# YAGS

## Yet Another Graph System

C. Cedillo[1] R. MacKinney-Romero[2] M.A. Pizaña[3]
I.A. Robles[4] R. Villarroel-Flores[5]

[1,2,3,4]Universidad Autónoma Metropolitana.
[5]Universidad Autónoma del Estado de Hidalgo

# Overview

1. Introduction
2. Graph operations
3. Morphisms of graphs
4. Backtracking

# Introduction

# About YAGS

- YAGS offers a graph-drawing subsystem, methods for dealing with graph homomorphism and a generic backtracking subsystem.
- YAGS was initiated by M.A. Pizaña in May 2003, and then incorporated the work of R. MacKinney-Romero, R. Villarroel-Flores, C. Cedillo and I.A. Robles.

# About YAGS

- YAGS offers a graph-drawing subsystem, methods for dealing with graph homomorphism and a generic backtracking subsystem.
- YAGS was initiated by M.A. Pizaña in May 2003, and then incorporated the work of R. MacKinney-Romero, R. Villarroel-Flores, C. Cedillo and I.A. Robles.
- It is a GAP package: `https://www.gap-system.org/`.

# About YAGS

- YAGS offers a graph-drawing subsystem, methods for dealing with graph homomorphism and a generic backtracking subsystem.
- YAGS was initiated by M.A. Pizaña in May 2003, and then incorporated the work of R. MacKinney-Romero, R. Villarroel-Flores, C. Cedillo and I.A. Robles.
- It is a GAP package: `https://www.gap-system.org/`.
- It is available at: `http://xamanek.izt.uam.mx/yags/`.

# About YAGS

- YAGS offers a graph-drawing subsystem, methods for dealing with graph homomorphism and a generic backtracking subsystem.
- YAGS was initiated by M.A. Pizaña in May 2003, and then incorporated the work of R. MacKinney-Romero, R. Villarroel-Flores, C. Cedillo and I.A. Robles.
- It is a GAP package: `https://www.gap-system.org/`.
- It is available at: `http://xamanek.izt.uam.mx/yags/`.
- It's free! (GNU General Public License 3).

## Load YAGS

```
/home/joe/Yags> gap
    --- some GAP info here ---
gap> RequirePackage("yags");

Loading YAGS - Yet Another Graph System, Version 0.0.3.
Copyright (C) 2016 by the YAGS authors; for details type: ?yags:authors
This is free software under GPLv3; for details type: ?yags:copyright

true
gap>
```

