

C3Part/Isofun User's guide

Version v2.0 - Jan. 2011

This documentation is still unfinished and needs polishing - feedback is welcome -

1	Presentation.....	2
1.1	Presentation	2
1.2	System requirements	2
1.3	Distribution.....	2
2	Basic concepts	3
2.1	Layered datagraph	3
2.2	Spines and aggregators	3
2.3	Common connected components	4
3	Input file format.....	4
3.1	Extended DIMACS format.....	4
3.1.1	the 'p' line	4
3.1.2	the 'n' lines	4
3.1.3	the 'e' lines	4
3.1.4	the 'c' lines	5
3.2	keywords for 'n' and 'e' lines	5
3.2.1	the 'color' keyword	5
3.2.2	the 'label' keyword	7
3.2.3	the 'comment' keyword.....	7
3.2.4	genome-oriented keywords.....	7
3.2.5	user defined keywords	7
4	Main program.....	8
4.1	Running the main program manually.....	8
4.2	program options	8
4.3	Program output.....	10
4.3.1	example 1:	10
4.3.2	example 2:	11
5	Shell scripts	13

Contacts :

Alain.Viari@inrialpes.fr

1 Presentation

1.1 Presentation

The C3Part/Isofun package implements a generic approach to merge information from two or more graphs representing biological data, such as genomes, metabolic pathways or protein-protein interactions, in order to infer functional coupling between them. The theory is described in refs [1, 2] and will not be detailed here. This documentation is targeted to end users. It provides information about practical usage of the C3Part/Isofun program. A separate documentation (developer's guide) is (actually will-be) provided for developers, in order to access the C3Part library directly from their own programs.

References

[1] Syntons, Metabolons and Interactons: an exact graph-theoretical approach for exploring neighbourhood between genomic and functional data.

F. Boyer, A. Morgat, L. Labarre, J. Pothier and A. Viari
Bioinformatics 21:4209-4215 (2005)

[2] Multiple alignment of biological networks: A flexible approach.

Y.-P. Deniérou, F. Boyer, M.-F. Sagot, and A. Viari.

In CPM'09, volume 5542 of *Lecture Notes in Computer Science*, pages 173-185.
Springer-Verlag, 2009.

[3] Bacterial syntenies an exact approach with quorum.

Y.-P. Deniérou, F. Boyer, M.-F. Sagot, and A. Viari.

BMC Bioinformatics - submitted.

[4] Y.-P. Deniérou - PhD Thesis (in French, add reference and link).

Notes

- ref [1] corresponds to the (older) C3Part/C package, implemented in C and available at : <http://www.inrialpes.fr/helix/people/viari/cccpart/index.html>. This version is no longer supported and has been superseded by the current C3Part/Isofun package.
- ref [2,3] corresponds to the current C3Part/Isofun package, implemented in Java. This version provides several improvements over the C version.

1.2 System requirements

The C3Part/Isofun package is implemented in Java and therefore can be run on any system providing a Java machine (Unix, Windows). By default the package has been compiled using JDK 6 and need the Java 6 standard environment to run properly. However sources are also compatible with JDK 5. In this case you need to recompile the classes before using the program, see the README file at the distribution root for instructions.

In addition the package also contains few shells scripts to help launching the main program and formatting its output. These scripts are only useful for Unix users (MacOSX, Linux, etc...). They require GNU gawk (> 3.1) and standard unix tools.

1.3 Distribution

README : readme file

LICENSE : CeCILL (aka GPL) license file
build.xml : **ant** configuration file for recompiling and unit testing
doc/ : documentation (User's guide and Java API)
lib/ : java libraries (Isofun.jar)
samples/ : sample input graphs
scripts/ : optional unix scripts (unix only)
src/ : java sources
tests/ : for program testing (unix only)

Note:

the 'tests' directory contain some global tests (for unix users only).

To run these tests issue :

```
cd tests
./dotest
```

this will run a couple of tests (about 10 to 15 minutes).

Each output line should end with "ok". In case of error don't worry too much (the test script is not polished), but please drop me an email with a cut and paste of the output.

2 Basic concepts

We briefly recall here some basics concept needed to understand this documentation. more details can be found in ref [2].

2.1 Layered datagraph

The biological networks to be studied are represented by a single graph, called a 'layered datagraph'.

