YAGS

Yet Another Graph System

Version 0.0.1

16 February 2016

R. MacKinney-Romero
M.A. Pizaña
R. Villarroel-Flores

R. MacKinney-Romero Email: rene@xanum.uam.mx

M.A. Pizaña Email: mpizana@gmail.com
Homepage: http://xamanek.izt.uam.mx/map/

R. Villarroel-Flores Email: rvf0068@gmail.com
Homepage: http://rvf0068.github.io

Copyright

CONTACT INFORMATION:

M.A. Pizaña

`yags@xamanek.izt.uam.mx`

`mpizana@gmail.com`

Departamento de Ingeniería Eléctrica

Universidad Autónoma Metropolitana

Av. San Rafael Atlixco 186.

Col. Vicentina, Del. Iztapalapa

Ciudad de México 09340 MEXICO.

Contents

Chapter 1

Preface

Ejemplo de cita: [1]

1.1 Disclaimer

THIS IS NOT AN OFFICIAL RELEASE YET, this is a version in development. This particular version, 0.0.1, changes from one day to another without warning and even without a change in the version number. Also, the operations and global variables can still change name or even disappear without warning. No commitment is made at the moment concerning compatibility of this version of the software with any future version.

As of this writing (16/Feb/2016) there are only two trustable chapters in this manual: Appendixes 'YAGS Functions by Topic' and 'YAGS Functions Reference'; also the file `cheatsheet-yags.pdf` (within directory: `YAGSDIR/doc/`) may be useful. All other chapters may contain errors, broken links and misleading information (with higher probability).

The first official version will be 0.0.2 and is scheduled to be ready this year (2016), so come back soon.

1.2 Welcome to YAGS

YAGS - *Yet Another Graph System* is a computing system for dealing with graphs, in the sense of Graph Theory (not bar graphs, pie charts nor graphs of functions). Hence our graphs are ordered pairs $G = (V,E)$, where V is a finite set of vertices and E is a finite set of edges which are (ordered or unordered) pairs of vertices.

YAGS was initiated by M.A. Pizaña in May 2003, and soon incorporated the work of R. MacKinney-Romero and R. Villarroel-Flores.

Our motivation here was this and that.

Our Pourposes and Aim.

authors, contacts

1.3 Citing YAGS

If you publish a result and you used YAGS during your research, please cite us as you would normally do with a research paper:

R. MacKinney-Romero, M.A. Pizaña and R. Villarroel-Flores.
YAGS - Yet Another Graph System, Version 0.0.1 (2016)
http://xamanek.izt.uam.mx/yags/

```
@manual{YAGS, author = {R. MacKinney-Romero and M.A. Pizaña and R.
Villarroel-Flores}, title = {YAGS - Yet Another Graph System, Version 0.0.1},
year = {2016}, note = {http://xamanek.izt.uam.mx/yags/}, }
```

1.4 Copyright

CONTACT INFORMATION:

M.A. Pizaña
yags@xamanek.izt.uam.mx
mpizana@gmail.com
Departamento de Ingeniería Eléctrica
Universidad Autónoma Metropolitana
Av. San Rafael Atlixco 186.
Col. Vicentina, Del. Iztapalapa
Ciudad de México 09340 MEXICO.

1.5 Authors

The authors of YAGS in the chronological order of their first contribution are as follows:

M.A. Pizaña
Departamento de Ingeniería Eléctrica
Universidad Autónoma Metropolitana
map@xanum.uam.mx

R. MacKinney-Romero
Departamento de Ingeniería Eléctrica
Universidad Autónoma Metropolitana
rene@xanum.uam.mx

R. Villarroel-Flores

Centro de Investigación en Matemáticas
Universidad Autónoma del Estado de Hidalgo
rafaelv@uaeh.edu.mx

1.6 More Information

More information about YAGS can be found on its official web page:
‘http://xamanek.izt.uam.mx/yags/’

You can receive notifications about YAGS (i.e. new releases, bug fixes, etc.) by subscribing to its email distribution list:
‘http://xamanek.izt.uam.mx/cgi-bin/mailman/listinfo/yagsnews/’

If you are a developer, you may contribute to our project on public repository:
‘https://github.com/yags/main/’

Chapter 2

Getting Started

2.1 What is YAGS?

2.2 Installing YAGS

2.3 Testing the Installation

2.4 A Gentle Tutorial

2.5 An Overview of the Manual

2.6 Cheatsheet

Chapter 3

Cliques

Chapter 4

Graph Categories

4.1 The Default Graph Category

4.2 The Target Graph Category

4.3 Changing the Target Graph Category Temporaryly

4.4 Digraphs, Tournaments, etc.

Chapter 5

Morphisms of Graphs

Chapter 6

Backtracking

6.1 A Simple Example

6.2 How Does it Work?

6.3 Backtracking in Depth

Appendix A

YAGS Functions by Topic

A.1 Most Common Functions

A.2 Drawing

A.3 Constructing Graphs

A.4 Families of Graphs

A.5 Small Graphs

A.6 Attributes and Properties

A.7 Unary Operators

A.8 Binary Operators

A.9 Cliques

Functions dealing with cliques.