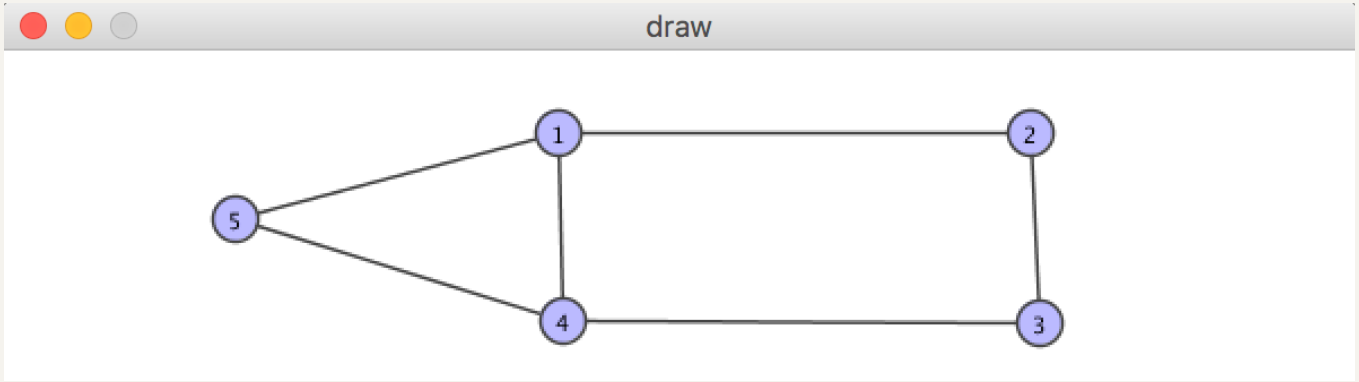
# Define and draw a graph

## Define a graph (by edges)

```
gap>g:=GraphByEdges([[1,2],[2,3],[3,4],[4,1],[1,5],[5,4]]);
```

## Draw a graph

```
gap> Draw(g);
```



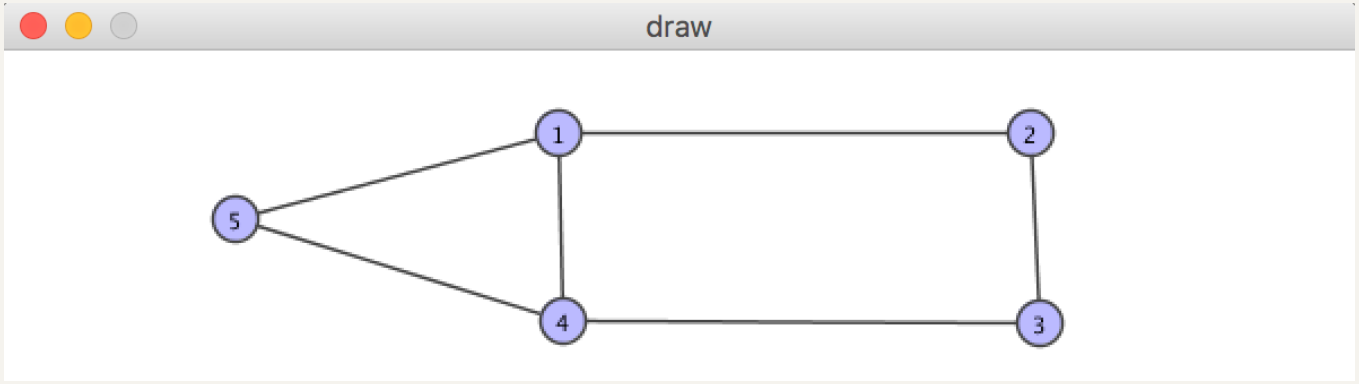
# Define and draw a graph

## Define a graph (by walks)

```
gap> g:=GraphByWalks([1,2,3,4,1],[1,4,5,1]);
```

## Draw a graph

```
gap> Draw(g);
```



# Graph Category

```
gap> g:=GraphByWalks([1,2,3,4,1],[1,4,5,1]);
 Graph( Category := SimpleGraphs, Order := 5, Size :=
6, Adjacencies := [ [ 2, 4, 5 ], [ 1, 3 ], [ 2, 4 ], [
1, 3, 5 ], [ 1, 4 ] ] )
```

# Graph Categories

- Graphs: May contain loops, arrows and edges.
- UndirectedGraphs: Can not contain plain arrows (only edges and loops).
- LooplessGraphs: Can not contain loops (only arrows and edges).
- OrientedGraphs:Can not contain edges nor loops (only arrows).
- SimpleGraphs: Can not contain loops nor arrows (only edges).

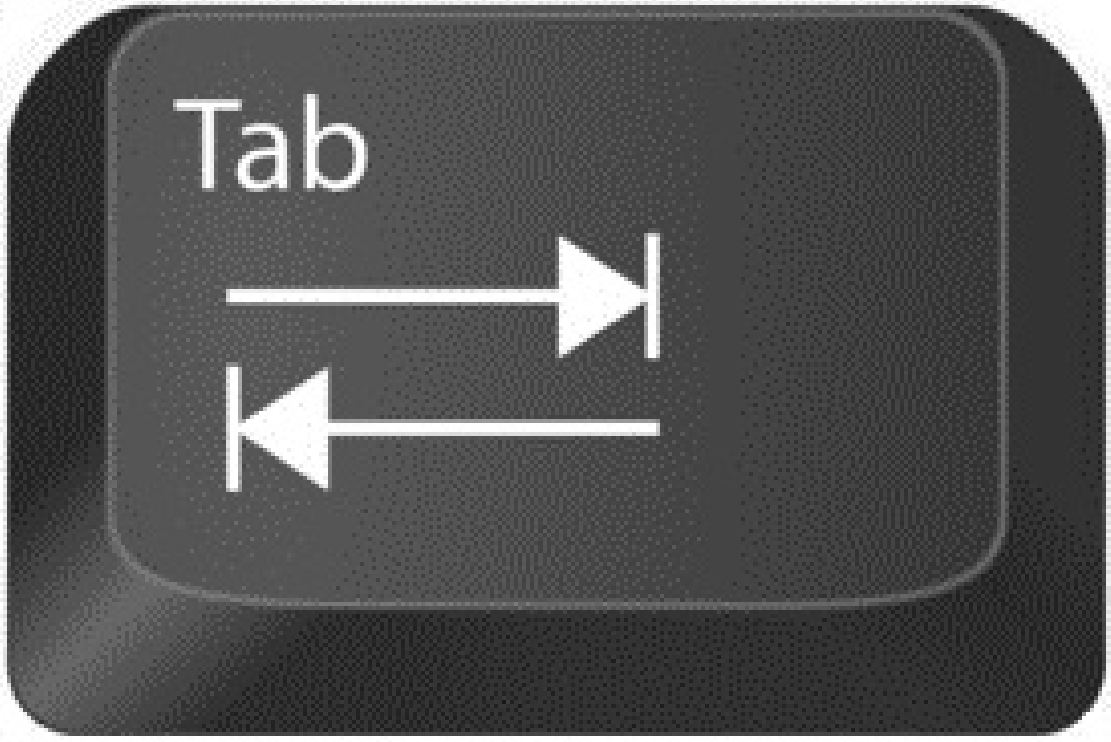
# Graph Category (example)

