DiVa

User manual for the <u>Di</u>sconnectivity <u>Va</u>luation command line tool

Björn Goemann

Department of Bioinformatics, UMG
Georg-August University Göttingen, Germany
bjoern.goemann@bioinf.med.uni-goettingen.de

Göttingen, October 2012

Contents

1 Introduction

DiVa (Disconnectivity Valuation tool) is a tool that analyzes a network by means of the network centrality the pairwise disconnectivity index (PDI). The PDI estimates how significant a network element is for sustaining the pairwise connections between vertices in a network. Network elements can be vertices, edges or topological patterns (like motifs). The significance of such an element is determined by simulating its knockout from the network and comparing the number of connected ordered pairs of vertices in the network before and after the knockout. Consider [1, 2] for detailed information about the methodology.

DiVa recognizes networks that are represented as directed or undirected graphs and can be used for the following purposes:

1. Screen a network by computing the PDI for either all vertices or edges to determine which of these elements may play a significant role for the connectivity of the network
2. Perform a targeted analysis by selecting particular elements or groups of them to determine their importance
3. Determine if and which kind of network patterns like motifs are important for the architecture of a network

This version of DiVa can be used to calculate the PDI on a single computer as well as in a computer grid to speed up the running time in case of very large networks.

2 License

This software can be used freely except for commercial purposes. Any publication containing results that are obtained from DiVa has to acknowledge the use of the program.

Please note, that there are absolutely no guarantees or warranties made by the available files for DiVa. Your are working with this program at your own risk.

3 Installation and requirements

Download the binary or the c++ source code from `http://www.bioinf.med.uni-goettingen.de/services/diva`. The binary has been compiled using g++ in version 4.4 on Ubuntu 12-4.

Making your own DiVa binary requires a C++ compiler and the C++ boost library in version 1.46 or higher installed on your system. The boost library is available at `http://www.boost.org`.

4 Usage

To start DiVa open a console, change to the location of the binary and enter

./DiVa <configuration file>

where *<configuration file>* is the name of DiVa's configuration file.

5 The configuration file

Each line of the configuration file starts with the name of a *<parameter>* followed by the delimitter = and the corresponding *<value>*, i.e.

<parameter>=<value>

The configuration file must contain all of the parameters as indicated below although not each one may require a value when running DiVa. Write

<parameter>=

if you do not whish do indicate a value, as for example if there is no processing file that has to be recognized for the computations. Start a line with the character # followed by some text or before the definition of a parameter to declare a comment, i.e.

# This is a comment

The parameters of the configuration file are as follows:

calculation_item
Defines on which kind of network element the PDI should be computed. Requires one of the values: *vertex* for the vertices of a graph, *edge* for its edges or *pattern* for topological patterns.

calculation_target
Defines the aim of the computation and requires one of the values: *chunk*, *pdi* or *full*. Write *full* if the parameter **runtime_mode** is set to *standalone* or *master* to compute the PDI on the choosen **calculation_item**. The values *chunk* and *pdi* are internal parameters that are used for running DiVa in a computer grid. The value *chunk* ensures that a DiVa slave process just computes the transtive closure(s) of the source network and thereby produces its so-called chunk result. After all slave processes are finished, the master process interprets the chunk results and computes the PDI. This is indicated in the configuration file of this master process by using the value *pdi*.

chunk_config_file_prefix
Defines the file name prefixes of the configuration files for the slave processes and for the final master process when DiVa is running on a computer grid. The rule for creating the file name for a slave process configuration file is *chunk_config_file_prefix<ChunkID>.txt* where *ChunkID* is the number of the respective slave process. For the final master process the rule is *chunk_config_file_prefix.master*.

chunk_log_file_prefix
Defines the file name prefixes of the log files for the slave processes when DiVa is running on a computer grid. The rule for creating the file name fpr the log file of a slave process is *chunk_log_file_prefix<ChunkID>.txt* where *ChunkID* is the number of the respective slave process.

chunk_path
Stands for the directory into which temporary configuration and result files are written to when DiVa is running on a computer grid.

chunk_processing_file_prefix
Defines the file name prefixes of the processing files for the slave processes when DiVa is running on a computer grid. The rule for creating the file name for the processing file of a slave process is *chunk_processing_file_prefix<ChunkID>.txt* where *ChunkID* is the number of the respective slave process.

chunk_result_file_prefix
Defines the file name prefixes of the chunk result files for the slave processes when DiVa is running on a computer grid. The rule for creating the file name of the chunk result file for a slave process is *chunk_result_file_prefix<ChunkID>.txt* where *ChunkID* is the number of the respective slave process.

