LeMoNe v2.5 - Manual

Eric Bonnet - Tom Michoel

June 15, 2010

1 Copyright and License

http://sourceforge.net/projects/epsgraphics/ (transferred under the GPL license (herewith attached))

(4) The software is using the Apache CLI library to parse the arguments http://commons.apache.org/cli/ (transferred under the Apache license (herewith attached))

(5) The software is using the BiNGO package to calculate GO enrichment http://www.psb.ugent.be/cbd/papers/BiNGO/. under the GPL license (herewith attached))

2 Installation

You will need the java version 1.6 installed on your system. The current version of LeMoNe is a command-line program. There's no graphical user interface. All you have to do is to unzip the package in a given directory, and you're basically done.

Let's take an example. Say the archive was unpacked in /home/jdoe/progs/. You can call the program by the command:

```
java -jar /home/jdoe/progs/lemone.jar
```

You can pass memory size parameters to the java virtual machine, it might help if you have a large dataset:

```
java -Xmx1000m -jar /home/jdoe/progs/lemone.jar
```

or (very large dataset)

```
java -Xmx10g -jar /home/jdoe/progs/lemone.jar
```

3 Synopsis

The purpose of the LeMoNe software package is to create a module network from expression data. The end result is a set of gene clusters (co-expressed genes), and their associated regulators. To achieve this goal, you'll have to follow the LeMoNe recipe, which consists basically in three steps. For each step, you'll have to use one or more specific part of the software, which are designated as "tasks". All the tasks are performed using the lemone.jar file, except the fuzzy clustering (step 2), which is done in Matlab.

1. Generate several cluster solutions ("ganesh" task).

2. Merge the different cluster solutions using the fuzzy clustering (Matlab procedure).

3. Assign regulators to each cluster, producing the module network ("regulators" task).

4 Usage

You need to specify which task to do with the option -task. There are a variable number of mandatory parameters for each task (see below). Other parameters will be assigned default values. Any non-mandatory parameter default value can be overriden by specifying a value as a command line parameter.

The -mir flag is turning on an option to add extra regulators, like microRNAs (or anything else). The format of the output (xml) file is then slightly different. The extra regulators are taken into account only when assigning regulators, and not before.

4.1 Task: ganesh

Function: cluster genes using a gibbs sampling procedure.

Parameters:
-data_dir: data directory where the input files are stored.
-data_file: expression data matrix.
-output_file: output file name.
-init_num_clust: initial number of clusters (usually half the number of genes).

Optional parameters:
-lambda: lambda parameter (default 0.1).
-mu: mu parameter (default 0.0).
-alpha: alpha parameter (default 0.1).
-beta: beta parameter (default 0.1).
-num_runs: number of runs (default 1).
-use_bayesian_score: use Bayesian score to cluster experiments (default true).
-burn_in: burn in steps for the gibbs sampler (default 50).
-num_steps: Total number of steps for the Gibbs sampler (default 100).
-sample_steps: steps for sampling (default 100).
-mir: use microRNA file format (default false).

-score_gain: cutoff score for hierarchical trees (default 0.0).

Examples:

```
java -Xmx1000m -jar lemone.jar -task ganesh -data_dir data/ -data_file
matrix.txt -init_num_clust 6000 -output_file net01
```

with extra regulators:

```
java -Xmx1000m -jar lemone.jar -task ganesh -mir -data_dir data/
-data_file mRNA.txt -init_num_clust 6000 -output_file net01
```

4.2 Task: fuzzy

Function: print out the file for the fuzzy clustering procedure (Matlab code).

Parameters:
-data_dir: data directory containing one or more xml files.
-output_file: output file name (text file).

Optional parameters:
-mir: use microRNA file format (default false).
Examples:

```
java -Xmx1000m -jar lemone.jar -task fuzzy -data_dir data/
-output_file fuzzy
```

With extra regulators:

```
java -Xmx1000m -jar lemone.jar -mir -task fuzzy -data_dir data/
-output_file fuzzyextra
```

4.3 Task: print_modules

Function: print all the clusters from one ganesh run.
Parameters:
-data_dir: data directory where the input files are stored.
-output_file: output file name (text file).

Optional parameters:
-mir: use microRNA file input (default false).

Examples:

```
java -Xmx1000m -jar lemone.jar -task print_modules -data_dir data/
-output_file clusters.txt
```

With extra regulators:

```
java -Xmx1000m -jar lemone.jar -mir -task print_modules -data_dir data/
-output_file clusters.txt
```

4.4 Task: regulators

Function: cluster experiments with gibbs sampling, build hierarchical trees and assign regulators.

