006/Analysis_Results/
m130512_050747_42209_c100524962550000001823079609281357_s1_p0.2.bax.h5
/mnt/data/2770276/0006/Analysis_Results/
m130512_050747_42209_c100524962550000001823079609281357_s1_p0.3.bax.h5
/mnt/data/2770276/0006/Analysis_Results/
m130512_050747_42209_c100524962550000001823079609281357_s1_p0.1.bax.h5
```

or, for SMRT Pipe versions **before** v2.1.0:

```
/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 for SMRT Pipe v2.1.0:

```
<?xml version="1.0"?>
<pacbioAnalysisInputs>
  <dataReferences>
    <url ref="run:0000000-0000"><location>/mnt/data/2770276/0006/Analysis_Results/
m130512_050747_42209_c100524
962550000001823079609281357_s1_p0.2.bax.h5</location></url>
    <url ref="run:0000000-0001"><location>/mnt/data/2770276/0006/Analysis_Results/
m130512_050747_42209_c100524
962550000001823079609281357_s1_p0.3.bax.h5</location></url>
    <url ref="run:0000000-0002"><location>/mnt/data/2770276/0006/Analysis_Results/
m130512_050747_42209_c100524
962550000001823079609281357_s1_p0.1.bax.h5</location></url>
  </dataReferences>
</pacbioAnalysisInputs>
```

For SMRT Pipe versions **before** v2.1.0:

```
<?xml version="1.0"?>
<pacbioAnalysisInputs>
 <dataReferences>
    <url ref="run:0000000-0000"><location>/share/data/
    /share/data/run_1 m100923_005722_00122_c15301919401091173_s0_p0.bas.h5
    <url ref="run:0000000-0001"><location>/share/data/
    /share/data/run_2/m100820_063008_00118_c04442556811011070_s0_p0.bas.h5
 </dataReferences>
</pacbioAnalysisInputs>
```

To run an analysis using these input files, use the following command:

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

The SMRT Pipe input format lets you specify 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.

Note: To get help for a script, run 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 option specifies an XML file that controls:

- Which 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 setting 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.

SMRT Portal protocol templates are located in:
`$SMRT_ROOT/current/common/protocols/.`

SMRT Pipe modules are located in:
`$SMRT_ROOT/current/analysis/lib/pythonx.x/pbpy-0.1-py2.7.egg/pbpy/smrtpipe/modules/.`

You specify parameters by entering a key-value pair in a `param` element.

- The name of the key is in the name attribute of the `param` element.
- 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: To reference a parameter value in other parameters, use the notation `${variable}` when specifying a value. For example, to reference a global parameter named `home`, use it in other parameters as `${home}`. SMRT Pipe supports arbitrary parameters in the settings XML file, so using 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="P_Filter">
  <param name="minLength">
    <value>50</value>
  </param>
  <param name="readScore">
    <value>0.75</value>
  </param>
 </module>

 <module name="P_FilterReports" />

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

 <module name="P_MappingReports" />
 <module name="P_Consensus" />
 <module name="P_ConsensusReports" />