directed_graph
Defines whether the source graph is directed or not. Use *1* if the graph is directed otherwise *0*.

graph_file
Stands for the full path and name of the file that contains the source graph.

log_device
Defines which device is used for logging. Select *console* for logging on the console window, *file* for writting log messages into the file specified with the parameter **log_file** and *both* for using both of the devices for logging.

log_file
The full path and name of the log file into which logging messages are written to when the parameter **log_device** is set to *file* or *both*.

log_level
Defines which kind of messages are logged. Select *INFO* for info messages, *ERROR* for error messages and *WARN* for warnings.

max_single_cores
Not used at the moment.

processing_file
The full path and name of a processing file that contains a set of single and/or groups of network elements (either vertices, edges or patterns, depending on the value of the parameter **calculation_item)** for which the PDI has to be computed.

result_file
Stands for the full path and file name of the result file(s) produced by DiVa.

runtime_mode
Defines whether DiVa is processing the calculations in sequential or parallel manner. Sequential manner means that all knockouts are done one after another, e.g. if you wish to compute the PDI for all vertices in a network then DiVa does the knockout for the first vertex, then for the second and so on. This procedure is typically the case when using DiVa on a single computer and is indicated by the value *standalone*. In contrast, parallel manner means that the knockouts are done simultaneously by running DiVa on several CPUs at the same time, e.g. when using a computer grid/cluster. In this case, choose the value *master* which enables DiVa to generate all necessary slave configuration and processing files to do the calculations simultaneously. In each slave configuration file, the value of this parameter is set to *slave*.

separator
Indicates the delimitter used in all input and output files.

single_core_threshold
Defines the maximum number of knockouts achieved by a single slave process of DiVa. When the parameter **runtime_mode** is set to *master*, the value of this parameter is used to determine the number of slave processes, with each slave process performing a fraction of the whole set of knockouts. For example, if you wish to compute the PDI for all vertices in a network that consists of 1000 vertices and the parameter **single_core_threshold** is set to 100, DiVa creates 10 slave configurations and processing files with each processing file containing 100 of the 1000 vertices. Hence, you may use 10 CPUs that may do these calculations in parallel.

6 Input files

6.1 Network file

A network has to be represented as a list of edges without multiple edges and edge weights, i.e.

[fromVertex] [separator] [toVertex]

where *[fromVertex]* is the source and *[toVertex]* the sink of an edge. The term *[separator]* refers to a character separator between the vertices and has to be defined in the configuration file.

Example 1 *A network containing the vertices A to E whose edges are separated by a semicolon should be formatted like*

```
A;B
B;C
B;D
C;D
C;E
D;B
```

6.2 Processing file

Instead of computing the PDI for all vertices, edges or patterns in a network, DiVa supports making a targeted analysis for user-defined subsets of such network elements. Subsets can be defined on vertices or edges and a subset may represent a single vertex/edge as well as a group of vertices/edges. Groups of vertices/edges are collections of at least two vertices/edges. Similar to a multiple gene knockout in the lab, the PDI for a group of vertices/edges is calculated for the whole collection instead of each vertex/edge separately.

To use the subset mechanism, supply DiVa with a processing file by setting the parameter **processing_file** in the configuration file 5. The respective processing file has to be formatted in such way that each row contains one subset. In the case of a single vertex, write the name of the vertex as defined in the network file in a row, i.e.

[Vertex]

Define a subset that consists of a group of vertices by placing the names of the vertices as given in the network file in a row. In this case, use the delimiter similar to the one in the network file to distinguish between the vertices of a group, i.e.

[Vertex1] [separator] [Vertex2] [separator] [Vertex3] ...

Example 2 *Recall the network*

```
A;B
B;C
B;D
C;D
C;E
D;B
```

and suppose you wish to estimate the PDI for the vertices A and B. In addition, you're interested in the vertices A, B and C as a group of vertices. The corresponding subset file would look like:

```
A
B
A;B;C
```

If you wish to calculate a subset of edges the contents the processing file may contain single edges as well as groups of edges. In the case of a single edge, specify the edge in the same way as in the network file, i.e.

[source] [separator] [sink]

where [source] is the vertex at the beginning of an edge and [sink] at then end. A group of edges can be defined by writing the respective edges in one row. Each edge must be indicated in the same way as in the network file, i.e.

[source] [separator] [sink] [separator] [source] [separator] [sink] ...