- `Basement( G, KnG, x )`
  `Basement( G, KnG, V )`
  Returns the basement of vertex *x* (vertex set *V*) of the iterated clique graph *KnG* with respect to *G*.

- `CliqueGraph( G )`
  `CliqueGraph( G, maxNumCli )`
  Returns the intersection graph of the (maximal) cliques of *G*; aborts if *maxNumCli* cliques are found.

- `CliqueNumber( G )`
  Returns the order, $\omega(G)$, of a maximum clique of *G*.

- `Cliques( `*`G`*` )`
  `Cliques( `*`G`*`, `*`maxNumCli`*` )`
  Returns the list of (maximal) cliques of `G`; aborts if `maxNumCli` cliques are found.

- `CompletesOfGivenOrder( `*`G`*`, `*`Ord`*` )`
  Returns the list of vertex sets of all complete subgraphs of order `Ord` of `G`.

- `IsCliqueGated( `*`G`*` )`
  Returns 'true' if `G` is a clique gated graph.

- `IsCliqueHelly( `*`G`*` )`
  Returns 'true' if the set of (maximal) cliques `G` satisfy the *Helly* property.

- `IsComplete( `*`G`*`, `*`L`*` )`
  Returns 'true' if `L` induces a complete subgraph of `G`.

- `IsCompleteGraph( `*`G`*` )`
  Returns 'true' if graph `G` is a complete graph, 'false' otherwise.

- `NumberOfCliques( `*`G`*` )`
  `NumberOfCliques( `*`G`*`, `*`maxNumCli`*` )`
  Returns the number of (maximal) cliques of `G`.

A.10 Morphisms and Isomorphisms

A.11 Graphs Categories

A.12 Digraphs

A.13 Groups and Rings

A.14 Backtracking

A.15 Miscellaneous

A.16 Undocumented

Appendix B

YAGS Functions Reference

This chapter contains a complete list of all YAGS's functions, with definitions, in alphabetical order.

B.1 Primera seccion

B.1.1 GraphCategory

▷ `GraphCategory([G, ...])` (function)

For internal use. Returns the minimal common category to a list of graphs. If the list of graphs is empty, the default category is returned. The partial order (by inclusion) among graph categories is as follows:

$$SimpleGraphs < `UndirectedGraphs' < `Graphs',$$

$$`OrientedGraphs' < `LooplessGraphs' < `Graphs',$$

$$`SimpleGraphs' < `LooplessGraphs' < `Graphs'$$

Example

```
gap> g1:=CompleteGraph(2:GraphCategory:=SimpleGraphs);
Graph( Category := SimpleGraphs, Order := 2, Size := 1, Adjacencies :=
[ [ 2 ], [ 1 ] ] )
gap> g2:=CompleteGraph(2:GraphCategory:=OrientedGraphs);
Graph( Category := OrientedGraphs, Order := 2, Size := 1, Adjacencies :=
[ [ 2 ], [  ] ] )
gap> g3:=CompleteGraph(2:GraphCategory:=UndirectedGraphs);
Graph( Category := UndirectedGraphs, Order := 2, Size := 3, Adjacencies :=
[ [ 1, 2 ], [ 1, 2 ] ] )
gap> GraphCategory([g1,g2,g3]);
<Operation "Graphs">
gap> GraphCategory([g1,g2]);
<Operation "LooplessGraphs">
gap> GraphCategory([g1,g3]);
<Operation "UndirectedGraphs">
```

B.1.2 Graphs

▷ Graphs (filter)

Graphs is the most general graph category in YAGS. This category contains all graphs that can be represented in YAGS. A graph in this category may contain loops, arrows and edges (which in YAGS are exactly the same as two opposite arrows between some pair of vertices). This graph category has no parent category.

Example

```
gap> GraphByWalks([1,1],[1,2],[2,1],[3,2]:GraphCategory:=Graphs);
Graph( Category := Graphs, Order := 3, Size := 4, Adjacencies :=
[ [ 1, 2 ], [ 1 ], [ 2 ] ] )
gap> GraphByWalks([1,1],[1,2],[2,1],[3,2]:GraphCategory:=SimpleGraphs);
Graph( Category := SimpleGraphs, Order := 3, Size := 2, Adjacencies :=
[ [ 2 ], [ 1, 3 ], [ 2 ] ] )
```

B.1.3 LooplessGraphs

▷ LooplessGraphs (filter)

LooplessGraphs is a graph category in YAGS. A graph in this category may contain arrows and edges but no loops. The parent of this category is Graphs.

Example

```
gap> GraphByWalks([1,1],[1,2],[2,1],[3,2]:GraphCategory:=Graphs);
Graph( Category := Graphs, Order := 3, Size := 4, Adjacencies :=
[ [ 1, 2 ], [ 1 ], [ 2 ] ] )
gap> GraphByWalks([1,1],[1,2],[2,1],[3,2]:GraphCategory:=LooplessGraphs);
Graph( Category := LooplessGraphs, Order := 3, Size := 3, Adjacencies :=
[ [ 2 ], [ 1 ], [ 2 ] ] )
```

B.1.4 Order

▷ Order(*G*) (attribute)

Returns: the number of vertices, of graph *G*.

