Manual BLSSpeller 1.0

Internet Based Communication Networks and Servives (IBCN)
Department of Information Technology (INTEC)
Ghent University
Contact: Jan Fostier[1]
Department of Plant Systems Biology
VIB - Ghent University
Contact: Klaas Vandepoele[2]

April 2, 2013

[1]jan.fostier@intec.ugent.be
[2]klaas.vandepoele@psb.vib-ugent.be

Contents

References

Dieter De Witte, Michiel Van Bel, Bart Dhoedt, Piet Demeester, Klaas Vandepoele and Jan Fostier (2013)
BLSSpeller: Exhaustive comparative motif discovery of conserved cis-regulatory elements. *Nucleic Acids Research* **xx**, xxx-xxx.

1 Package contents

The main folder - BLSSpellerTool - consists of the following files and directories:

- `cmake` This folder contains cmake related scripts
- `src` The location of the main C++ source code files
- `UnitTest` This folder contains files related to unit testing of the software
- `BLSSpellerManual.pdf` The documentation file (this file)
- `CMakeLists.txt` This file is required to allow cmake to build the code
- `TestSet` This folder contains the Monocot dataset to test the code

2 Installation and requirements

2.1 Basic Installation on Linux

This package requires CMake to build the software. If you don't have CMake already installed, refer to the next section "Installing CMake".
The installation of blsspeller is simple. First, unzip the BLSSpellerTool.tar.gz file:

```
tar -xzvf BLSSpellerTool.tar.gz
cd BLSSpellerTool
```

From this directory, run the following commands:

```
mkdir build
cd build
cmake ..
```

Afterwards run:

```
make
```

It is required that you have a Pthreads library installed. Support for MPI and Googletest unit testing framework is optional.

2.2 Installing CMake

As root, execute the following commands:

- on Redhat / Fedora distributions:

  ```
  yum install cmake
  ```

- on Ubuntu / Debian distributions:

  ```
  aptitude install cmake
  ```

2.3 Installing Google Sparse Hash

The software makes use of the Google Sparse Hash functionalities. This library can be downloaded from: `http://code.google.com/p/sparsehash/?redir=1`

2.4 Running the unit test framework

If the user has installed Googletest the unit tests can be run from his system. To run the unit tests do the following from the BLSSpellerTool directory

```
cd UnitTest
../build/UnitTest/unittest
```

If the unit test fails please do contact us for support.

3 Using BLS Speller

The BLSSpeller software provides 3 main functionalities to the user: de novo comparative motif discovery, pattern matching (with BLS cutoff) and target prediction. The executable `blsspeller` can be found in the `build/src` directory created during the installation. To simplify the commandline we recommend creating a symbolic link from the build directory to the location where your dataset is stored, in our case this would be the Testset directory. To create a symbolic link go to the dataset directory and do the following

```
cd Testset
ln -s ../build/src/blsspeller
```

The main functionalities can now be run as follows:

- Alignment-free De Novo:

  ```
  ./blsspeller -deNovoCompMotifDiscovery parametersAFDiscovery.data
  ```

- Alignment-based De Novo:

  ```
  ./blsspeller -alignmentDiscovery parametersABDiscovery.data
  ```

- Pattern matching:

  ```
  ./blsspeller -patternMatching parametersPM.data
  ```

- Target prediction:

  ```
  ./blsspeller -targetPredictor parametersTM.data
  ```

To get an overview of all scripts included in the package run `./blsspeller` without any arguments. The content of the settings files `parametersXX.data` will be discussed in the next section. The parametersXX.data files are working settings files running on a subset of 100 gene families of the Monocot dataset.

3.1 Settings file

3.1.1 Dataset format

The dataset contains a selection file providing information about the content of the gene families. The filename of the selection file must be added to the settings file:

```
Selection_File=simpleFormat.selection
```

The format of the selection file can be set as follows:

```
Input_Format=SIMPLE, PLAZA
```

The format of the selection file for `Input_Format=SIMPLE` is explained below: The first line of the simpleFormat.selection file looks as follows:

```
iORTHO000001  4 BD1G74660 OS01G01010 SB03G009250 ZM03G07960
```

The first string is the name of the gene family, then the number of genes in the family is given, followed by a list of the actual gene IDs. From this file the software creates a mapping between the name of the orthologous groups (`iORTHO000001`) and the genes which are present in the family. The gene names start with two capital characters indicating the name of the organism.
The actual promoter sequences are stored in one file per organism. This is indicated in the settings file as follows:

```
Sequence_File=bdi_plaza2_5_500bupstream.tfa
```

