NAME

clmprotocols − Work flows and protocols for mcl and friends

DESCRIPTION

A guide to doing analysis with mcl and its helper programs.

Network representation

The clustering program **mcl** expects the name of file as its first argument. If the **--abc** option is used, the file is assumed to adhere to a simple format where a network is specified edge by edge, one line and one edge at a time. Each line describes an edge as two labels and a numerical value, all separated by white space. The labels and the value respectively identify the two nodes and the edge weight. The format is called ABC-format, where 'A' and 'B' represent the two labels and 'C' represents the edge weight. The latter is optional; if omitted the edge weight is set to one. If ABC-format is used, the output is returned as a listing of clusters, each cluster given as a line of white-space separated labels.

MCL can also utilize a second representation, which is a stringent and unambiguous format for both input and output. This is called *matrix format* and it is required when using other programs in the mcl suite, for example when comparing and analysing clusterings using **clm(1)** or when extracting and transforming networks using **mcx(1)**. Native mode (matrix format) is entered simply by *not* specifying **--abc**.

The recommended approach using **mcl** is to convert an external format to ABC-format. The program **mcxload(1)** reads the latter and creates a native network file and a dictionary file that maps network nodes to labels. All applications in the MCL suite, including **mcl** itself, can read this native network file format. Label output can be obtained using **mcxdump(1)**. The workflow is thus:

```
#  External format has been converted to file data.abc (abc format)

mcxload -abc data.abc --stream-mirror -write-tab data.tab -o data.mci

mcl data.mci -I 1.4
mcl data.mci -I 2
mcl data.mci -I 4

mcxdump -icl out.data.mci.I14 -tabr data.tab -o dump.data.mci.I14
mcxdump -icl out.data.mci.I20 -tabr data.tab -o dump.data.mci.I20
mcxdump -icl out.data.mci.I40 -tabr data.tab -o dump.data.mci.I40
```

In this example the cluster output is stored in native format and dumped to labels using mcxdump. The stored output can now be used to learn more about the clusterings. An example is the following, where **clm(1)** is applied in mode **dist** to gauge the distance between different clusterings.

```
clm dist --chain out.data.mci.I{14,20,40}
```

Loading large networks

If you deal with very large networks (say with hundreds of millions of edges), it is recommended to use binary format (cf **mcxio(5)**). This is simply achieved by adding `--write-binary` to the mcxload command line. The resulting file is no longer human-readable but will be faster to read by a factor between ten- or twenty-fold compared to standard MCL-edge network format, and a factor around fifty-fold compared to label format. All MCL-edge programs are able to read binary format, and speed of reading will be somewhere in the order of millions of edges per second, compared to, for example, roughly 100K edges per second for label format.

Memory usage for mcxload can be lowered by replacing the option `--stream-mirror` with `-ri max`.

Converting between formats

Converting label format to tabular format

Label format, two or three (including weight) columns:

```
mcxload -abc data.abc --stream-mirror -write-tab data.tab -o data.mci
mcxdump -imx data.mci -tab data.tab --dump-table
```

Simple Interaction File (SIF) format:

```
mcxload -sif data.sif --stream-mirror -write-tab data.tab -o data.mci
mcxdump -imx data.mci -tab data.tab --dump-table
```

These two examples are very similar, and differ only in the way the input to **mcxload** is specified.

Using threading and job dispatching

The programs **mcxarray**, **mcx clcf**, **mcx ctty**, **mcx diameter** and **clm info2** can all make use of both threading and job dispatching. The clustering program **mcl** can only use threading.

Instructing these programs to use threads is easy. It just requires supplying **-t <num>**, e.g. use **-t 4** to generate four threads. It is only sensible to use `<num>` threads on a machine that has at least `<num>` CPUs. It is additionally recommended that a threaded program has exclusive access to those CPUs and does not have to contend with other jobs.