</smrtpipeSettings>
```

SMRT® Portal Protocols

Following are the secondary analysis protocols included in SMRT Analysis v2.3.0, with the SMRT Pipe module(s) called by each protocol. Many of these modules are described later in this document.

BAM_Resequencing_Beta:

- Used for whole-genome or targeted resequencing.
- Filters reads, maps them to a provided reference sequence, and then calls a consensus sequence.
- This is a version of the `RS_Resequencing` protocol which uses BAM rather than cmp.h5 as the alignment file format. This protocol is **faster** than `RS_Resequencing` for large jobs, but is **not** guaranteed to produce identical results.
  - P_Filter
  - P_Mapping
  - P_BAMMapping
  - P_MappedConsensus

RS_AHA_Scaffolding:

- Used for hybrid assembly of genomes up to 200 Mb in size with PacBio reads.

- Improve existing assemblies up to 200 Mb in size by scaffolding with PacBio long reads to join contigs.
- Filters reads and assembles them with high confidence contigs into scaffolds using a combination of algorithms developed by Pacific Biosciences and the AMOS open-source project.
  - P_Filter
  - P_AHA

RS_BridgeMapper:

- Used for troubleshooting *de novo* assemblies, variants, indels, and so on.
- Returns split alignments of PacBio reads using BLASR.
- Filters reads by length and quality, maps them to a provided reference sequence, and identifies consensus and variants against this reference using Quiver.
  - P_Filter
  - P_Mapping
  - P_GenomicConsensus
  - P_BridgeMapper

RS_HGAP_Assembly.2:

- HGAP (Hierarchical Genome Assembly Process) performs high quality *de novo* assembly using a single PacBio library preparation.
- HGAP consists of pre-assembly, *de novo* assembly with Celera® Assembler, and assembly polishing with Quiver.
- The protocol is optimized for **quality**.
  - P_PreAssembler
  - P_CeleraAssembler
  - P_Mapping
  - P_AssemblyPolishing

RS_HGAP_Assembly.3:

- HGAP (Hierarchical Genome Assembly Process) performs high quality *de novo* assembly using a single PacBio library preparation.
- HGAP consists of pre-assembly, *de novo* assembly with PacBio's AssembleUnitig, and assembly polishing with Quiver.
- The protocol is optimized for **speed**. It introduces a new unitig consensus caller that is substantially faster than the one included with `P_CeleraAssembler`, used in `RS_HGAP_Assembly.2`. This protocol is designed with larger genomes in mind, but can also be used as a replacement for `RS_HGAP_Assembly.2`.
- To see an example on how to setup and run `RS_HGAP_Assembly.3` using `smrtpipe.py`, see the `smrtpipe_assembly_hgap3` example included with `smrtpipe.py`.
  - P_PreAssemblerDagcon
  - P_AssembleUnitig
  - P_Mapping
  - P_AssemblyPolishing

RS_IsoSeq:

- Reads of insert from SMRT Cell cDNA modules are generated, filtered by length and quality, classified into full-length or non-full-length, chimeric or non-chimeric reads, and then mapped against the reference using GMAP to span introns.
- *de novo* consensus isoforms are predicted from classified reads of insert optionally using the ICE (Iterative Clustering and Error Correction) algorithm, and optionally polished via Quiver and classified into High-QV and Low-QV isoforms.
  - P_CCS
  - P_IsoSeqClassify
  - P_IsoSeqCluster

RS_Long_Amplicon_Analysis:

- Used to determine phased consensus sequences for pooled amplicon data.
- Can pool up to 5 distinct amplicons. Reads are clustered into high-level groups, then each group is phased and consensus is called using Quiver.
- Filters chimeric sequences.
- Optionally splits reads by barcode if the sample is barcoded.
  - P_LongAmpliconAnalysis
  - P_Barcode

RS_Minor_Variant:

- Used to call minor variants in a heterogeneous data set against a user-provided reference sequence.
  - P_CCS
  - P_Mapping

RS_Modification_Detection:

- A resequencing analysis that identifies common bacterial base modifications (6-mA, 4-mC, and optionally TET-converted 5-mC).
- Filters reads by length and quality, maps them against a provided reference sequence, and then calls variants.
  - P_Filter
  - P_Mapping
  - P_GenomicConsensus
  - P_ModificationDetection

RS_Modification_and_Motif_Analysis:

- A resequencing analysis that identifies common bacterial base modifications (6-mA, 4-mC, and optionally TET-converted 5-mC), and then analyzes the methyltransferase recognition motifs.
- Filters reads by length and quality, maps them against a provided reference sequence, and then calls variants.
  - P_Filter
  - P_Mapping
  - P_GenomicConsensus
  - P_ModificationDetection
  - P_MotifFinder

RS_PreAssembler:

- Used to build a set of highly accurate long reads for use in *de novo* assembly, using the hierarchical genome assembly process (HGAP).
- Takes each read exceeding a minimum length, aligns all reads against it, trims the edges, and then takes the consensus.
  - PreAssemblerSFilter
  - P_PreAssembler

RS_ReadsOfInsert:

- Used to process reads from the insert sequence of single molecules and estimate the length of the insert sequence loaded onto a SMRT Cell.
- Generates reads from the insert sequence of single molecules, optionally splitting by barcode.
- Replaces the Circular Consensus Sequencing (CCS) protocol, which has been moved off the primary analysis software on the instrument as of v2.1.0.
- To obtain the closest approximation of CCS as it existed on-instrument, specify `MinCompletePasses = 2` and `MinPredictedAccuracy = 0.9`.
  - P_CCS
  - P_Barcode

RS_ReadsOfInsert_Mapping:

- Used for whole-genome or targeted resequencing.
- Filters reads, maps them to a provided reference sequence, and then identifies consensus and variants against this reference.
- Haploid variants and small indels, but **not** diploid variants, are called during consensus.
- Uses Reads of Insert (formerly known as CCS) data during mapping.
  - P_Filter
  - P_CCS
  - BLASR_De_Novo_CCS

RS_Resequencing:

- Used for whole-genome or targeted resequencing.
- Filters reads, maps them to a provided reference sequence, and then calls a consensus sequence.
  - P_Filter
  - P_Mapping
  - P_GenomicConsensus

RS_Resequencing_Barcode:

- Used for whole-genome or targeted resequencing for **barcoded** samples.
- Filters reads, maps them to a provided reference sequence, and then calls a consensus sequence.
  - P_Filter
  - P_Mapping

  - P_Barcode

RS_Site_Acceptance_Test:

- Site acceptance test workflow for lambda resequencing.
- Generates a report displaying site acceptance test metrics.
  - P_Filter
  - P_Mapping
  - P_GenomicConsensus

RS_Subreads:

- Filters reads based on the minimum read length and read quality specified.
  - P_Filter

SMRT® Pipe Modules and Their 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 should 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.

Global Parameters

Global parameters are potentially used in multiple modules. In the SMRT Pipe internals, they are accessed in the "global" namespace.

Following are some 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**)

P_AHA (AHA Scaffolding) Module

This module scaffolds high-confidence contigs, such as those from Illumina® data, using Pacific Biosciences' long reads.

Input:

`P_AHA.py` uses two kinds of input instead of one:

- A FASTA file of high-confidence sequences to be scaffolded. These are typically contigs assembled from Illumina® short-read sequence data. They are passed to AHA as a reference sequence in the `settings.xml` input file.
- Pacific Biosciences' long reads, in HDF5 format. These are used to join the high-confidence contigs into a scaffold. Note that versions of the AHA Scaffolding algorithm prior to v2.1.0 accepted reads in FASTA format. After v2.1.0, users with FASTA formatted reads should use the underlying executable `pbaha.py`.

Sample `settings.xml` file for **long** reads, with only customer-facing parameters:

```
<?xml version="1.0"?>
<smrtpipeSettings>
  <global>
    <param name="reference">
        <value>/mnt/secondary-siv/references/ecoli_contig</value>
    </param>
  </global>
  <module name="P_Fetch"/>
  <module name="P_Filter">
    <param name="minLength">
        <value>50</value>
    </param>
    <param name="minSubReadLength">
        <value>50</value>
    </param>
    <param name="readScore">
        <value>0.75</value>
    </param>
  </module>
  <module name="P_FilterReports"/>
  <module name="P_AHA">
    <param name="fillin">
        <value>False</value>
    </param>
    <param name="blasrOpts">
        <value>-minMatch 10 -minPctIdentity 70 -bestn 10 -noSplitSubreads</value>
    </param>
    <param name="instrumentModel">
        <value>RS</value>
    </param>
    <param name="paramSchedule">
        <value>6,3,75,100;6,3,75,100;5,3,75,100;6,2,75,100;6,2,75,100;5,2,75,100
        </value>
    </param>
    <param name="maxIterations">
        <value>6</value>
    </param>
    <param name="description">
        <value>AHA ("A Hybrid Assembler") is the PacBio hybrid assembly algorithm.
        It is based on the open source assembly software package AMOS, with
        additional software components tailored to PacBio's long reads and error
        profile.</value>
    </param>
  </module>
</smrtpipeSettings>
```

Output:

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

Parameters:

Parameter	Default Value	Description
`paramSchedule`	`None`	Specifies parameter schedules used for iterative hybrid assembly. Schedules are in comma-delimited tuples, separated by semicolons. **Example**: `6,3,75;6,3,75;6,2,75;6,2,75`. The fields, in order, are: • Minimum alignment 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.
`fillin`	`False`	Specifies whether to use long reads.
`blasrOpts`		Options passed directly to BLASR for aligning reads to contigs. Default value: `-minMatch 10 -minPctIdentity 60 -bestn 10 -noSplitSubreads`
`maxIterations`	`6`	Specifies the maximum number of iterations to use from `paramSchedule`. If `paramSchedule` is **larger** than `maxIterations`, it is truncated at `maxIterations`. If `paramSchedule` is **smaller** than `maxIterations`, the last iteration of `paramSchedule` is repeated.
`cleanup`	`True`	Specifies whether to clean up intermediate files. (These files can be useful for debugging purposes.)
`runNucmer`	`True`	Specifies whether to use `Nucmer` to detect repeat locations. This can improve assemblies, but can be very slow on large highly repetitive genomes.
`gapFillOpts`	`""`	Options to be passed directly to `gapFiller.py`.
`noScaffoldImages`	`True`	Specifies **not** producing SVG files of the scaffolds. Creating these files can be expensive for large assemblies, but is recommended for small assemblies.

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

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

Known Issues

- 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>
  ```

P_AnalysisHook Module

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

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 called with the command line `exeCmd jobDir`, where `jobDir` is the root of the SMRT Pipe output for this analysis. (**Optional**)

P_AssemblyPolishing Module

This module is used in HGAP to polish draft assemblies using Quiver.

Input:

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

Output:

- `data/polished_assembly.fasta.gz`: The consensus sequence in FASTA format.
- `data/polished_assembly.fastq.gz`: The consensus sequence in FASTQ format.
- `results/polished_report.html`: HTML-formatted report for the polished assembly.
- `results/polished_report.xml`: XML-formatted report for the polished assembly.

Parameters:

Parameter	Default Value	Description
`enableMapQVFilter`	`True`	<??>

P_AssembleUnitig Module

This module is new to HGAP.3. It calls `P_CeleraAssembler configure` to assemble corrected reads into unitigs, truncating the traditional `P_CeleraAssembler` workflow after the unitigger stage. This avoids the time-consuming unitig consensus, CA/utgcns, stage built into `P_CeleraAssembler` in favor of our own, much faster, unitig consensus caller `PB/utgcns` **(https://github.com/pbjd/pbutgcns)**.

Input:

- `corrected.fastq`: FASTQ file of corrected long seed reads generated by `pbdagcon` during the pre-assembler stage.

Output:

- `draft_consensus.fasta`: A decent first cut of the assembly (typically ~QV30). Contains both contigs and degenerates.

Parameters:

Parameter	Default Value	Description
`Genome Size`	`None`	Specifies the approximate size of the sample genome.
`Target Coverage`	`None`	Specifies how much coverage to allow into the assembly.

P_BAMMapping Module

This experimental module aligns reads against a reference sequence, possibly a multi-contig reference. If the `P_Filter` module is run first, then **only** the reads which passed filtering are aligned.

Output:

- `data/aligned_reads_by_contig.chunk*.bam`: One or more BAM files that contain alignments of the input reads. Each file contain alignments to a subset of contigs of the reference.
- `data/aligned_reads_by_contig.chunk*.bam.bti`: Bamtools BTI index files for each alignment BAM.
- `data/alignment_summary.gff`: Summary information.

Parameters:

Parameter	Default Value	Description
`maxHits`	`10`	Finds sub-optimal alignments and reports up to this many hits per read. **(Optional)**
`minAnchorSize`	`12`	Ignores anchors **smaller** than this size when finding candidate hits for dynamic programming alignment. **(Optional)**

Parameter	Default Value	Description
maxDivergence	30	Specifies maximum divergence between read and reference to allow a mapping. Divergence = (1 - accuracy).
blasr_opts	None	Specifies options passed to BLASR.
concordant	FALSE	Map all subreads from one ZMW to the same genomic location.

P_Barcode Module

This module provides access to the `pbbarcode` command-line tools, which you use to identify barcodes in PacBio reads.

Input:

- Complete barcode FASTA file: A standard FASTA file with barcodes less than 48 bp in length. Based on the score mode you specify, the barcode file might need to contain an even number of barcodes.

