SSR_pipeline version 0.95

Contents:

0) *Disclaimer:*

Although this software program has been used by the U.S. Geological Survey (USGS), no warranty, expressed or implied, is made by the USGS or the U.S. Government as to the accuracy and functioning of the program and related program material nor shall the fact of distribution constitute any such warranty, and no responsibility is assumed by the USGS in connection therewith.

1) *Overview:*

SSR_pipeline is a flexible set of programs designed to efficiently identify SSR (microsatellite) sequences from paired-end high throughput Illumina DNA sequencing analyses. The program suite is comprised of three separate analysis modules along with a fourth control module that can be used to automate analyses of large volumes of data. Each of the three separate analysis modules can also be used independently from one another as needed. All modules are implemented in the Python programming language and can therefore be used from nearly any computer operating system (Linux, Mac, Windows). Modules associated with the program as are follows:

- *SSR_pipeline*: This is a "controller" module that can be used to manage the operation of the other three modules contained in *SSR_pipeline*. It is provided to allow users to quickly and efficiently go from raw Illumina data to sets of candidate microsatellite loci as quickly and easily as possible.
- *quality_sort*: This module is used to process the raw FASTQ files produced during Illumina runs and identify the subset of paired-end reads where sequence pairs have passed quality standards.
- *joinseqs*: This module is used to align the two sequences from each paired-end read to produce a longer single DNA sequence. An additional extension to this module (*joinseqs_ext*) contains a set of compiled functions that provide substantial performance improvements. The algorithm implemented in *joinseqs* is essentially a Python-based implementation of the FLASH sequence alignment protocol (Magoč and Salzberg 2011).
- *SSR_search*: This module is used to identify and analyze the set of DNA sequences that contain SSRs (microsatellites). It is based on an extremely efficient and flexible search algorithm that theoretically allows for detection of *any size of microsatellite desired*. In practice, we have frequently experienced memory issues on 32-bit systems when trying to detect motifs larger than 25 bp in size. These limitations are reduced on 64-bit systems.

2) *Installation:*

You can use the programs in two different ways: by downloading stand-alone executable files or working with the original source code.

Stand-alone executables: The first way involves simply downloading sets of pre-generated stand-alone executable files. We have created separate distributions for Windows, Linux, and MacOS using the Python packaging software PyInstaller (http://www.pyinstaller.org/). These executables bundle the programs and a minimal installation of Python into single files so that they can be run on computers that do not already have Python installed. Note that these versions can still be used even if Python is present on a specific machine.

- Windows executables were created using Python 2.7.3 under 32-bit Windows XP and to our knowledge should function just fine under any later Windows version. We have also tested the programs using 32-bit Windows Vista and 64-bit Windows 7 without any issues.
- 32-bit Linux executables were built under CentOS 5.8 using Python 2.4 and glibc (Gnu C Library) version 2.5. These should work with any 32-bit Linux as long as it is using glibc version 2.5 or later (you should be able to check your glibc version by issuing the command “/lib/libc.so.6” or “ldd –version” from the terminal.
- 64-bit Linux executables were built under CentOS 6.2 using Python 2.6.6 and glibc (Gnu C Library) version 2.12. These should work with any 64-bit Linux as long as it is using glibc version 2.12 or later (you should be able to check your glibc version by issuing the command “/lib/libc.so.6” or “ldd –version” from the terminal.
- 64-bit OS X binaries were built under OSX 10.8.2 using Apple's system Python (version 2.7.2). These binaries are compatible with OS X 10.6 and later.