6.3 FanMod dump file

Estimating the topological significance of network patterns like motifs with DiVa requires their detection in a given network at first. Since the latter is not supported by DiVa, it has to be achieved by an external software. Currently, DiVa can interpret the output from the widely used FanMod software for motif detection [3]. To identify topological patterns in your network, run FanMod and make sure that you have enabled the optional dump file output in CSV format. DiVa requires the resulting dump file for estimating the PDI of the topological patterns in your networks. In the configuration file of DiVa, set the resulting dump file as the value for the parameter **processing_file** [5].

7 Output files

7.1 Chunk result file

A chunk result file contains the temporary results as calculated by a DiVa slave process when using DiVa's feature of distributed computing. In particular, each row of a chunk result file refers to the number of connected ordered pairs of vertices in the subnetwork G' of your input graph G that is achieved by removing the network element(s) in the same row of the corresponding processing file of the slave process.

Example 3 *Suppose, that line 2 of the processing file of the i-th slave process contains the edge*

A;B

Then the second row of the corresponding chunk result file, i.e.

123

refers to those connected ordered pairs of vertices whose connectivity is not affected by removing the edge from vertex A to B from the input network.

7.2 Result files

The result files contain the estimated PDI values for the network elements as defined in DiVa's configuration file. Depending on the value of the parameter **calculation_item** one or two result files are produced by DiVa.

If the parameter **calculation_item** is set to *vertex* or *edge* one result file is written. If no processing file was given, each row contains either a vertex or an edge followed by the PDI value. If a processing file was given a row may also refer to a group of edges or vertices followed by the corresponding PDIV value. Each column in the result file is separated using the delimitter defined in the configuration file.

If the parameter **calculation_item** is set to *pattern* two result files are written. While one file contains the PDI value for all pattern instances found in your network, the other one contains the PDI for the patterns themselves[1]. The latter file contains two columns: the first one refers to the adjacency matrix code of a pattern and the second column to the PDI. The columns are separated by using the delimitter defined in the configuration file.

The same delimitter is used in the result file for the pattern instances

[1]The PDI of a pattern is the average PDI of its instances.

in which each row refers to one pattern instance. The first column stands for the corresponding pattern of an instance in adjacency matrix code and each of the following columns contains one of the participating vertices of the instance. The last row comprises the estimated PDI value.

7.3 Log file

Depending on the choosen parameters **log_device** and **log_level** error, info and/or warning messages of DiVa are written into the log file as given by the parameter **log_file** in the configuration file.

8 Sequential and distributed computing

This section describes the two basic strategies (sequential or distributed) that can be used to estimate the PDI values for a number of network elements with DiVa. Consider the necessary steps for computing the PDI for a single network element e:

1. Count the number of connected ordered pairs of vertices in graph G
2. Build the subgraph G' of G that differs from G in that G' does not contain the network element e
3. Count the number of connected ordered pairs of vertices in the subgraph G'
4. Compute the PDI for the network element e

The approach used for counting the number of connected ordered pairs of vertices in a graph G in DiVa is to compute the transitive closure of G which runs significantly faster than the standard Warshall algorithm. The C++ boost library supplies an implementation of a very efficient transitive closure algorithm which is used in a modified version in DiVa.

However, while the first step above is a one-time only calculation when estimating the PDI for numerous network elements, the steps 2 to 4 have to be executed for each of the respective network elements. The overall time for estimating the PDI for numerous network elements significantly depends on the size of graph G and the total number of these network elements. Fortunately, the calculcation of the PDI (in particular the time-consuming steps 2 and 3) for a network element e is independent from the same calculation for another network element f. As a consequence, instead of estimating the PDI values for a number of network elements in a *sequential* manner, this task may also be achieved in *parallel* thereby reducing the runtime of DiVa.

8.1 Sequential computing

The standard way for estimating the PDI of some network elements is to use a sequential strategy, especially when you intend to run DiVa on a single computer. In this case, DiVa executes the necessary steps for estimating the PDI values for one network element after another, i.e. it computes the PDI of network element e before the PDI of network element f. This is a good choice for small- to medium-sized networks with less than 10000 vertices and 50000 edges, since DiVa does this job very quickly. Setting up DiVa to act in this manner requires to set the parameter **runtime_mode** to *standalone*.

8.2 Distributed computing

8.2.1 How it works

In cases, when you have large or very large networks with hundreds of thousands or millions of vertices/edges the sequential strategy for computing the PDI is likely to be inefficient. If you have a multi-processor computer or even a computer grid/cluster you can make use of DiVa's feature to parallelize the calculations. This is realized by running several instances of DiVa, i.e. *slave processes*, at the same time on different CPUs/computers with each slave process executing just a fraction of the total calculations. The configuration of these slave processes is done by DiVa.