  Example:

  ```
  <param name="barcode.fasta">
     <value>/mnt/secondary/Smrtpipe/martin/prod/data/
     workflows/barcode_complete.fasta</value>
  </param>
  ```

- Barcode scoring method: It relates directly to the particular sample preparation protocol used to construct the molecules. Depending on the scoring mode, the barcodes are grouped together in different ways. Valid options are:
  - `symmetric`: Supports barcode designs with two **identical** barcodes on both sides of a SMRTbell™ template. Example: For barcodes (**A, B**), molecules are labeled as **A--A** or **B--B**.
  - `paired`: Supports barcode designs with two distinct barcodes on each side of the molecule, with neither barcode appearing without its mate. Minimum example: (ALeft, ARight, BLeft, BRight), where the following barcode sets are checked: **ALeft**--**ARight**, **BLeft--BRight**.

  Example:

  ```
  <param name="mode">
     <value>symmetric</value>
  </param>
  ```

- Pad arguments: Defines how many bases to include from the adapter, and how many bases to include from the insert. Ideally, this is `0` and `0`. This produces shorter alignments; however, if the adapter-calling algorithm slips a little one might lose a little sensitivity and/or specificity because of this. Do **not** specify pad argument unless you have a compelling use case.

  Examples:

```
<param name="adapterSidePad">
  <value>2</value>
</param>

<param name="insertSidePad">
  <value>2</value>
</param>
```

Output:

- `/data/*.bc.h5`: Barcode calls and their scores for each ZMW.
- `/data/barcode.fofn`: Contains a list of files.
- Other files are output based on the protocol used to call the `P_Barcode` module. **Example**: `/data/aligned_reads.cmp.h5`, returned by the `RS_Resequencing_Barcode` protocol.

P_BridgeMapper Module

This module creates split alignments of Pacific Biosciences' reads for viewing with SMRT View. The split alignments can be used to infer the presence of assembly errors or structural variation. `P_BridgeMapper` works by first using BLASR to get primary alignments for filtered subreads. Then, `P_BridgeMapper` calls BLASR again, mapping any portions of those subreads not contained in the primary alignments.

Input:

- `input.fofn`: A file containing the file names of the raw input files used for the analysis.
- `data/aligned_reads.cmp.h5`: The initial alignments for each subread.

Ouput:

- `data/split_reads.bridgemapper.gz`: A gzipped, tab-separated file of split alignments. This file is consumed by SMRT View. **Note**: The meanings of some of the columns in this file have changed:
  - The columns for BLASR scores now contain placeholder values.
  - The columns for the starts and ends of alignments now follow the convention used in cmp.h5 files: Start is **always** less than end, regardless of the orientation of the alignment.

Parameters:

Parameter	Default Value	Description
`minRootLength`	`250`	Only consider subreads with primary alignments **longer** than this threshold.
`minAffixLength`	`50`	Only report split alignments with secondary alignments **longer** than this threshold.

P_CCS (Reads of Insert) Module

This module computes Read of Insert/CCS sequences from single-molecule reads. It is used to estimate the length of the insert sequence loaded onto a SMRT Cell. Reads of Insert **replaces** the Circular Consensus Sequencing (CCS) protocol, which has been moved off the primary analysis instrument as of v2.1.0.

Input:

- `bas.h5` files

Output:

- `data/<movie_name>.fasta`: A FASTA file containing the consensus sequences of each molecule passing quality filtering.
- `data/<movie_name>.fastq`: A FASTQ file containing the consensus sequences and base quality of each molecule passing quality filtering.
- `data/<movie_name>.ccs.h5`: A ccs.h5 (similar to a bas.h5) file containing a representation of the CCS sequences and quality values.

Parameters:

Parameter	Default Value	Description
`minFullPasses`	`2`	The raw sequence must make **at least** this number of passes over the insert sequence to emit a CCS read for this ZMW.
`minPredictedAccuracy`	`0.9`	The **minimum** allowed value of the predicted consensus accuracy to emit a CCS read for this ZMW.
`minLength`	`None`	The **minimum** length of CCS reads in bases. **(Optional)**
`maxLength`	`None`	The **maximum** length of CCS reads in bases. **(Optional)**

- **Note**: Use the default values to obtain the closest approximation of CCS as it existed on-instrument.

P_Fetch Module

This module fetches the input data and generates a file of the file names of the `input.pls` files for downstream analysis. This module has **no** exposed parameters.

Output:

- `pls.fofn` (File of file names of the input `.pls` files)

P_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.

Input:

- `bas.h5` files (**pre v2.1.0**) or bax.h5 files (**post v2.1.0**)

Output:

- `data/filtering_summary.csv`: Includes raw metrics and filtering information for each read (**not** subread) found in the original `bas.h5` files.
- `rgn.h5` (one for each input `bas.h5` file): Filtering information generated by the module.

Parameters:

Parameter	Default Value	Description
`minLength`	`None`	Reads with a high quality region read length **below** this threshold are filtered out. (**Optional**)
`maxLength`	`None`	Reads with a high quality region read length **above** this threshold are filtered out. (**Optional**)
`minSubReadLength`	`None`	Subreads **shorter** than this length are filtered out.
`maxSubReadLength`	`None`	Subreads **longer** than this length are filtered out.
`minSNR`	`None`	Reads with signal-to-noise ratio **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 to trim reads to the high-quality region. (**Optional**)
`artifact`	`None`	Reads with a read artifact score **less than** this (negative) number are filtered out. No number indicates no artifact filtering. Reasonable thresholds are typically between `-1000` and `-200`. (**Optional**)

P_Genomic Consensus (Quiver) Module

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

Input:

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

Output:

- `data/aligned_reads.cmp.h5`
- `data/variants.gff.gz`: A gzipped GFF3 file containing variants versus the reference.
- `data/consensus.fastq.gz`: The consensus sequence in FASTQ format.
- `data/alignment_summary.gff`, `data/variants.vc`: Useful information about variants.