For the afore-mentioned programs it is additionally possible to split the computational load over multiple machines. If `<N>` machines are available then `<N>` jobs should be started. Each job should have an identical parameter **-J N** (e.g. **-J 10**), and varying parameters **-j 0**, **-j 1**, ... **-j N-1** (e.g. **-j 9**). It is possible to use threads in each individual job, but the number of threads should be identical across all jobs issued. Output should typically be directed using a convention such as **-o out.0**, **-o out.1**, ... **-o out.9**.

After all jobs have finished the outputs must be combined to form the final answer. The manner in which this is done is dependent on the program used. With the example output above this would be done as follows. It can be seen that **clm info2** is not yet supported by **mcx collect** and requires somewhat idiosyncratic processing.

```
# mcx diameter:
   mcx collect --add-column -o out.diameter out.{0,1,2,3,4,5,6,7,8,9}

# mcx ctty:
   mcx collect --add-column -o out.ctty out.{0,1,2,3,4,5,6,7,8,9}

# mcx clcf:
   mcx collect --add-column -o out.clcf out.{0,1,2,3,4,5,6,7,8,9}

# mcxarray:
   mcx collect --add-matrix -o out.ctty out.{0,1,2,3,4,5,6,7,8,9}

# clm info2:
   clxdo add_table out.{0,1,2,3,4,5,6,7,8,9} > out.info2
```

Clustering similarity graphs encoded in BLAST results

A specific instance of the workflow above is the clustering of proteins based on their sequence similarities. In the most typical scenario the external format is BLAST output, which needs to be transformed to ABC format. In the examples below the input is in columnar blast format obtained with the blast -m8 option. It requires a version of **mcl** at least as recent as `09-061`. First we create an ABC-formatted file using the external columnar BLAST format, which is assumed to be in a file called `seq.cblast`.

```
cut -f 1,2,11 seq.cblast > seq.abc
```

The columnar format in the file `seq.cblast` has, for a given BLAST hit, the sequence labels in the first two columns and the asssociated E-value in column 11. It is parsed by the standard UNIX cut(1) utility. The format must have been created with the BLAST -m8 option so that no comment lines are present. Alternatively these can be filtered out using grep. The newly created `seq.abc` file is loaded by **mcxload(1)**, which writes both a network file `seq.mci` and a dictionary file `seq.tab`.

```
mcxload -abc seq.abc --stream-mirror --stream-neg-log10 -stream-tf 'ceil(200)'
      -o seq.mci -write-tab seq.tab
```

The `--stream-mirror` option ensures that the resulting network will be undirected, as recommended when using **mcl**. Omitting this option would result in a directed network as BLAST E-values generally differ between two sequences. The default course of action for **mcxload(1)** is to use the best value found between a pair of labels. The next option, `--abc-neg-log10` tranforms the numerical values in the input (the BLAST E-values) by taking the logarithm in base 10 and subsequently negating the sign. Finally, the transformed values are capped so that any E-value below 1e-200 is set to a maximum allowed edge weight of 200.

To obtain clusterings from `seq.mci` and `seq.tab` one has two choices. The first is to generate an abstract clustering representation and from that obtain the label output, as follows. Below the **-o** option is not used, so mcl will create meaningful and unique output names by itself. The default way of doing this is to preprend the prefix `out.` and to append a suffix encoding the inflation value used, with inflation encoded using two digits of precision and the decimal separator removed.

```
mcl seq.mci -I 1.4
mcl seq.mci -I 2
mcl seq.mci -I 4
mcl seq.mci -I 6

mcxdump -icl out.seq.mci.I14 -tabr seq.tab -o dump.seq.mci.I14
mcxdump -icl out.seq.mci.I20 -tabr seq.tab -o dump.seq.mci.I20
mcxdump -icl out.seq.mci.I40 -tabr seq.tab -o dump.seq.mci.I40
mcxdump -icl out.seq.mci.I60 -tabr seq.tab -o dump.seq.mci.I60
```

Now the file `out.seq.tab.I14` and its associates can be used for example to compute the distances between the encoded clusterings with **clm dist**, to compute a set of strictly reconciled nested clusterings with **clm order**, or to compute an efficiency criterion with **clm info**.