Figure 1 shows how this feature works in principle when using a computer grid (the workflow is the same when using a several CPUs of one computer). The basic idea of this feature is that a DiVa master process initially builds the processing queue that contains all network elements for which the PDI has to be computed. Then, based on the settings in the configuration file of DiVa the number of slave processes is calculated and this queue is divided into several sub-queues with each queue being executed by a slave process. However, the task of a slave process is not to estimate the PDI for the network elements of its processing queue, because this would mean to count the number of connected ordered pairs of vertices in the source graph G again and again. Therefore, a slave process does just the time-consuming parts of the PDI calculation for each of the network elements e in its processing file which means to count the number of connected ordered pairs of vertices in the subgraph $G' = G \backslash \{e\}$. Subsequently, the DiVa master process collects the so-called chunk results of all slave processes in order to compute the desired PDI values very quickly.

Altogether, letting DiVa do the calculations in parallel consists of three steps:

1. The first step is to set up an appropriate DiVa master configuration file (see below) and to run DiVa with this configuration file. DiVa

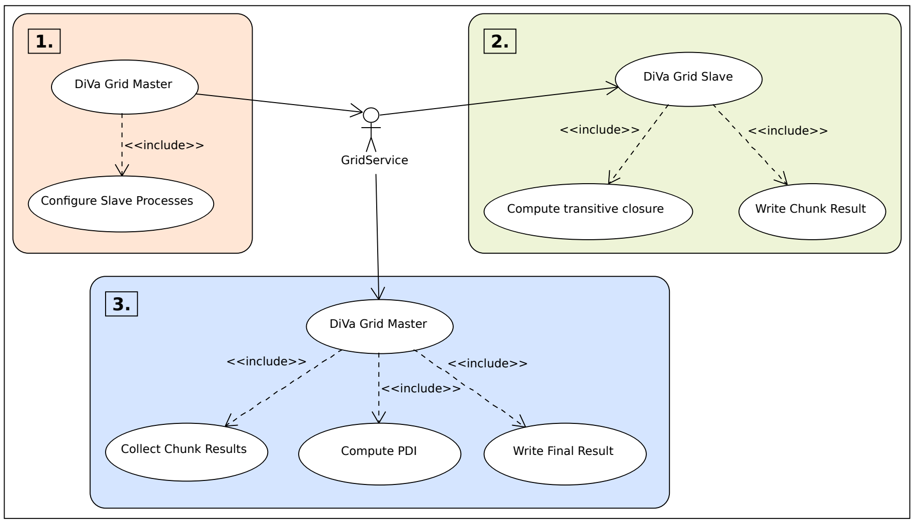

Figure 1: Workflow of DiVa when running on a computer grid.

automatically creates the configuration and processing files for the slave processes. In addition, another configuration file for the final DiVa master process call is created which collects the chunk results and computes the PDI.

2. The second step is to add all slave processes to the job processing queue of your grid service. In particular, this means adding program calls to DiVa with the created slave configuration files such as

   DiVa <i-th slave configuration file>

   that can be executed on a computer in the grid.

3. When all slave processes have finished their calculations run DiVa once again with the configuration file for the final DiVa master process to calculate the PDI and to output the final results.

8.2.2 Configuring DiVa

To enable the feature of DiVa of distributing the calculations set the parameter **runtime_mode** in the configuration file to *master*. In addition consider all parameters starting with the name **chunk** to define the names of the input/output files of the slave processes and the directory into which these files are written to.

When running DiVa with your configuration file the following things happen:

- For each slave process, a configuration file will be created in the directory as given by the parameter **chunk_path** in your configuration file. Based on the value of the parameter **chunk_config_file_prefix** the rule for creating the file name is *chunk_config_file_prefix<ChunkID>.txt*. Finally, a configuration file for the final DiVa master process is created in the same directory by using the file name *chunk_config_file_prefix<ChunkID>.master*
- For each slave process, a processing file will be created in the directory as given by the parameter **chunk_path** in your configuration file. Based on the value of the parameter **chunk_processing_file_prefix** the rule for creating the file name is *chunk_processing_file_prefix<ChunkID>.txt*.

The values of the parameters **chunk_log_file_prefix** and **chunk_result_file_prefix** in your configuration file will be used for the configuration files of the slave processes. Hence, each slave processes writes its results in a file called *chunk_result_file_prefix<ChunkID>.txt* in the directory as given by the parameter **chunk_path** in your configuration file.

9 Example configurations

The following example configurations as well as the indicated example network are taken from the DiVa source code package. The example network represents transcriptional regulation in Escherichia Coli and has been published in [4].