For every organism one line should be added to the settings file. Two types of content for the sequence file are supported. The standard format is:

```
>BD1G19220
AATCGCAGCGCGC
```

This format (gene name + sequence) is indicated in the settings file as follows:

```
Seq_Format=FASTA
```

For the second type the sequence file contains more information:

```
>BD1G00210  Bd1  BD1G00200  -  BD1G00210  +
ATCTGAACCTTTAAAAAAAATAAGGAACA
```

This format (gene name - chromosome name - flanking gene left - flanking gene right - orientation + sequence) is indicated in the settings file as follows:

```
Seq_Format=MULTIFASTA
```

Note that the extra information is not used by the software.
The input format is very specific, we advice future users to contact us if they would have special requirements concerning the input format. We could look into modifications to allow their data format to be supported or suggest ways to transform their input data.
For alignment-based de novo motif discovery the input data consists of the gene families aligned with a multiple sequence aligner. (Dialign-TX in the paper). This file is specified by:

```
Aligned_Seq_Input=AlignAll500b.txt
```

This file has the same format as the FASTA files described earlier but the sequences can now contain indels.
For both the pattern matching and the target prediction a list of patterns is required, which can be found in the following file:

```
Patterns_Filename=MaizeMotifPatterns.txt, MaizeMotifTargets.txt
```

For the pattern matching one line of the file consists of a motif together with a BLS threshold. For the target predictor only motifs must be supplied by the user, again one per line.

3.1.2 Species tree file

The phylogenetic distances between the organisms are specified using a newick string. (see `http://en.wikipedia.org/wiki/Newick_format`)
The newick file is specified by adding the following line to the settings file:

```
Species_Tree=speciesTree.txt
```

The content of the file is given by a Newick string, as an example the tree used in the Monocots study is given:

```
((BD:0.2688,OS:0.2688):0.0538,(SB:0.086,ZM:0.086):0.2366);
```

Note that the organisms are indicated using two capital characters, which is consistent with the first two capitals of the gene names. This arrangement is strict since every gene family will have a unique phylogenetic tree in which the organism names are substituted with the actual gene names. In case of paralogs the newick string will be modified accordingly. For the Monocots study no phylogenetic trees were available per gene family, if these are available in the user's data we would suggest to contact us, it should not be a big effort to extend the software to support this feature.

3.1.3 Parameters for different algorithms

First, the minimal and the maximal length of the motifs to be discovered is set:

```
kmin=6
kmax=12
```

The alphabet for the motif model is set as follows:

```
Motif_Alphabet=ACGT, IUPAC, TWOFOLDS, TWOFOLDSANDN, DONTCARES, SEVENCHARS
```

- `ACGT`: reports only exact motifs ($size = 4$)
- `DONTCARES`: report motifs allowing for Any characters (N) apart from the four bases ($size = 5$)
- `IUPAC`: reports motifs using the full IUPAC degenerate alphabet ($size = 15$)
- `TWOFOLDS`: report motifs allowing the four bases and 6 twofold degenerate characters ($size = 10$)
- `TWOFOLDSANDN`: same alphabet as previous but also allow N ($size = 11$)
- `SEVENCHARS`: report motifs in an alphabet containing the base chars, N and two degenerate chars S (=A—T) and W (=C—G) which are expected to be more common than other twofold degenerates. ($size = 7$)

The maximum degeneracy of a motif is defined as the product of the degeneracy of all its characters, it therefore represents the maximum number of exact words matching with the motif.

```
Space_Cutoff=64
```

A cutoff of 64 in the `DONTCARES` alphabet implies that a maximum of 3 Ns is permitted ($= 4^3$).
The BLS cutoff can be set by the following parameter and represents the minimal BLS which is taken into account in the confidence charts.

```
BLS_Cutoff=40
```

The confidence chart is given by a staircase approximation where the length of the intervals are set as follows:

```
BLS_Interval=10
```

This implies that the confidence will be calculated for the following BLS thresholds: `BLS_Cutoff`, `BLS_Cutoff+10`, ... For the Monocot simulations this implies that 6 thresholds were analyzed: 40,50, ... 90.
To add the reverse strands of the sequences set following parameter to 1 then the reverse complements of the sequences are generated and added to the gene families.

```
Reverse_Strands=0,1
```