Parameters:

Parameter	Default Value	Description
`makeBed`	`True`	Specifies whether to output a BED representation of the variants. (**Optional**)
`makeVcf`	`True`	Specifies whether to output a VCF representation of the variants. (**Optional**)

P_IsoSeqClassify Module

This module is used for Iso-Seq™ cDNA analysis, including cDNA reads quality control and mapping to a provided reference.

1. The module generates reads of insert from SMRT Cell cDNA molecules, removes cDNA primers and polyA sequences from reads, and then classifies reads of insert into full-length or non-full-length, chimeric or non-chimeric reads.
2. The module then maps classified reads and predicted consensus isoforms to a provided reference sequence.

Input:

- `input.fofn`: A file containing the file names of PacBio movies produced by the `P_Fetch` module.
- `reads_of_insert.fasta`: A FASTA file containing reads of insert produced by the `P_CCS` module.
- `reads_of_insert.fofn`: A file containing the file names of reads of insert ccs.h5 files produced by the `P_CCS` module.

Output:

- `isoseq_draft.fasta`: A FASTA file containing all classified reads of insert.

- `isoseq_flnc.fasta`: A FASTA file containing full-length non-chimeric reads generated by `pbtranscript.py classify` in the `P_IsoSeqclassify` module.
- `isoseq_nfl.fasta`: A FASTA file containing non-full-length reads generated by `pbtranscript.py classify` in the `P_IsoSeqClassify` module.
- `Isoseq_primer_info.csv`: A CSV file containing classified reads of insert, including:
  - The read ID and strand
  - Whether 3' primer, 5’ primer, and/or polyA tail are seen
  - The position of any 3' primer, 5’ primer, and/or polyA tail
  - ID of any primer seen in this read, and whether this read is chimeric or not.
- `classify_summary.txt`: A text file summarizing `P_IsoSeqClassify` results.

Parameters:

Parameter	Default Value	Description
`minSeqLen`	`300`	Specifies the **minimum** length of reads of insert to analyze.
`customizedPrimerFa`	`Empty String`	A FASTA file containing customized primers, which is used to detect primers in reads of insert. **(Optional)**
`ignorePolyA`	`False`	Specifies whether or not full-length reads of insert require polyA tails.
`gmap_n`	`0`	The **maximum** number of paths to show per isoform; that is, the GMAP `--npaths` option. If set to `0`, GMAP outputs **two** paths if chimeras are detected; **one** path if chimeras are **not** detected.

P_IsoSeqCluster Module

This module is used for Iso-Seq™ cDNA analysis, including *de novo* consensus isoforms prediction and polishing.

1. The module **optionally** predicts *de novo* consensus isoforms from classified reads of insert using the ICE (Iterative Clustering and Error Correction) algorithm.
2. The module **optionally** polishes predicted consensus isoforms using Quiver and classifies the polished isoforms into high-QV or low-QV isoforms based on user-specified criteria.

Input:

- `input.fofn`: A file containing the file names of PacBio movies produced by the `P_Fetch` module.
- `reads_of_insert.fofn`: A file containing the file names of reads of insert ccs.h5 files produced by the `P_CCS` module.

- `isoseq_flnc.fasta`: A FASTA file containing full-length non-chimeric reads generated by `pbtranscript.py classify` in the `P_IsoSeqClassify` module.
- `isoseq_nfl.fasta`: A FASTA file containing non-full-length reads generated by `pbtranscript.py classify` in the `P_IsoSeqClassify` module.

Output:

- `consensus_isoforms.fasta`: A FASTA file containing predicted consensus isoforms generated by the `P_IsoSeqCluster` module. These isoforms are **not** Quiver-polished.
- `polished_high_qv_consensus_isoforms.fasta|q`: A FASTA/FASTQ file containing polished high-QV consensus isoforms generated by the `P_IsoSeqCLuster` module. Produced **only** if the **Call Quiver to polish consensus isoforms** option is specified.
- `polished_low_qv_consensus_isoforms.fasta|q`: A FASTA/FASTQ file containing polished low-QV consensus isoforms generated by the `P_IsoSeqCluster` module. Produced **only** if the **Call Quiver to polish consensus isoforms** option is specified.
- `isoseq_cluster_info.csv`: A CSV file containing information on predicted isoforms. Each line includes: cluster ID, ID of a read which support this cluster, and whether this supportive read is full-length or non-full-length.
- `cluster_summary.txt`: A text file summarizing `P_IsoSeqCluster` results.
- `aligned_consensus_isoforms.sam|bam|cmp.h5`: A SAM|BAM|cmp.h5 file containing alignments of predicted consensus isoforms, and the reference sequence.

Parameters:

Parameter	Default Value	Description
`cluster`	`False`	Specifies whether or not to predict *de novo* consensus isoforms using the ICE (Iterative Clustering and Error correction) algorithm.
`cDNASize`	`under1k`	Specifies the estimated cDNA size. Values are {`"under1k"|"between1k2k"|"between2k3k"|"above3k"`}
`quiver`	`False`	Specifies whether or not to call Quiver to polish consensus isoforms.
`hq_quiver_min_accuracy`	`0.99`	Specifies the **minimum** Quiver accuracy needed to classify an polished isoform as high-QV.

P_LongAmpliconAnalysis Module

This module finds *de novo* phased consensus sequences from a pooled set of (possibly diploid) amplicons.

Input:

- `bas.h5` files

Output:

- `data/amplicon_analysis.fasta/q`: A FASTA/FASTQ file containing the high-quality, non-chimeric sequences found.
- `data/amplicon_analysis_chimeras_noise.fasta/q`: A FASTA/FASTQ file containing the low-quality, chimeric sequences found.
- `data/amplicon_analysis_summary.csv`: A .csv file containing summary information about each read.
- `data/amplicon_analysis.csv`: A .csv file containing coverage and QV information at the per-base level.