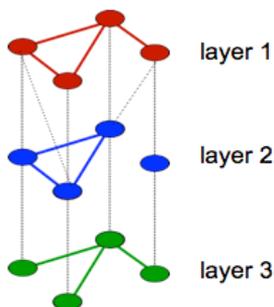


Figure 1

In this a graph, each layer represents a biological network (genome, ppi, metabolic network) and interlayer edges (dotted lines in Figure 1) represent a correspondence relation between vertices of each network (note that this relation is not necessarily one-to-one).

2.2 Spines and aggregators

\$\$to be finished\$\$

2.3 Common connected components

\$\$to be finished\$\$

3 Input file format

3.1 Extended DIMACS format

The input file, describing the layered datagraph, is composed of 3 parts (that should appear in this order) :

a single 'p' line // optional - graph title
a series of 'n' lines // mandatory - list of vertices
a series of 'e' lines // mandatory - list of edges

Each line is composed of a signal character key ('p', 'n' or 'e' respectively), followed by fields separated by any number of blanks or tab characters.

3.1.1 the 'p' line

the 'p' line is optional, it has been kept for backward compatibility with previous versions but is currently unused.

```
p <title>
```

3.1.2 the 'n' lines

the 'n' lines are mandatory. They describe the layered-datagraph vertices.

each 'n' line has the form :

```
n [<keyword>=<value>]*
```

<keyword> and <value> are described hereafter.

Note: the vertices are (1-based) numbered in the same order as they appear in the input file. Hence the first vertex receives index 1, the second receives index 2 and so on.

3.1.3 the 'e' lines

the 'e' lines are mandatory. They describe the layered-datagraph (undirected) edges.

each 'e' line has the form :

```
e <index1> <index2> [<keyword>=<value>]*
```

<index1> : the 1-based index for the first vertex

<index2> : the 1-based index for the second vertex

note: since the graph is undirected, there is no need to duplicate edges. Doing so will generate a warning.

3.1.4 the 'c' lines

In addition to 'p', 'n' and 'e' line, you may add any number of lines starting with 'c' to be used as comment lines. These lines are simply ignored by the reader.

3.2 keywords for 'n' and 'e' lines

There are two types of keywords : builtin (i.e. predefined) keywords and user's keywords.

builtin keywords are listed in the following table :

Keyword	For vertex	For edges	Description
color	mandatory	optional	color of vertex or edge (see below)
label	optional	unused	vertex label
comment	optional	optional	comment
rank	optional	unused	gene rank
orientation	optional	unused	gene orientation

3.2.1 the 'color' keyword

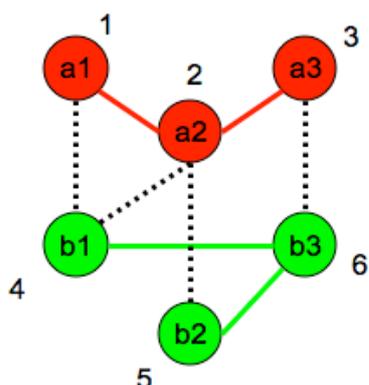
This keyword is mandatory for vertices ('n' lines). It denotes the layer to which the vertex (or edge) belongs. This is indicated as:

color=<number>

where <number> is a 1-based index of the layer (arbitrarily chosen) or '0' to indicate inter-layer edges.

example:

The following 2-layers-datagraph



may write as :

```
n label=a1 color=1
n label=a2 color=1
n label=a3 color=1
n label=b1 color=2
n label=b2 color=2
n label=b3 color=2
e 1 2 color=1
e 2 3 color=1
e 4 6 color=2
e 5 6 color=2
e 1 4 color=0
e 2 4 color=0
e 2 5 color=0
e 3 6 color=0
```

Notes:

- color=0 is reserved to edges (to denote the interlayer correspondence relation). Assigning color=0 to a vertex will raise an error.