Example

```
    gap> Order(Icosahedron);
    12
```

B.1.5 OrientedGraphs

▷ OrientedGraphs (filter)

OrientedGraphs is a graph category in YAGS. A graph in this category may contain arrows, but no loops or edges. The parent of this category is LooplessGraphs.

Example

```
gap> GraphByWalks([1,1],[1,2],[2,1],[3,2]:GraphCategory:=Graphs);
Graph( Category := Graphs, Order := 3, Size := 4, Adjacencies :=
[ [ 1, 2 ], [ 1 ], [ 2 ] ] )
```

```
gap> GraphByWalks([1,1],[1,2],[2,1],[3,2]:GraphCategory:=OrientedGraphs);
Graph( Category := OrientedGraphs, Order := 3, Size := 2, Adjacencies :=
[ [ 2 ], [  ], [ 2 ] ] )
```

B.1.6 SetDefaulGraphCategory

▷ SetDefaulGraphCategory(*Catgy*) (function)

Sets the default graph category to *Catgy*. The default graph category is used when constructing new graphs when no other graph category is indicated. New graphs are always forced to comply with the TargetGraphCategory, so loops may be removed, and arrows may replaced by edges or viceversa, depending on the category that the new graph belongs to. The available graph categories are: SimpleGraphs, OrientedGraphs, UndirectedGraphs, LooplessGraphs, and Graphs.

Example
```
gap> SetDefaultGraphCategory(Graphs);
gap> GraphByWalks([1,1],[1,2],[2,1],[3,2]);
Graph( Category := Graphs, Order := 3, Size := 4, Adjacencies :=
[ [ 1, 2 ], [ 1 ], [ 2 ] ] )
gap> SetDefaultGraphCategory(LooplessGraphs);
gap> GraphByWalks([1,1],[1,2],[2,1],[3,2]);
Graph( Category := LooplessGraphs, Order := 3, Size := 3, Adjacencies :=
[ [ 2 ], [ 1 ], [ 2 ] ] )
gap> SetDefaultGraphCategory(UndirectedGraphs);
gap> GraphByWalks([1,1],[1,2],[2,1],[3,2]);
Graph( Category := UndirectedGraphs, Order := 3, Size := 3, Adjacencies :=
[ [ 1, 2 ], [ 1, 3 ], [ 2 ] ] )
gap> SetDefaultGraphCategory(SimpleGraphs);
gap> GraphByWalks([1,1],[1,2],[2,1],[3,2]);
Graph( Category := SimpleGraphs, Order := 3, Size := 2, Adjacencies :=
[ [ 2 ], [ 1, 3 ], [ 2 ] ] )
gap> SetDefaultGraphCategory(OrientedGraphs);
gap> GraphByWalks([1,1],[1,2],[2,1],[3,2]);
Graph( Category := OrientedGraphs, Order := 3, Size := 2, Adjacencies :=
[ [ 2 ], [  ], [ 2 ] ] )
```

B.1.7 SimpleGraphs

▷ SimpleGraphs (filter)

SimpleGraphs is a graph category in YAGS. A graph in this category may contain edges, but no loops or arrows. The category has two parents: LooplessGraphs and UndirectedGraphs.

Example
```
gap> GraphByWalks([1,1],[1,2],[2,1],[3,2]:GraphCategory:=Graphs);
Graph( Category := Graphs, Order := 3, Size := 4, Adjacencies :=
[ [ 1, 2 ], [ 1 ], [ 2 ] ] )
gap> GraphByWalks([1,1],[1,2],[2,1],[3,2]:GraphCategory:=SimpleGraphs);
Graph( Category := SimpleGraphs, Order := 3, Size := 2, Adjacencies :=
[ [ 2 ], [ 1, 3 ], [ 2 ] ] )
```

B.1.8 UndirectedGraphs

▷ `UndirectedGraphs` (filter)

`UndirectedGraphs` is a graph category in YAGS. A graph in this category may contain edges and loops, but no arrows. The parent of this category is `Graphs`.

Example
```
gap> GraphByWalks([1,1],[1,2],[2,1],[3,2]:GraphCategory:=Graphs);
Graph( Category := Graphs, Order := 3, Size := 4, Adjacencies :=
[ [ 1, 2 ], [ 1 ], [ 2 ] ] )
gap> GraphByWalks([1,1],[1,2],[2,1],[3,2]:GraphCategory:=UndirectedGraphs);
Graph( Category := UndirectedGraphs, Order := 3, Size := 3, Adjacencies :=
[ [ 1, 2 ], [ 1, 3 ], [ 2 ] ] )
```

References

[1] F. Larrión, V. Neumann-Lara and M.A. Pizaña. *Clique divergent clockwork graphs and partial orders*. Discrete Appl. Math. **141** (2004) 195–207. 5

Index