Parameters:

Parameter	Default Value	Description
`minLength`	`1000`	Use **only** subreads longer than this threshold. Should be set to ~75% of the shortest amplicon length.
`minReadScore`	`0.78`	Use **only** reads with a Read Score **higher** than this value.
`maxReads`	`2000`	Use **at most** this number of reads to find results. Values greater than `10,000` may cause long run times.

P_MappedConsensus Module

This experimental module calls a consensus sequence using alignments. It can use either BAM or cmp.h5 alignment file formats.

Output:

- `consensus.fasta`: The consensus sequence.

P_Mapping (BLASR) Module

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

Output:

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

Parameters:

Parameter	Default Value	Description
maxHits	10	FInds sub-optimal alignments and report up to this many hits per read. (**Optional**)
minAnchorSize	12	Ignores anchors **smaller** than this size when finding candidate hits for dynamic programming alignment. (**Optional**)
maxDivergence	30	Specifies maximum divergence between read and reference to allow a mapping. Divergence = (1 - accuracy).
placeRepeatsRandomly	True	Specifies that if BLASR maps a read to more than one location with equal probability, then it **randomly** selects which location it chooses as the best location. If set to False, defaults to the **first** on the list of matches.
pbalign_opts		Specifies default options passed to the underlying pbalign script. Default value = --seed=1 --minAccuracy=0.75 --minLength=50 --concordant --algorithmOptions="-useQuality"
pbalign_advanced_opts	Empty String	Passes advanced options to the underlying pbalign.py script. (**Optional**) **Note**: This option is now exposed in SMRT Portal to give advanced users more freedom to pass non-standard parameters to the underlying pbalign script. However, use this option **with care**.
--useccs	None	A parameter sent to the underlying pbalign.py script via the pbalign_opts or pbalign_advanced_opts parameters. **(Optional)** Values are: • useccsdenovo: Maps just the *de novo* called sequence and reports. (Does **not** include quality values.) • useccs: Maps the *de novo* called sequence, then aligns full passes to the sequence that the *de novo* called sequence aligns to. • useccsall: 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.
sambam	False	Specifies whether to output a BAM representation of the cmp.h5 file. (**Optional**)
gff2Bed	False	Specifies whether to output a BED representation of the depth of coverage summary. (**Optional**)

P_Modification Detection Module

This module uses the cmp.h5 output by the P_Mapping module to:

1. **Compare** observed IPDs in the cmp.h5 file at each reference position on each strand with control IPDs. Control IPDs are supplied by either an in-silico computational model, or observed IPDs from unmodified “control” DNA.
2. **Generate** modifications.csv and modifications.gff reporting statistics on the IPD comparison.

Predicted Kinetic Background Control vs Case-Control Analysis

By default, the control IPDs are generated per-base of the reference with an in-silico model of the expected IPD values for each position, based on sequence context. The computational model is called the **Predicted IPD Background Contro**l. Even in normal unmodified DNA, the IPD at any particular point will vary. Internal studies at Pacific Biosciences show that most of the variation in mean IPD across a genome can be predicted from a 12-base sequence context surrounding the active site of the DNA polymerase. The bounds of the relevant context window correspond to the window of DNA in contact with the polymerase, as seen in DNA/polymerase crystal structures. The module includes a pre-trained lookup table mapping 12-mer DNA sequences to mean IPDs observed in PacBio® data.

Filtering and Trimming

Some PacBio data features require special attention for good modification detection performance. The module inspects the alignment between the observed bases and the reference sequence. For an IPD measurement to be included in the analysis, the read sequence **must** match the reference sequence for `K` around the cognate base; currently, `K = 1`. The IPD distribution at some locus can be seen as a mixture of the “normal” incorporation process IPD (sensitive to the local sequence context and DNA modifications) and a contaminating “pause” process IPD which has a much longer duration (the mean is >10x longer than normal), but rarely happens (~1% of IPDs).

Pauses are defined as pulses with an IPD >10x longer than the mean IPD at that context. Heuristics are used to filter out the pauses.

Statistical Testing

The module tests the hypothesis that IPDs observed at a particular locus in the sample have longer means than IPDs observed at the same locus in **unmodified** DNA. If a Whole-Genome-Amplified data set is generated, which removes DNA modifications, the module uses a case-control, two-sample t-test.

The module also provides a pre-calibrated **Predicted Kinetic Background Control** model which predicts the unmodified IPD, given a 12-base sequence context. In that case, the module uses a one-sample t-test, with an adjustment to account for error in the control model.

Input:

- `aligned_reads.cmp.h5`: A standard cmp.h5 file with alignments and IPD information that supplies the kinetic data for modification detection.
- `Reference Sequence`: The path to a SMRT Portal reference repository entry for the reference sequence used to perform alignments.

Output:

- `modifications.csv`: Contains one row for each (reference position, strand) pair that appeared in the data set with coverage of at least `x`. (`x` defaults to `3`, but is configurable using the `ipdSummary.py –minCoverage` option.) The reference position index is 1-based for compatibility with the GFF file in the R environment.
- `modifications.gff`: Each template position/strand pair whose p-value exceeds the p-value threshold displays as a row. (The default threshold is `p=0.01` or `score=20`.) The file is compliant with the GFF version 3 specification, and the template position is 1-based, per the GFF specification. The strand column refers to the strand carrying the detected modification, which is the **opposite** strand from those used to detect the modification.

  The auxiliary data column of the GFF file contains other statistics useful for downstream analysis or filtering. This includes the coverage level of the reads used to make the call, and +/- 20 bp sequence context surrounding the site.