Working with source code: If desired, you may instead download the raw source code, install your own version of Python, and compile the extension module that the program suite uses for some of the analyses. This strategy may be useful if you would prefer to have the program optimized for your particular operating system and hardware, or if the stand-alone executables described above do not work for your system. If you adopt this approach, the file that you download will give you the following files and directory structure once it is unpackaged:

```
.directory_where_you_unpackaged_SSR_pipeline
|   joinseqs.py
|   quality_sort.py
|   SSR_pipeline.py
|   SSR_search.py
|
+---cython_code
|       joinseqs_ext.py
|       joinseqs_ext.pyx
|       readme--compiling_the_extension.txt
|       setup.py
|
+---documentation
|       SSR_pipeline_documentation.doc
|
\---sample_data
        pair1_100k_reads.fastq.gz
        pair2_100k_reads.fastq.gz
        readme_sample_data.txt
        sample_config_file.txt
```

After downloading the files, follow the instructions in the readme file within the cython_code directory for additional instructions. After performing the steps described in the instructions, and new file called “joinseqs_ext.pyd” (Windows) or “joinseqs_ext.so” (Linux, Mac) will be created and copied into the main SSR_pipeline directory.

3) *Working with* SSR_pipeline*:*

After an Illumina run is finished, users should apply the standard Illumina *cassava* pipeline software to their data to perform base calling and quality assessment steps. After completing this process, your data will be ready for analysis with *SSR_pipeline*.

SSR_pipeline is invoked from the command line using the following syntax:

`python SSR_pipeline.py configuration_file_name` (if using the Python source code)

or
`SSR_pipeline.exe configuration_file_name` (if using pre-packaged Windows files)
or
`./SSR_pipeline configuration_file_name` (if using pre-packaged Linux or Mac files).

When running, the program will successively invoke routines from the modules *quality_sort*, *joinseqs*, and *SSR_search* (in that order) using parameter values specified in your configuration files (see below). Analysis output files will be placed in a subdirectory relative to the original data files, and a simple text file called "analysis_report.txt" will be generated that contains a numerical summary of the analysis procedure.

Configuration files are formatted as follows. See below for definitions of variables.

```
nSets=1
#the first line of the file needs to be formatted as above and specify ‘nSets’
#comments can be placed on a line after a '#' sign
#the first line can not be a comment
#your configuration file can contain as may files sets as you would like.
#set1:
read_set_1 = ['1a_40000.fastq.gz']
read_set_2 = ['1b_40000.fastq.gz']
#put an 'r' in front of the path name on Windows machines
path_to_files = r'C:\Illumina_runs\uo5jan2012'
generic_output_name = '40k_test'
keepZipped = 'T'
alignment_Parameters = [10, 0.25, 70]
microsat_search_params = [(2, 7, 40),
                          (3, 6, 40),
                          (4, 5, 40),
                          (5, 4, 40),
                          (6, 4, 40)]
#end of information for a set of files is indicated by a semicolon (';')
#the next set of files to be analyzed can be placed below the semicolon.
;
```

`nSets` specifies the number of file sets you want to analyze

`read_set_1` and `read_set_2` define lists of file names that you want to have analyzed as groups. The number of entries in each list should be the same and specify the congruent reads from paired-end sequencing runs.

`path_to_files` specifies the full path to the data files listed in read_set_1 and read_set_2. Place an ‘r’ in front of the path name on Windows machines.

`generic_output_name` specifies the name of a subdirectory within path_to_files that will be used to hold analysis output files.

`keepZipped` takes on the value `‘T’` or `‘F’.` If `‘T’` (True), then the output files produced by the program will be stored in gzipped format to save disk space.

`alignment_parameters` is a list of values used by the FLASH sequence alignment procedure (see Magoč and Salzberg 2011). The values specify the alignment parameters *`mino_olap`*`,` *`mr`*`,` `and` *`max_olap`*`,` respectively, as described in the paper. See more information about these parameters under ‘**Module *joinseqs***’ below.

`microsat_search_params` is a list of parameter sets that specify how the microsatellite sequence searches will be performed. Each parameter set is specified by a set of three numbers listed within a pair of round brackets that are themselves nested within square brackets. The three numbers refer to the parameters `repeat_len`, `min_repeats`, and `flank_len`. See section '**Module *SSR_search***' below for more information about the definitions of these parameters.