The regulators can be of two types, depending on the nature of the data. Their expression value can be continuous, like expression values measured with microarrays (for transcription factors, signal transducers, kinases, microRNAs, etc..). Regulators can also have discrete data, like for example a clinical parameter having only 0 or 1 values. The calculation of the score is slightly different if the regulator is continuous or discrete. The regulator type is indicated in the input file (see section 6 of the manual).

Parameters:
-data_dir: data directory where the input files are stored.
-data_file: expression data matrix.
-reg_file: list of candidate regulators.
-cluster_file: list of clusters.
-num_clust: number of clusters in the cluster file (max. index value).
-output_file: output file name.
-mir_file: microRNA expression data input file (mandatory with -mir flag).

Optional parameters:
-lambda: lambda parameter (default 0.1).
-mu: mu parameter (default 0.0).
-alpha: alpha parameter (default 0.1).
-beta: beta parameter (default 0.1).
-num_runs: number of runs (default 1).

-use_bayesian_score: use Bayesian score to cluster experiments (default true).
-burn_in: burn in steps for the gibbs sampler (default 100).
-num_steps: Total number of steps for the Gibbs sampler (default 1100).
-sample_steps: steps for sampling (default 100).
-mir: use microRNA file input (default false).
-score_gain: cutoff score for hierarchical trees (default 0.0).
-num_reg: number of regulators assigned per node (default 10).

Examples:

```
java -Xmx1000m -jar lemone.jar -task regulators -data_dir data/ -data_file
matrix.txt -reg_file reg.txt -cluster_file tight_clusters.txt
-num_clust 99 -output_file reg01
```

with extra regulators:

```
java -Xmx1000m -jar lemone.jar -task regulators -data_dir data/ -data_file
mRNA.txt -mir_file micro.txt -reg_file reg.txt -cluster_file tight.txt
-num_clust 99 -output_file reg01
```

4.5 Task: print_regulators

Function: print regulators for each cluster and their score. The default behavior of this task is to print only the regulators that have a score above a cutoff value defined as the top 1% of all assigned regulators.

Parameters:
-data_dir: data directory where the input files are stored.
-output_file: output file name (text file).

Optional parameters:
-mir: use microRNA file input (default false).
-matlab: start the numbering of the modules at 1 instead of 0.
-all_regulators: print all the regulators instead of the top 1%.

Examples:

```
java -Xmx1000m -jar lemone.jar -task print_regulators -data_dir data/
-output_file reg.txt
```

With extra regulators:

```
java -Xmx1000m -jar lemone.jar -mir -task print_regulators -data_dir data/
-output_file reg.txt
```

4.6 Task: print_random_regulators

Function: print all randomly assigned regulators for each cluster and their score.

Parameters:
-data_dir: data directory where the input files are stored.
-output_file: output file name (text file).

Optional parameters:
-mir: use microRNA file input (default false).
-matlab: start the numbering of the modules at 1 instead of 0.

Examples:

```
java -Xmx1000m -jar lemone.jar -task print_random_regulators -data_dir data/
-output_file reg.txt
```

With extra regulators:

```
java -Xmx1000m -jar lemone.jar -mir -task print_random_regulators -data_dir data/
-output_file reg.txt
```

4.7 Task: go_annotation

Function: calculate GO enrichment in the modules.

-data_dir: data directory where the input files are stored.
-output_file: output file name (text file).
-go_annot_file: GO annotation file name.

Optional parameters:
-go_ref_file: set of genes used as a reference for GO calculations (default: all modules genes).
-go_ontology_file: Type of ontology used (default: GO_Biological_process).
-go_p_value: GO p value cutoff (default 0.05).
-go_annot_def: flag to indicate that a custom annotation file will be used, and not the BINGO internal one.
-mir: use microRNA file input (default false).