- you probably have noticed that specifying color for edges is somehow redundant. Indeed the color of an edge can be easily deduced from its connected vertices (i.e. the vertices color if vertices are of the same color or 0 if they are of different colors). Therefore specifying a color for edge is optional and the previous example may write as well as :

```
n label=a1 color=1
n label=a2 color=1
n label=a3 color=1
n label=b1 color=2
n label=b2 color=2
n label=b3 color=2
e 1 2
e 2 3
e 4 6
e 5 6
e 1 4
e 2 4
e 2 5
e 3 6
```

- an alternate way for specifying color is to use the 'colors' keyword instead of 'color', in the following way :

```
colors=<binary>
```

where <binary> is a string composed of 0's and at most one '1'. The position of the '1' (reading from right to left) indicates the color. The rightmost position corresponds to color 0. For instance

```
colors=100 is equivalent to color=2
colors=1   is equivalent to color=0
```

This (ugly) form is provided only for backward compatibility with previous versions of C3Part but its usage is strongly discouraged.

All other builtin keywords are optional

3.2.2 the 'label' keyword

This keyword is only useful for vertices. It allows to assign a label to a vertex. This label is further used by some scripts to produce more readable output. The syntax is :

```
label=<alphanum>+
```

where alphanum = [a-z] | [A-Z] | [0-9] | '_'

Note: it is also possible to use blanks (or non alphanumeric characters) in label. To do so, the label value should be enclosed within double quotes

```
label="<character>+"
```

examples:

```
n label=dnaA
n label="my beautiful gene"
```

3.2.3 the 'comment' keyword

This keyword (that can be used both for vertices and edges) is mostly used to make the input file more friendly. The value is currently unused by the program. The syntax is :

```
comment="<character>*"
```

3.2.4 genome-oriented keywords

In addition, for some genome-oriented applications, the following keywords can be assigned to vertices :

```
rank=<number>
orientation=+|-
```

rank is the (1-based) rank of the gene on the chromosome
orientation indicates the chromosome strand : '+' for direct strand and '-' for reverse 'strand'

These keywords are used by some options of the Isofun program (see below)

3.2.5 user defined keywords

Finally, you may define your own keywords by :

```
<key>=<value>
```

key : <alphanum>+
value : <alphanum>+ | "<character>+"

This is only useful if you are writing our own program (using the C3Part Java library).

4 Main program

4.1 Running the main program manually

Once you have defined the input layered datagraph, you can run the main program by :

```
java [jvm_options] -cp <dist_root>/lib/IsoFun.jar  
helix.graph.program.isofun.Main <input_file> [isofun_options]
```

jvm_options : Java options. The most useful are :

-Xms<size> set initial Java heap size
-Xmx<size> set maximum Java heap size

dist_root : root of package distribution

input_file : layered datagraph input file

isofun_options : Isofun options. each option as the form :

--option_name value

options names and values are described in the following table

4.2 program options

Option_name	Value	Default	Description
algo	otf otfql otfqg	otf	various algos otf : regular algo without jokers, described in ref [2] otfql, otfqg : otf with quorum still unpublished note: other values are still experimental
minsize	integer	2	ccc min size i.e. the minimum number of multinodes in the ccc
mineltsize	integer	2	ccc min element size i.e the minimum number of different datanodes in the ccc
aggregator	clique star cc dense integer	clique	node aggregator - tell how datanodes are aggregated to form a spine. clique: spine is a clique star: spine is a star centered on the first color (see also --optimizer)

			cc: spine is a connected component dense: spine is a cc with edge density 'nn' % (nn is an integer in range 0, 100)
maxstar	integer	0	set the maximum number of jokers (for otfqg and otflq algos only)
mincore	integer	0	set the minimum number of core spines (for otfqg and otflq algos only)
colors	Integer+	keep all datagraph colors	keep only specified colors for processing example: --colors 2 3
optimizer	off local global user integer+	local	select colors permutation optimization off: no optimizer (use colors in the same order as given in input) local: optimize colors locally (during split) global: optimize colors globally user: specify color order example: --colors user 1 3 2
deltagap	Integer+	0	Perform delta closure(s) of datagraph before processing. you may provide a list of delta's, one foreach color. Example: --deltagap 0 5
debug	on off	off	turn on/off debugger
verbose	on off	off	turn on/off verbosity
memdog	on off	off	try to report the actual memory used (slow down process)
check	on off	on	check datagraph colors integrity before processing
cleanup	on off	on	cleanup datagraph before processing
reportgaps	on off	off	report on gene contiguity and gaps in syntons. This options requires the 'rank' and 'orientation" keyword on vertices (genes).