The alignment-based runs are in principle only working on the forward strand. The reverse strand motifs can be added by the following parameter:

```
Reverse_Motifs=1
```

The following parameters determine when a confidence chart is generated. It must occur with at least one BLS threshold with a particular confidence cutoff.

```
Confidence_Cutoff=90
```

On top of that we require the motif satisfying these constraints to simultaneously occur in at least a minimal number of gene families. (this restriction severly limits the output since the majority of motifs only occur in one gene family and are therefore expected to be false positives).

```
Min_FamOcc=5
```

The confidence of a motif for a certain BLS threshold is determined by comparing the number of families in which it occurs (with the given threshold) with the same number for a background consisting of number of control motifs. The minimal number of control motifs is set by the following parameter.

```
#Control_Motifs=20
```

The control motifs are generated via een random shuffling algorithm which generates the following number of shuffles:

```
#Shuffles=1000
```

Only the different control motifs are kept and the set should at least contain 20 motifs to be considered a valid background. The target matching also generates confidence charts and therefore requires a background model. For efficiency reasons here the `Control_Motifs` parameter is used as the number of shuffles generated per pattern.
The number of gene families must also be determined in the settings file. This number serves as an upper boundary to the number of families taken into account. For testing purposes this number can be lowered to for example 100 families.

```
#Genefamilies=17724
```

3.2 Output

First, the algorithm can be run in `Verbose` mode. Then every node will send information on its progress through the dataset and the time required for different steps of the algorithm. Note that this may slow down the algorithm.

```
Verbose=0,1
```

The confidence charts are sent to a file name `NodeXX.txt` where XX denotes the processor ID. The complete motif output can then be generated by concatenating the different Node files (shell command `cat`). The directory where the Node files are written should be specified otherwise no output will be generated. Also note that the user should create the directory before running the algorithm otherwise the output will be lost.

```
Motif_Output=Motifs/
```

The confidence charts have the following format:

```
Motif= CCGCCGGC
bls      conf(med,firsQ,secondQ)          #motifOccs #controlOccs

40   (80,59,100)                      5          1
50       (80,59,100)                      5          1
60       (100,80,100)                     5          0
70       (100,100,100)                    2          0
80       (0,0,0)                          0          0
90       (0,0,0)                          0          0
```

The first column shows the different BLS thresholds for which the confidence is calculated. The second column shows the confidence together with the first and second quartile. The third column corresponds to the number of gene families the motif was found satisfying the BLS threshold and the last column shows the median of the number of occurrences of the background.
The output of the pattern matching is a file containing all matches in the dataset of the motif satisfying the BLS threshold specified in the patterns file. One line of the pattern matching output looks as follows:

```
GACNGAC      50      iORTHO0000001      BD1G74660    +
```

This reads as: the motif `GACNGAC` occurs in gene family `iORTHO0000001` with a BLS $\geq 50\%$ (BLS from patterns file). It occurs on gene `BD1G74660` and resides on the forward strand. For every gene where the motif occurs (and every orientation!) one line is added to the output file. Note that in case of parallel pattern matching every process generate a patterns file. These files must be concatenated afterwards. The target predictor generates a similar file but here the BLS threshold is 0, it thererfore reports all occurrences together with the actual BLS of the motif. Every node of the pattern matcher generates the matches of the gene families the node has analyzed.

```
Pattern_Output=PMOutput/
```

The target predictor also generate the confidence charts for the motifs in the patterns file. Since there is no cutoff defined the thresholds range from 0 to 90.

3.3 Parallellization

The algorithm supports MPI parallelization. It automatically detects whether the system has MPI support. The number of processes can be specified from the command line:

```
> mpirun -np nProcs ./blsspeller -deNovoCompMotifDiscovery parametersXX.data
```

The algorithm is not perfectly scalable in the sense that it is not possible to run the algorithm with degeneracy 64 on 1 core due to RAM limitations. The motifs emitted per gene family are stored in memory, therefore the algorithm can run out of memory and fail. As a rule of thumb one can use the memory requirements to store a motif frequency vector wich is:

$$\text{mem} = 12b + 4b \times \#\text{thresholds} \tag{1}$$

The gene families are approximately uniformly distributed over the cores. (the load balancing takes into account the presence of paralogs in gene families which attributes them a bigger weight)