FMotifs Guide

V2.0

Centre for Biomedical Technology, UPM
Madrid, Spain

General information

FMotifs is a program for the exhaustive counting of 3-nodes motifs. Thanks to its internal structure, it is especially suited for the study of medium-size networks with high link density, e.g. functional network representation reconstruction of brain dynamics.

Contact email: massimiliano.zanin@ctb.upm.es

***FMotifs* Version:** 2.0
This guide version: 2.0

Availability: The program for Windows environment, as well as C++ source code are available at www.mzanin.com

Changes history

Version 2.0
Added support for parallel computation.
Disconnected nodes are now disregarded from the computation.
Error log file generated in case of unexpected events.
General performance improvements.

Version 1.0
Initial version of the program.

The idea behind *FMotifs*

Motifs[1] are one of the simplest and yet of the most powerful topological characteristics that can be assessed in a network, defined as specific patterns of interconnections created by a small number of connected nodes.

While motifs have traditionally been used to characterize large networks, e.g. transcriptional regulation networks, they have recently been applied to the study of functional network representations of brain dynamics[2]. While such networks are usually small, of the order of hundreds of nodes, their high link density poses a challenge for traditional algorithms that are designed for the study of large sparse graphs.

FMotifs is a software designed to handle such situations, by performing exhaustive 3-nodes motifs enumeration in medium-size dense networks. The computational cost mainly depends on the number of nodes, with link density having little effect on the time required to obtain the result. As an example, for a network of 150 nodes and link density of 0.5, the computation time is reduced from 10.6 seconds of the MFINDER tool[3], down to 0.10 seconds. *FMotifs* has been designed for a simple integration with other programs, e.g. MATLAB, thus allowing the batch analysis of large sets of networks. Furthermore, it takes advantage of multi-core processors through OpenMP, enabling important speed improvements when such hardware is available.

[1] Milo, R., Shen-Orr, S., Itzkovitz, S., Kashtan, N., Chklovskii, D., & Alon, U. (2002). *Network motifs: simple building blocks of complex networks*. Science, 298(5594), 824.
[2] Zanin, M., Sousa, P., Papo, D., Bajo, R., García-Prieto, J., del Pozo, F., Menasalvas, E., & Boccaletti, S. (2012). *Optimizing Functional Network Representation of Multivariate Time Series*. Scientific Reports, 2.
[3] Kashtan, N., Itzkovitz, S., Milo, R., & Alon, U. (2002). *Mfinder tool guide*. Department of Molecular Cell Biology and Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot Israel, Tech. Rep.

Using *FMotifs*

Installation

FMotifs is available in two forms:

1. A command-line executable for Windows environments. Just download *FMotifs.exe* into your working directory, and execute it with the appropriate command line options. In order to work correctly, two DLL files are needed (included in the distribution): *libgomp-1.dll* and *pthreadGC2.dll*.

2. A C++ class, which can be used in any custom development.

Executing FMotifs at the *command prompt*

FMotifs.exe <input network file name> [-am | -par]
Calculates the motifs for the input file specified. Network should be codified in a raw text format (e.g., a *txt* file), specifying each one of the links. The format for links should be the following:

<source node> <tab> <destination node> <tab> <weight>

For instance, the following text describes a network of three nodes connected in a triangular motif:

```
1    2    1
2    3    1
3    1    1
```