In addition the following options are **still experimental** and should be used with caution
(please contact me before using them)

deltaeps	Integer+	-1	Subnetworks topology conservation default value (-1) means no conservation (default CCC behavior). A value of 0 means strict topology conservation (this may be used with genomes for instance to keep strict gene order). Other values should not (yet) be used.
tandem	on off	off	Gene tandem pre-filtering.
lookahead	allofcolor allpath oneway twoways	twoways	Lookahead control (for otflq algo only)
compressor	off inset	off	Graph compression mode

4.3 Program output

By default the program produces output on <stdout> (console) in the following format :

each connecton is introduced by a 'q' line followed by 'v' lines. the 'q' line describes the size and number of elements in the connecton. the 'v' lines describe the spines.

```
q size n1 ... nk
v label1 ... labelk rank1 ... rankk '| '|
```

k : number of colors (i.e. layers)

size : number of multinodes in the connecton

ni : number of different datanodes in the connecton

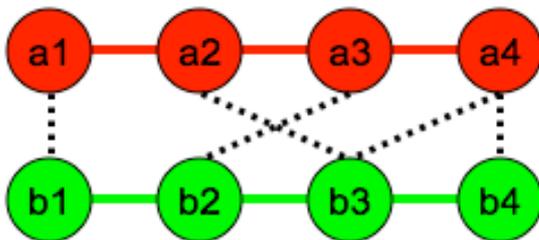
labeli : label of datanode

ranki: rank of datanode (if provided with the 'rank' keyword, else 0)

note: the rank and '| '| fields are provided for backward compatibility with previous versions.

4.3.1 example 1:

the following 2-colors datagraph



is described by the samples/simple1.poly input file

```
c
c datagraph
c
p simple_example
n label=a1 color=1
n label=a2 color=1
n label=a3 color=1
n label=a4 color=1
n label=b1 color=2
n label=b2 color=2
n label=b3 color=2
n label=b4 color=2
e 1 2 color=1
e 2 3 color=1
e 3 4 color=1
e 5 6 color=2
e 6 7 color=2
e 7 8 color=2
e 1 5 color=0
```

```
e 2 7 color=0
e 3 6 color=0
e 4 7 color=0
e 4 8 color=0
```

running :

```
java -cp lib/IsoFun.jar helix.graph.program.isofun.Main
samples/simple1.poly
```

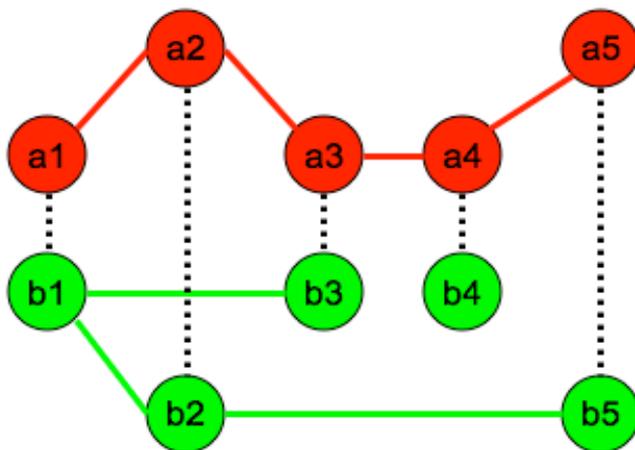
should produce the following output on console :

```
q 5 4 4
v a3 b2 0 0 |  |
v a4 b3 0 0 |  |
v a4 b4 0 0 |  |
v a2 b3 0 0 |  |
v a1 b1 0 0 |  |
```

there is therefore one connecton, composed of 5 spines : (a1,b1), (a2,b3), (a3, b2), (a4,b3) and (a4,b4).