Results are generally indexed by reference position and reference strand. In all cases, the strand value refers to the strand carrying the modification in the DNA sample. The kinetic effect of the modification is observed in read sequences aligning to the opposite strand, so reads aligning to the positive strand carry information about modification on the negative strand and vice versa. The module **always** reports the strand containing the putative modification.

Parameters:

Parameter	Default Value	Description
`identifyModifications`	`False`	Specifies whether to use a multi-site model to identify the modification type.
`tetTreated`	`False`	Specifies whether the sample was TET-treated to amplify the signal of m5C modifications.

P_MotifFinder (Motif Analysis) Module

This module finds sequence motifs containing base modifications. The primary application is finding restriction-modification systems in prokaryotic genomes. `P_MotifFinder` analyzes the output of the `P_ModificationDetection` module.

Input:

- `modifications.csv`: Contains one row for each (reference position, strand) pair that appeared in the data set with coverage of at least `x`.
- `modifications.gff`: Each template position/strand pair whose p-value exceeds the p-value threshold displays as a row.

Output:

- `data/motif_summary.csv`: A summary of the detected motifs, as well as the evidence for motifs.
- `data/motifs.gff`: A reprocessed version of `modifications.gff` (from `P_ModificationDetection`) containing motif annotations.

Parameters:

Parameter	Default Value	Description
`minScore`	`35`	Only consider detected modifications with a Modification QV **above** this threshold.

P_PreAssemblerDagcon Module

This module provides the primary difference in `RS_HGAP_Assembly.3`. `P_PreAssemblerDagcon` was designed as a drop-in replacement for the correction step in `RS_HGAP_Assembly.2`, providing the same functionality much faster and more efficiently than the `P_PreAssembler` module. It includes a simple, alignment-based chimera filter that reduces effects caused by missing SMRTbell™ adapters, such as spurious contigs in assemblies.

Note that the quality values in the FASTQ file for the corrected reads are uniformly set to `QV24`. This is determined by mapping corrected reads to a known reference and appears to work well on a broad set of data. We are considering deriving QV values directly from the data.

Input:

- Filtered subreads FASTA file, generated by `P_Filter`.
- `params.xml`
- `input.xml`

The module is a much simpler design and can **only** be run using SMRT Pipe in combination with the filtered subreads module. Auto-seed cutoff still targets 30X seed reads.

Parameters:

Parameter	Default Value	Description
`targetChunks`	`None`	Specifies how many chunks to split the seed reads (target) into. In the example on the next page, the value is set to `6`, which generates approximately 5X (30X/6) worth of sequence per split file, or chunk. If set to `1`, then set `splitBestn` to the same value as `totalBestn`.
`splitBestn`	`None`	Must be adjusted based on `targetChunks`. This is roughly 1.5 to 2 times the coverage found in a given split file, though this may produce false positives in some cases. This could affect correction, so be careful
`totalBestn`	`24`	Based on the total coverage of 30X. This default is sensible in most cases.

Sample `input.xml`, bas.h5-only input mode

- **Note**: bas.h5 input files **must** have the suffix `bas.h5`.

```
<pacbioAnalysisInputs>
<?xml version="1.0"?>
<pacbioAnalysisInputs>
    <dataReferences>
        <url ref="run:0000000-0001">
    <location>
        /path/to/input.bas.h5
    </location>
</url>
    </dataReferences>
</pacbioAnalysisInputs>
```

Sample `params.xml`, bas.h5-only input mode

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<smrtpipeSettings>
    <module id="P_Filter" >
        <param name="minLength"><value>100</value></param>
        <param name="minSubReadLength"><value>500</value></param>
        <param name="readScore"><value>0.80</value></param>
    </module>
    <module id="P_PreAssemblerDagcon">
        <param name="computeLengthCutoff"><value>true</value></param>
        <param name="minLongReadLength"><value>6000</value></param>
        <param name="targetChunks"><value>6</value></param>
        <param name="splitBestn"><value>11</value></param>
        <param name="totalBestn"><value>24</value></param>
        <param name="blasrOpts"><value> -noSplitSubreads -minReadLength 200
         -maxScore -1000 -maxLCPLength 16 </value></param>
    </module>
</smrtpipeSettings>
```

Output:

- `data/corrected.fasta, data/corrected fastq`: FASTQ and FASTA file of corrected long reads.

- `preassembler_report.json`: JSON-formated pre-assembly report.
- `preassembler_report.html`: HTML-formated pre-assembly report.

SMRT® Pipe Tools

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

- **All** the tools are located at `$SEYMOUR_HOME/analysis/bin`.
- Use the `--help` option to see usage information for each tool. (Some tools are undocumented.)

Building the SMRT Pipe tools manually, without SMRT Portal, SMRT View, or Kodos

- It is currently **not** possible to build the SMRT Pipe tools without SMRT Portal, SMRT View, or Kodos.

SMRT® Pipe File Structure

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

`<jobID>/job.sh`

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

`<jobID>/settings.xml`

- Contains the modules (and their associated parameters) to be run as part of the SMRT Pipe run.

`<jobID>/metadata.rdf`

- Contains all important metadata associated with the job. This includes metadata propagated from primary results, links to all reports and data files exposed to users, and high-level summary metrics computed during the job. The file is an entry point to the job by tools such as SMRT Portal and SMRT View. `metadata.rdf` 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`

- This file (“file of file names”) is generated early during a job and contains the file names of the raw input files used for the analysis.

`<jobID>/input.xml`

- Used to specify the input files to be analyzed in a job, and is passed on to the command line.

`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

The `Data` directory is where most raw files generated by the pipeline are stored. (**Note**: The following are example output files - for more details about specific files, see the sections dealing with individual modules.)

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

- Mapping and consensus data from secondary analysis.

`alignment_summary.gff`

- Alignment data summarized on sequence regions.

`variants.gff.gz`

- All sequence variants called from consensus sequence.

`toc.xml`

- **Deprecated** -The master index information for the job outputs is now included in 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 are located 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 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 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 upload.

Notes on FASTA files to be used in the reference repository:

- The FASTA header should **not** be processed in any way when imported into the reference repository.
- The FASTA header **cannot** contain any tab characters, colons, double quotes, or additional `>` characters beyond the standard demarcation of the start of the header.

Within a multi-sequence FASTA file, the header must be **unique**.

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.

Understanding the SMRT Pipe Workflow

One way to understand how the various scripts and commands interact within a SMRT Pipe protocol is to run an analysis in SMRT Portal, then look inside the workflow directory for your completed job. Here is an example from HGAP.3:

```
workflow/
├── P_AssembleUnitig
│   ├── genFrgFile.sh
│   ├── getUnitigs.sh
│   ├── runCaToUnitig.sh
│   ├── unitigConsensus_001of005.sh
│   ├── unitigConsensus_002of005.sh
│   ├── unitigConsensus_003of005.sh
│   ├── unitigConsensus_004of005.sh
│   ├── unitigConsensus_005of005.sh
│   ├── unitigConsensus.unitigs.Scatter.sh
│   ├── unitigConsensus.utgConsensus.Gather.sh
│   └── writeRunCASpec.sh
├── P_AssemblyPolishing
│   ├── callConsensus.sh
│   ├── enrichAlnSummary.sh
│   ├── polishedJsonReport.sh
│   ├── topCorrectionsJsonReport.sh
│   ├── variantsJsonReport.sh
│   └── zipPolishedFasta.sh
├── P_Fetch
│   ├── adapterRpt.sh
│   ├── overviewRpt.sh
│   └── toFofn.sh
├── P_Filter
│   ├── filter_001of003.sh
│   ├── filter_002of003.sh
│   ├── filter_003of003.sh
│   ├── filter.rgnFofn.Gather.sh
│   ├── filter.summary.Gather.sh
│   ├── subreads_001of003.sh
│   ├── subreads_002of003.sh
│   ├── subreads_003of003.sh
│   ├── subreads.subreadFastq.Gather.sh
│   ├── subreads.subreads.Gather.sh
│   └── subreadSummary.sh
├── P_FilterReports
│   ├── loadingRpt.sh
│   ├── statsRpt.sh
│   └── subreadRpt.sh
├── P_Mapping
│   ├── align_001of003.sh
│   ├── align_002of003.sh
│   ├── align_003of003.sh
│   ├── align.cmpH5.Gather.sh
│   ├── align.plsFofn.Scatter.sh
│   ├── covGFF.sh
│   ├── gff2Bed.sh
│   ├── repack.sh
│   ├── samBam.sh
│   ├── sort.sh
│   └── unmapped.sh
├── P_MappingReports
│   ├── coverageJsonReport.sh
```

```
│   └── statsJsonReport.sh
├── P_PreAssemblerDagcon
│   ├── filterLongReadsByLength.sh
│   ├── hgapAlignForCorrection_001of006.sh
│   ├── hgapAlignForCorrection_002of006.sh
│   ├── hgapAlignForCorrection_003of006.sh
│   ├── hgapAlignForCorrection_004of006.sh
│   ├── hgapAlignForCorrection_005of006.sh
│   ├── hgapAlignForCorrection_006of006.sh
│   ├── hgapAlignForCorrection.blasrM4Fofn.Gather.sh
│   ├── hgapAlignForCorrection.blasrM4.Gather.sh
│   ├── hgapAlignForCorrection.target.Scatter.sh
│   ├── hgapCorrection_001of006.sh
│   ├── hgapCorrection_002of006.sh
│   ├── hgapCorrection_003of006.sh
│   ├── hgapCorrection_004of006.sh
│   ├── hgapCorrection_005of006.sh
│   ├── hgapCorrection_006of006.sh
│   ├── hgapCorrection.fasta.Gather.sh
│   ├── hgapCorrection.fastq.Gather.sh
│   ├── hgapFilterM4_001of006.sh
│   ├── hgapFilterM4_002of006.sh
│   ├── hgapFilterM4_003of006.sh
│   ├── hgapFilterM4_004of006.sh
│   ├── hgapFilterM4_005of006.sh
│   ├── hgapFilterM4_006of006.sh
│   ├── hgapFilterM4.blasrM4Filtered.Gather.sh
│   └── preAssemblerJsonReport.sh
├── P_ReferenceUploader
│   └── runUploaderUnitig.sh
├── Workflow.details.dot
├── Workflow.details.html
├── Workflow.details.svg
├── Workflow.profile.html
├── Workflow.rdf
├── Workflow.summary.dot
├── Workflow.summary.html
└── Workflow.summary.svg
9 directories, 82 files
```

The workflow task subdirectories, prefixed with `P_`, contain all the shell scripts created for the analysis. Inside those scripts are the input(s), output(s) and the actual commands to tools, like `blasr`, `GenomicConsensus`, `runCA`, and so on.

At the top level, the `Workflow.*` files have a SVG graphical output of the workflow DAG, diagramming how each script and outputs interact as part of the overall workflow.

`Workflow*.html` files can be opened in a web browser, and each node in the DAG can be clicked to reveal the script, or log file for that task. The job directory may need to be copied to a location on the file system that can be opened from the browser to view in this fashion.

To view the execution order of the tasks, run this command on either `smrtpipe.log` or `master.log`:

```
grep 'smrtpipe.status refreshTargets' ./master.log
```


P/N 001-353-082-10