Pash 3.1

Manual

Program that implements the Positional Hashing method, and ancillary scripts, as described in the following papers:

Coarfa C, Yu F, Miller CA, Chen Z, Harris AR, Milosavljevic A (2010). Pash 3.0: A versatile software package for read mapping and integrative analysis of genomic and epigenomic variation using massively parallel DNA sequencing. BMC Bioinformatics 11:572

Coarfa, C. and Milosavljevic,A (2008). Pash 2.0: Scaleable Sequence Anchoring for Next-Generation Sequencing Technologies. Pacific Symposium on Biocomputing 13:102-113

Kalafus KJ, Jackson AR, and Milosavljevic A (2004). Pash: Efficient Genome-Scale Sequence Anchoring by Positional Hashing. Genome Research 14: 672-678

1. PASH INSTALLATION

==========

The Pash is available as source package with source codes as well as installation package with ready-to-use binaries for Linux 64-bit. We encourage you to use the provided binaries. However, if for some reason you want to build Pash by yourself, see section 3 at the end of this file for compilation details.

The provided binaries require the following environment:

- 64-bit Linux Kernel, version >= 2.6.9
- GNU C Library, version >= 2.5

In general, the binaries should run on any 64-bit Linux distribution supported nowadays.

To install Pash just download the installation package and copy all files from bin directory to target location. Some of the additional tools included in the package need ruby environment (version>=1.8). The best solution is to use ruby interpreter included in your Linux distribution. You can check if ruby is installed by running the following command:

```
ruby --version
```

If you do not have ruby, install it by using a package manager from your Linux distribution

2. RUNNING PASH

==========

2.1. Using Pash to align sequencing reads to a reference genome

a. Aligning reads to a reference genome

In order to align reads to a reference genome using default Pash parameters one can use the following command:

```
pash3 -g ref.fa -r myReads.fastq -o outPashTest.sam
```

Many options are available in Pash that allow one to control sensitivity/specificity of the alignments, and that can lead to significant differences in the time required for alignment. For a list and description of all such options run `pash3` without any arguments, or with the --help option.

Large FASTQ files should be divided into smaller ones before processing. It may be done with the following command:

```
pash3_splitFastq.rb test.fastq 1000000 test.split.1m
```

b. Aligning bisulfite-treated reads to a reference genome

In order to align bisulfite treated reads, one must first process the target genome to make it suitable for bisulfite sequencing mapping. The auxiliary script `pash3_getRCChrom.rb` can be used to perform that task as shown below:

```
pash3_getRCChrom.rb ref.fa ref.dnameth.fa
```

Once the appropriate version of the reference genome has been generated, one can use the `pash3` program with the option `-B` to align bisulfite treated reads to the reference genome.

```
pash3 -g ref.dnameth.fa -r myReads.fastq -o outBisulfite.sam -B
```

2.2. Improving alignment performance by ignoring overrepresented kmers

Overrepresented kmers are not very informative for the correct placing of reads on the genome since they occur in many different places. Ignoring such kmers can lead to a significant decrease in the time that it takes to complete alignments using Pash without much loss in sensitivity. By using the auxiliary scripts provided in the Pash package, the user can calculate the occurence frequency for each kmer of a given

size throughout the target genome. The user can then generate a list of overrepresented kmers and instruct Pash to ignore such kmers in the alignment.

The program that computes the occurrence frequency of kmers is `pash3_keyFreq`. As an example we can calculate the occurrence frequency of kmers in the "genome" `ref.fa` by running the command:

```
pash3_keyFreq -o ref.13.21.kf -p 11101101100011010101l ref.fa
```

Kmers are sub-sequences of the reference genome that are used by Pash as keys in a hash table, allowing for efficient access to positions where such kmers occur in the genome. Kmers could simply be defined as sequences of a certain number of consecutive bases that occur in the genome. However, we have observed that we obtain better performance with Pash if we define Kmers as sequences of bases that are not necessarily consecutive. Such kmers can be defined by first specifying a window size and then selecting a particular subset of the bases in that window to be part of the kmer. The option `-p` in the command above is used to specify how to define the kmers that will be used. The length of the string passed to `-p` defines the size of the window. The positions in that window that will be included in the kmer are defined by the presence of a 1. Positions with 0 will not be part of the kmer. The same pattern must be set during `pash3` run (check `-p` command line option of `pash3`). Probably the best idea is to use one of the predefined pattern. In this case you should specify the name of predefined pattern instead of 0-1 sequence, e.g.:

```
pash3_keyFreq -o ref.13.21.kf -p 13from21 ref.fa
```

To check the list of predefined patterns just run the `pash3_keyFreq` command without any parameters. If no pattern is specified, the predefined pattern `12from18` is used by default.

The default output of the `pash3_keyFreq` program will be in binary format. To generate the occurrence frequency table in a human-readable form, use the `-h` flag as below:

```
pash3_keyFreq -h -o ref.13.21.kf.h -p 13from21 ref.fa
```

A list of kmers to be ignored can be generated using the script `pash3_makeIgnoreList`. In order to keep only kmers that appear a single time in the entire genome one would run the following command:

```
pash3_makeIgnoreList -i ref.13.21.kf -o r13.21.1.il -c 1
```

For mapping against mammalian size genomes, we typically build a kmer histogram, and set a kmer limit in the `pash3_makeIgnoreList` command such that 95% of the genome kmers are considered for possible matches, and only the most overrepresented 5% are discarded.

Finally, one can tell Pash to ignore kmers contained in a list by using the option `-L`:

```
pash3 -g ref.fa -r in.fastq -o out.sam -p 13from21 -L r13.21.1.il
```

3. PASH BUILD INSTRUCTIONS

==========

You have to read this section only if you want to build Pash by yourself. To do that you have to download the package with source codes. The steps described below were tested by us. We used this procedure to prepare binaries for 64-bit Linux included in the corresponding installation package.

You can also try to build Pash on different platform or configuration than described below, but it may require some additional work.

1.1 External dependencies

- headers for glibc (the GNU C library) - we used the version mentioned in section 1 (see beginning of this file).
- gcc/g++ - the GNU C++ compiler, we used version 4.8.4.
- make - the GNU Make, we used version 3.81.
- glib - the GLib library (from www.gtk.org) - both libraries and headers, we used version 1.2.10.

1.2. Specifying build parameters

If necessary, modify the file src/Makefile.include. You need to edit the following options:

```
CC=gcc-4.8
```

```
CXX=g++-4.8
```

gcc/g++ 4.8 is the default option, you can also try different C/C++ compiler.

`GLIB_INCLUDE=`... include options for GLIB on your system.

`GLIB_LIB=`... library location and files for GLIB on your system.

For example, if you are using glib-2.0 installed on your system in /usr/apps/glib-2.0, you would specify

```
GLIB_INCLUDE=-I/usr/apps/glib-2.0/include
```

```
GLIB_LIB=-L/usr/lib -lglib-2.0
```

By default, Pash is using pkg-config to attempt to determine the compiling option for the glib library.

1.3. Building Pash

Run `make` in `src` directory. This will build `pash3`, `pash3_makeIgnoreList` and `pash3_keyFreq` in the `src/pash` and `src/util` directories.