Alternatively, label output can be obtained directly from **mcl** as follows.

```
mcl seq.mci -I 1.4  -use-tab seq.tab
mcl seq.mci -I 2  -use-tab seq.tab
mcl seq.mci -I 4  -use-tab seq.tab
mcl seq.mci -I 6  -use-tab seq.tab
```

Clustering expression data

The clustering of expression data constitutes another workflow. In this case the external format usually is a tabular file format containing labels for genes or probes and numerical values measuring the expression values or fold changes across a series of conditions or experiments. Such tabular files can be processed by **mcxarray(1)**, which comes installed with **mcl**. The program computes correlations (either Pearson or Spearmann) between genes, and creates an edge between genes if their correlation exceeds the specified cutoff. From this **mcxarray(1)** creates both a network file and a dictionary file. In the example below, the file `expr.data` is in tabular format with one row of column headers (e.g. tags for experiments) and one column of row identifiers (e.g. probe or gene identifiers).

```
mcxarray -data expr.data -skipr 1 -skipc 1 -o expr.mci -write-tab expr.tab --pearson -co 0.7 -tf 'abs(),add(-0.7)'
```

This uses the Pearson correlation, ignoring values below 0.7. The remaining values in the interval `[0.7-1]` are remapped to the interval `[0-0.3]`. This is recommended so that the edge weights will have increased contrast between them, as **mcl** is affected by relative differences (ratios) between edge weights rather than absolute differences. To illustrate this, values 0.75 and 0.95 are mapped to 0.05 and 0.25, with respective ratios 0.79 and 0.25. The network file `expr.mci` and the dictionary file `expr.tab` can now be used as before.

It is possible to investigate the effect of the correlation cutoff as follows. First a network is generated at a very low threshold, and this network is analysed using **mcxquery**.

```
mcxarray -data expr.data -skipr 1 -skipc 1 -o expr20.mcx --write-binary --pearson -co 0.2 -tf 'abs()'
mcx query -imx expr20.mcx --vary-correlation
```

The output is in a tabular format describing the properties of the network at increasing correlation thresholds. Examples are the size of the biggest component, the number of orphan nodes (not connected to any other node), and the mean and median node degrees. A good way to choose the cutoff is to balance the number of singletons and the median node degree. Both should preferably not be too high. For example the number of orphan nodes should be less than ten percent of the total number of nodes, and the median node degree should be at most one hundred neighbours.

Reducing node degrees in the graph

A good way to lower node degrees in a network is to require that an edge is among the best *k* edges (those of highest weight) for *both* nodes incident to the edge, for some value of *k*. This is achieved by using `knn(k)` in the argument to the **-tf** option to mcl or **mcx alter**. To give an example, a graph was formed on translations in Ensembl release 57 on 2.6M nodes. The similarities were obtained from BLAST scores, leading to a graph with a total edge count of 300M, with best-connected nodes of degree respectively 11148, 9083, 9070, 9019 and 8988, and with mean node degree 233. These degrees are unreasonable. The graph was subjected to **mcx query** to investigate the effect of varying k-NN parameters. A good heuristic is to choose a value that does not significantly change the number of singletons in the input graph. In the example it meant that **-tf 'knn(160)'** was feasible, leading to a mean node degree of 98.

A second approach to reduce node degrees is to employ the **-ceil-nb** option. This ranks nodes by node degree, highest first. Nodes are considered in order of rank, and edges of low weight are removed from the graph until a node satisfies the node degree threshold specified by **-ceil-nb**.

SEE ALSO

mcxio(5).

AUTHOR

Stijn van Dongen.