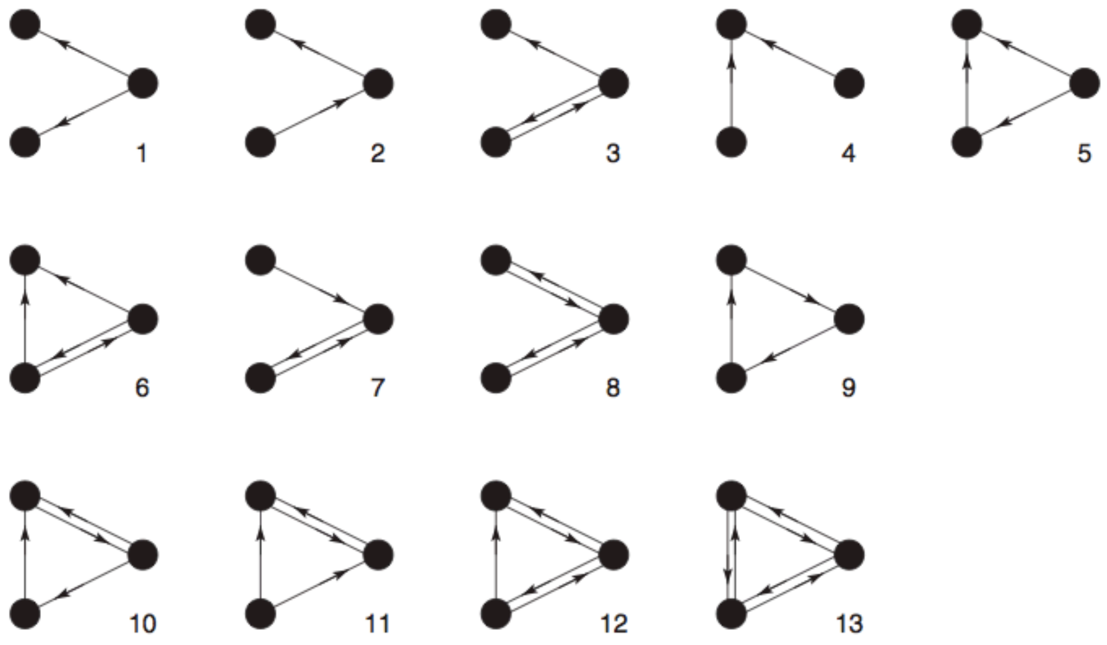
Notice that the weight of links is disregarded, and is included in the input file for compatibility with other programs, e.g. MFINDER. Node numbering can both start from zero or one, as the program will automatically disregard any unconnected node. The output is saved in a text file, with each row specifying the number of instances detected for each motif. The output file name is:

<input network file name>_MAT.txt

The previous example would result in the following output:

1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	1
10	0
11	0
12	0
13	0

Thus, only one instance of motif number 9 has been detected. The numeration of motifs follows the usual convention, i.e.:



The –*am* option
Calculates the motifs for the input file specified, with the network formatted as an adjacency matrix. The previous example would read:

0	1	0
0	0	1
1	0	0

The –*par* option
Enables the multi-core parallel processing of the input file, implemented in *OpenMP*[4]. The program firstly detects the number of cores available in the machine, for then splitting the computation into different instances. This results in a reduction of the computational cost proportional to the number of cores (results may vary depending on the machine architecture).

FMotifs.exe -info
Returns information about the program, including its version.

Example networks

Two example networks are provided at www.mzanin.com/FMotifs, representing the brain activity of a control subject, and of a patient suffering from Mild Cognitive Impairment. For further information, please refer to:

Zanin, M., Sousa, P., Papo, D., Bajo, R., García-Prieto, J., del Pozo, F., Menasalvas, E., & Boccaletti, S. (2012). *Optimizing Functional Network Representation of Multivariate Time Series*. Scientific Reports, 2.

Integration with 3rd party software

One of the aims behind the creation of *FMotifs* was the development of a tool that could be easily integrated with 3rd party software, in order to perform automated analysis of large network collections. In what follows, we present an example of how this integration can be performed with a MATLAB program. The example is organized in three parts: preparing a file with the input, execution of *FMotifs*, and loading the results into MATLAB.