4) ***Working with individual program modules:***

Individual program modules may be run outside of the main *SSR_pipeline* analysis pathway whenever desired. The most typical independent use of these modules will likely involve either 1) generating new paired-end alignments with module *joinseqs* using different sets of analysis parameters, or 2) performing new microsatellite searches with module *SSR_search* using different analysis parameters. Command line usage for each module is described below. Note that the following information can be obtained for each program at a command prompt by invoking the program name without any additional parameters (e.g., by simply typing "quality sort.py" or "quality_sort.exe" or "./quality_sort").

All programs can be used to analyze files that are uncompressed or in gzipped format (the latter used to save disk space). Files that end with the extension '.gz' are assumed to be gzipped. Sequence files can likewise either be fastq or fasta files. If sequence headers begin with the fastq symbol '@' then all sequences are assumed to be in fastq format. Otherwise, all sequences are assumed to be in fasta format.

Module *quality_sort*:

Program usage is as follows:

```
      quality_sort.py f1 f2 subdir keepZipped
```

where

```
      f1 = complete name and path to file 1
      f2 = complete name and path to file 2
      subdir = name of a subdirectory to f1 and f2 where output
            will be written
      keepZipped = T or F, corresponding to True or False. If
            True, then output files will be written as gzipped archives
            to save disk space.
```

Output files from using *quality_sort* will be placed in the subdirectory specified by the 'subdir' parameter. The output files can be easily identified based on the presence of the string '_quality_passed' before the output files' file extension.

Module *joinseqs*: This program module will usually be applied to the output from the *quality_sort* module described above.

Program usage is as follows:

```
      joinseqs.py f1 f2 min_olap mr max_olap keepZipped
```

where

```
      f1 = complete name and path to file 1
      f2 = complete name and path to file 2
      min_olap = smallest sequence overlap to be be evaluated (try using 10)
      mr = the mismatch ratio threshold defining the maximum proportion of
            mismatches allowed within the overlap between two sequences (try
            using 0.25)
      max_olap = maximum sequence overlap beyond which a penalized mr is
            calculated (try using 70)
      keepZipped = T or F, corresponding to True or False. If True, then output
            files will be written as gzipped archives to save disk space
```

The alignment algorithm is based on:
Magoc and Salzberg (2011) FLASH: Fast length adjustment of short reads to
improve genome assemblies. Bioinformatics 27:2957-2963.

Output files from using *joinseqs* will be located in the same directory as the input files used for analysis. The output files can be easily identified based on the presence of the self-explanatory prefixes ‘joined_reads’ and ‘unjoined’.

Module *SSR_search*: Analyses based on *SSR_search* will usually be performed on the output files produced by *joinseqs*. However, the program is sufficiently flexible such that it can also be used to efficiently identify SSR sequences in any fastq or fasta sequence file.

Program usage is as follows:

```
      SSR_search.py f1 repeat_len min_repeats flank_len
```

where

```
      f1 = complete name and path to file to be analyzed
      repeat_len = integer specifying the desired repeat motif length to look
            for (di-, tri-, tetranucleotides, etc.)
      min_repeats = integer specifying the minimum number of repeat units
            that a sequence needs to possess before it will be saved in the
            analysis output
      flank_len = integer specifying the desired length of flanking sequences.
            An SSR must be detected outside of the flanking sequences in order to
            be recorded in output files. Note that by setting this value to 0, all
            sequences containing SSRs can be identified.
```

Output files from using *SSR_search* are produced in FASTA format and can be found in the same directory as the input file used for analysis. Output files can be easily identified based on the presence of the self-explanatory prefix “SSRs” along with the incorporation of analysis parameter values into the output filename. Information regarding the lengths, motifs, and positions associated with each detected SSR are incorporated into the sequence headers within the FASTA file itself. For example, a sequence with the text string “`TAG-8(84,107)`” appended to its FASTA header identifies 8 repeats of the trinucleotide motif TAG, where the repeats span bases 84 through 107 of the sequence itself.

References:

Magoč T and SL Salzberg. 2011. FLASH: fast length adjustment of short reads to improve genome assemblies. Bioinformatics 27:2957-2963.