9.1 All vertices / single computer

This configuration computes the PDI for each vertex in the network *ecoli.txt* on a single computer:

```
# Run DiVa on a single CPU
runtime_mode=standalone
# Compute the PDI for vertices
calculation_item=vertex
# Calculate the PDI
calculation_target=full
# Full path and name of the input network file
graph_file=ecoli.txt
# The vertices of each edge are delimitted using the ';' character
separator=;
# The input graph is directed
directed_graph=1
# Full path and name of the result file
result_file=ecoli_vertices.pdi
# No processing file is given
processing_file=
# No need to define, since DiVa is running on a single CPU
chunk_path=
# No need to define, since DiVa is running on a single CPU
chunk_config_file_prefix=
# No need to define, since DiVa is running on a single CPU
chunk_log_file_prefix=
# No need to define, since DiVa is running on a single CPU
chunk_processing_file_prefix=
# No need to define, since DiVa is running on a single CPU
chunk_result_file_prefix=
# Full path and name of the log file
log_file=ecoli_vertices.log
# Log all kinds of messages
log_level=INFO
# Log into a file
log_device=file
# Not relevant when running on a single CPU
single_core_threshold=100
# Not relevant when running on a single CPU
max_single_cores=1
```

9.2 Subset of vertices / single computer

This configuration computes the PDI for a specific subset of vertices of the network *ecoli.txt* on a single computer:

```
# Run DiVa on a single CPU
runtime_mode=standalone
# Compute the PDI for vertices
calculation_item=vertex
# Calculate the PDI
calculation_target=full
# Full path and name of the input network file
graph_file=ecoli.txt
# The vertices of each edge are delimitted using the ';' character
separator=;
# The input graph is directed
directed_graph=1
# Full path and name of the result file
result_file=ecoli_vertices.pdi
# Calculate the PDI only for those vertices/groups of vertices
# contained in the processing file
processing_file=ecoli_vertices_subset.pdi
# No need to define, since DiVa is running on a single CPU
chunk_path=
# No need to define, since DiVa is running on a single CPU
chunk_config_file_prefix=
# No need to define, since DiVa is running on a single CPU
chunk_log_file_prefix=
# No need to define, since DiVa is running on a single CPU
chunk_processing_file_prefix=
# No need to define, since DiVa is running on a single CPU
chunk_result_file_prefix=
# Full path and name of the log file
log_file=ecoli_vertices.log
# Log all kinds of messages
log_level=INFO
# Log into a file
log_device=file
# Not relevant when running on a single CPU
single_core_threshold=100
# Not relevant when running on a single CPU
max_single_cores=1
```

9.3 All edges / computer grid

This configuration computes the PDI for all edges in the network *ecoli.txt* on a computer grid:

```
# Run DiVa on a several CPUs or in a computer grid
runtime_mode=master
# Compute the PDI for edges
calculation_item=edge
# Calculate the PDI
calculation_target=full
# Full path and name of the input network file
graph_file=ecoli.txt
# The vertices of each edge are delimitted using the ';' character
separator=;
# The input graph is directed
directed_graph=1
# Full path and name of the result file
result_file=ecoli_vertices.pdi
# No processing file is given
processing_file=
# Temporary results are written into the current directory
chunk_path=
# The file name prefix of the configuration files of the slave processes
chunk_config_file_prefix=ecoli_chunk_configuration
# The file name prefix of the log files of the slave processes
chunk_log_file_prefix=ecoli_chunk_log
# The file name prefix of the processing files of the slave processes
chunk_processing_file_prefix=ecoli_chunk_processing
# The file name prefix of the result files of the slave processes
chunk_result_file_prefix=ecoli_chunk_result
# Full path and name of the log file when running DiVa with this configuration file
log_file=ecoli_vertices.log
# Log all kinds of messages
log_level=INFO
# Log into a file
log_device=file
# The processing file of a slave process contains 100 edges at most
single_core_threshold=100
# Not relevant
max_single_cores=1
```

References

[1] A.P. Potapov, B. Goemann, and E. Wingender. The pairwise disconnectivity index as a new metric for the topological analysis of regulatory networks. *BMC Bioinformatics*, 9:227, 2008.

[2] B. Goemann, E. Wingender, and A. P. Potapov. An approach to evaluate the topological significance of motifs and other patterns in regulatory networks. *BMC Systems Biology*, 3:53, 2009.

[3] S. Wernicke and F. Rasche. Fanmod: a tool for fast network motif detection. *Bioinformatics*, 22:1152–1153, 2006.

[4] S. Shen-Orr, R. Milo, S. Mangan, and U. Alon. Network motifs in the transcriptional network of e. coli. *Nature genetics*, 31:1 – 5, 2002.