4.3.2example 2:

the following 2-colors datagraph



is described by the samples/simple2.poly input file

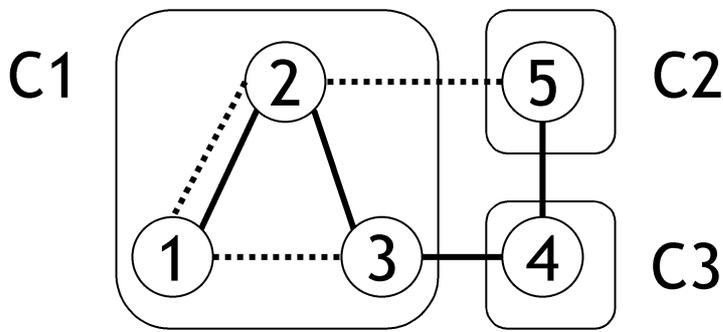
```
c
c datagraph
c
p simple_example
n label=a1 color=1
n label=a2 color=1
n label=a3 color=1
n label=a4 color=1
n label=a5 color=1
```

```

n label=b1 color=2
n label=b2 color=2
n label=b3 color=2
n label=b4 color=2
n label=b5 color=2
e 1 2 color=1
e 2 3 color=1
e 3 4 color=1
e 4 5 color=1
e 6 7 color=2
e 6 8 color=2
e 7 10 color=2
e 1 6 color=0
e 2 7 color=0
e 3 8 color=0
e 4 9 color=0
e 5 10 color=0

```

and corresponds to the following multigraph :



where each multinode 'i' correspond to the pair (ai, bi)

running :

```

java -cp lib/IsoFun.jar helix.graph.program.isofun.Main
samples/simple2.poly

```

should produce the following output on console :

```

q 3 3 3
v a2 b2 0 0 |  |
v a1 b1 0 0 |  |
v a3 b3 0 0 |  |

```

there is therefore one connecton, composed of 3 spines : (a1,b1), (a2,b2) and (a3, b3).

note that small connectons (size 1) are not printed out because of the --minsize and --minelsize options (default is 2). to print them change these options to 1 :

```

java -cp lib/IsoFun.jar helix.graph.program.isofun.Main
samples/simple2.poly --minsize 1 --minelsize 1

```

that should produce :

```

q 3 3 3
v a1 b1 0 0 |  |

```

```
v a3 b3 0 0 | |
v a2 b2 0 0 | |
q 1 1 1
v a4 b4 0 0 | |
q 1 1 1
v a5 b5 0 0 | |
```

i.e. three connectons (two of them are of size 1)

5 Shell scripts

the 'scripts' directory contains a shell script called 'Y3P_Partition'

it is intended for Unix users as an helper to run the java program.
the call syntax is :

```
scripts/Y3P_Partition input_file [isofun_options]
```

This scripts basically do the same thing as running the program manually. The two main differences are :

- the options are specified as '-option value' (instead of '--option value)
- the output is redirected to a new file called input_name.part containing the connectons in a more human readable format.

Hence the previous examples can also be run as :

```
scripts/Y3P_Partition samples/simple1.poly
```

producing the 'simple1.part' file :

```
XX
XX CCC statistics :
XX N = 1
XX min = 5
XX max = 5
XX histogram:
XX t 5 1
XX
XX Elements statistics :
XX N = 1
XX min = 4
XX max = 4
XX histogram:
XX e 4 1
XX
XX Classes sorted by nodeSize :
XX
CL score=5 nodeSize=5 min[eltSize]=4 eltSize : 4 4
VE a1 b1 0 0 | |
VE a4 b4 0 0 | |
VE a3 b2 0 0 | |
VE a2 b3 0 0 | |
VE a4 b3 0 0 | |
XX
```

and

```
scripts/Y3P_Partition samples/simple2.poly -minsize 1 -mineltsize 1
```

producing the 'simple2.part' file :

```
XX
XX CCC statistics :
XX N = 3
XX min = 1
XX max = 3
XX histogram:
XX t 1 2
XX t 3 1
XX
XX Elements statistics :
XX N = 3
XX min = 1
XX max = 3
XX histogram:
XX e 1 2
XX e 3 1
XX
XX Classes sorted by nodeSize :
XX
CL score=3 nodeSize=3 min[eltSize]=3 eltSize : 3 3
VE a3 b3 0 0 |  |
VE a1 b1 0 0 |  |
VE a2 b2 0 0 |  |
XX
CL score=1 nodeSize=1 min[eltSize]=1 eltSize : 1 1
VE a4 b4 0 0 |  |
XX
CL score=1 nodeSize=1 min[eltSize]=1 eltSize : 1 1
VE a5 b5 0 0 |  |
XX
```