1. Preparing the input

Let us suppose one is to analyze a network that is stored in a matrix *AM*, of size $n \times n$, *n* being the number of nodes. Each element of the matrix has a value of 1 when a link exists between the two considered nodes, and 0 otherwise. The following code saves this network in a format suitable to be analyzed by *FMotifs*:

[4] For more information, please refer to http://www.openmp.org.

```
fid = fopen(fileName, 'wt');
for n1 = 1 : numNodes
        for n2 = 1 : numNodes
                if n1 == n2
                        continue;
                end
                if AM(n1, n2) == 1
                        fprintf(fid, '%d\t%d\t1\n', n1, n2);
                end
        end
end
fclose(fid);
```

Notice that *filename* codifies the name of the destination file, and *numNodes* the number of nodes composing the network.

2. Executing *FMotifs*

Once the input network has been saved to the file *filename*, the following code will execute *FMotifs*:

```
myCommand = ['FMotifs.exe ' fileName ];
[status, result] = system( myCommand );
```

In order to avoid errors, the program *FMotifs.exe* should be in the same directory of the MATLAB script, or in a directory accessible through its *path*. Furthermore, if one wants to take advantage of the parallel capabilities of *FMotifs*, the first line should be modified as follows:

```
myCommand = ['FMotifs.exe ' filename ‘ -par ];
```

3. Retrieving the results

Finally, the motifs should be retrieved from the output file, whose name will be the same as the input file, plus a “_MAT” ending. The following code retrieves such results, and stores them in a *motif* vector:

```
tempRes = load( fileNameOut );
motifs = tempRes(:, 2);
```

Notice that *fileNameOut* encodes the name of the file where outputs are returned.

Finally, it may be useful to delete the two intermediate files that have been created in this process; this can be performed with the following code:

```
delete( fileName );
delete( fileNameOut );
```

Using the *FastMotifs3* class in a C++ program

Instead of executing *FMotifs* from an external program, it may be useful to directly integrate its code inside a custom C++ software, thus avoiding the files writing/reading steps. In what follows, two situations are considered: a standard analysis, and a parallel analysis.

Standard, single-thread analysis

As a first step, it is necessary to initialize the *FastMotifs3* class, and allocate the necessary memory for storing the adjacency matrix:

```
FastMotifs3 *myMotifs = new FastMotifs3();
myMotifs->AllocMemory( numNodes );
```

As in the previous example, numNodes codifies the number of nodes composing the network. Next, it is necessary to fill the adjacency matrix with the values corresponding to the analyzed networks; in what follows, we suppose this information is stored in the *myNet* array:

```
memcpy( myMotifs->AM, myNet, sizeof(bool) * numNodes * numNodes );
```

Finally, the main function of the class should be called:

```
myMotifs->CalculateMotifs();
```

Results are stored in the *myMotifs->Motifs* array, which can be accessed for any subsequent computation.

Standard, single-thread analysis

The second example we proposes deals with the enumeration of motifs in large networks; in such cases, it may be useful to "split" the calculation in different parts, so that each one of them can be executed in different CPUs.

The first part of the process, involving the class initialization and the network information preparation, is the same as exposed above:

```
FastMotifs3 *myMotifs = new FastMotifs3();
myMotifs->AllocMemory( numNodes );
memcpy( myMotifs->AM, myNet, sizeof(bool) * numNodes * numNodes );
```

The actual motif enumeration is performed by means of the following two functions:

```
myMotifs->ParCreateIndices(num_cores);
myMotifs->ParCalculateMotifs(act_core);
```

The first one, *ParCreateIndices*, splits the computation among different parts, here indicated by the variable *num_cores*. Next, the function *ParCalculateMotifs* enumerates the motifs that are present in the i^{th} part, as defined by *act_core*. For the sake of clarity, let us suppose that the calculation is splitted in two parts. The first program should then execute:

```
myMotifs->ParCreateIndices(2);
myMotifs->ParCalculateMotifs(0);
```

while the second program:

```
myMotifs->ParCreateIndices(2);
myMotifs->ParCalculateMotifs(1);
```

The total number of motifs is then given by the sum of the results obtained by both programs.