Examples:

```
java -Xmx1000m -jar lemone.jar -mir -task go_annotation -data_dir data/
-output_file go.txt -go_annot_file H_sapiens_default
```

In this case the annotation file is taken from the BiNGO distribution. There are a whole bunch of files available:

```
C_elegans_default
C_jejuni_RM1221_default
D_ethenogenes_195_default
D_melanogaster_default
E_chaffeensis_Arkansas_default
G_gallus_default
H_sapiens_default
M_musculus_default
O_sativa_japonica_default
P_falsiparum_3D7_default
S_cerevisiae_default
S_pombe_default
```

It's also possible to provide a custom made go annotation file.

```
java -Xmx1000m -jar lemone.jar -mir -task go_annotation -data_dir data/
-output_file go.txt -go_annot_file mygo.txt -go_annot_def
```

A list of gene to be used as the reference to calculate GO enrichment can be provided, otherwise the whole set of genes contained in the modules will be used as the reference.

```
java -Xmx1000m -jar lemone.jar -mir -task go_annotation -data_dir data/
-output_file go.txt -go_annot_file C_elegans_default
-go_ref_file mygenes.txt
```

4.8 Task: figures

Function: generate a figure for each module (cluster genes + regulators).

The figures are created in the EPS format (encapsulated postscript).

Parameters:
-data_dir: data directory where the input files are stored.
-top_regulators: text file containing an ordered list of regulators for each cluster.

Optional parameters:
-mir: use microRNA file input (default false).
-use_global_mean: flag to use the global mean of the dataset as the reference for the colors (default is to use the mean of the module).

Examples:

```
java -Xmx1000m -jar lemone.jar -task figures -data_dir data/
-top_regulators top_reg.txt
```

With extra regulators:

```
java -Xmx1000m -jar lemone.jar -mir -task figures -data_dir data/
-top_regulators top_reg.txt
```

5 Fuzzy clustering

A couple of matlab scripts are available in the matlab/ directory to perform fuzzy clustering. The procedure is the following:

```
% launch matlab and include the scripts in your path
addpath('/home/jdoe/progs/LeMoNe_v2.0/matlab')

% build probabilities matrix from fuzzy clustering text file
[s,data,idx] = makepairwiseprob('fuzzy.txt');

% perform fuzzy clustering
p = matrixClustSym(s)

% perform optimal cutoff to get tight clusters
[xopt, qopt,n1,n2,nc] = optCutoff(p,s);

% write tight clusters to output file
writeclusters(qopt,idx,'tight_clusters_opt.txt')
```

6 File formats

All data files are simple tab-delimited text files.

The datafile contain expression values for every gene, and is formatted like this:

```
<header line>
gene_id1  description value1 value2 .... valueN
gene_id2  description value1 value2 .... valueN
 ...
gene_idM  description value1 value2 .... valueN
```

The cluster file indicate to which cluster any gene belongs to.

```
geneid1 cluster_number
geneid2 cluster_number
...
```

The gene IDs and the cluster numbers are separated by a single space.

The regulators file contains a list of genes that will be considered as candidate regulators to learn the module network. The file is a tab delimited text file having two columns. The first column corresponds to the gene ID, while the second determine the type of the regulator: "d" for discrete regulators (e.g. clinical parameters) and "c" for continuous data types (transcription factors, signal transducers, microRNAs, etc.). If the file has only one column consisting of gene IDs, they will be considered as continuous by default.

7 Advanced tasks

Assigning regulators to modules can be time-consuming. To facilitate this process, we have added a couple of tasks that basically split the assignment of regulators for each module. Each module assignment can be run separately in parallel, on a computer cluster for example.

7.1 Task: experiments

Function: cluster experiments with Gibbs sampler and build hierarchical trees, without regulators assignment.

-data_dir: data directory where the input files are stored.
-data_file: expression data matrix.
-reg_file: list of candidate regulators (mandatory, but won't be used).
-cluster_file: list of clusters.
-num_clust: number of clusters in the cluster file (max. index value).
-output_file: output file name.

Optional parameters: -lambda: lambda parameter (default 0.1).
-mu: mu parameter (default 0.0).
-alpha: alpha parameter (default 0.1).
-beta: beta parameter (default 0.1).
-num_runs: number of runs (default 1).
-use_bayesian_score: use Bayesian score to cluster experiments (default true).
-burn_in: burn in steps for the gibbs sampler (default 100).
-num_steps: total number of steps for the Gibbs sampler (default 1100).
-sample_steps: steps for sampling (default 100).
-score_gain: cutoff score for hierarchical trees (default 0.0).

7.2 Task: split_reg

Function: assign regulators for a range of modules.

Parameters:
-data_dir: data directory where the input file is stored. -output_file: output file name.

Optional parameters:
-num_reg: number of regulators assigned (default 10).
-beta_reg: beta parameter for regulators assignment (default 20).
-range: range of modules where regulators will be assigned (e.g. 0:10).

7.3 Task: split_concatenate

Function: build one xml file from several split regulators assignment files.

-data_dir: data directory where the input file is stored.
-output_file: output file name.