## SetDefaultGraphCategory

```
gap> SetDefaultGraphCategory(Graphs);
gap> CompleteGraph(4);
 Graph( Category := Graphs, Order := 4, Size := 16,
Adjacencies := [ [ 1, 2, 3, 4 ], [ 1, 2, 3, 4 ], [ 1,
2, 3, 4 ], [ 1, 2, 3, 4 ] ] )
gap>SetDefaultGraphCategory(SimpleGraphs);
gap>CompleteGraph(4);
Graph( Category := SimpleGraphs, Order := 4, Size := 6,
Adjacencies := [ [ 2, 3, 4 ], [ 1, 3, 4 ], [ 1, 2, 4 ],
[ 1, 2, 3 ] ] )
```

# Command completition

TAB key for command completition:



# Command completition and online help

Typing `GraphBy<TAB><TAB>` in the command line shows:

- GraphByAdjMatrix
- GraphByAdjacencies
- GraphByCompleteCover
- GraphByEdges
- GraphByRelation
- GraphByWalks

You can always access the online help by typing:

`?yags:GraphByWalks`

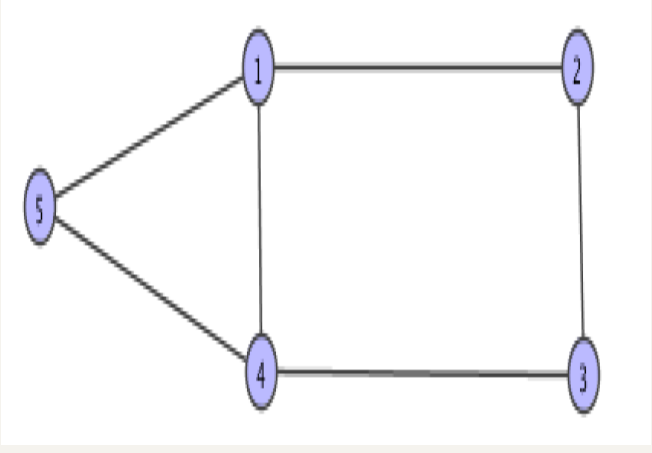
# Graph operations

# Induced subgraph

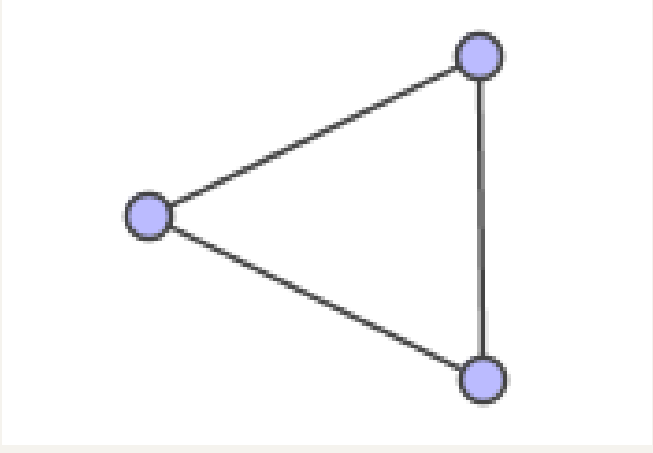
## Define a graph (induced subgraph)

```
gap> h:=InducedSubgraph(g,[1,4,5]);
```

## Draw(g)




## Draw(h)



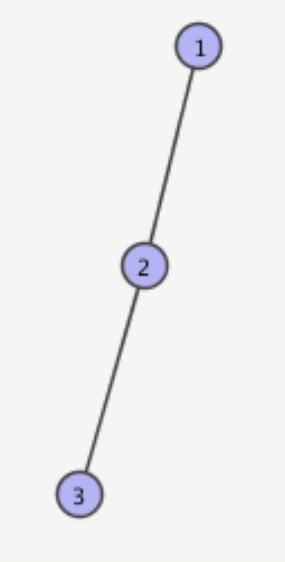
# Intersection Graph

## Define a graph (intersection graph)

```
gap> g:=IntersectionGraph([[1,2,3],[3,4,5],[5,6,7]]);
```

## Draw a graph

```
gap> Draw(g);
```



# Properties of a graph

For the previous graph:

| Command | Result |
|---|---|
| `MaxDegree(g)` | 3 |
| `MinDegree(g)` | 2 |
| `Diameter(g)` | 2 |
| `Girth(g)` | 3 |
| `NumberOfCliques(g)` | 4 |
| `NumberOfConnectedComponents(g)` | 1 |

Also:

`VertexDegree(g,v)`, `Distance(g,x,y)`, `Eccentricity(g,x)`, `Radius(g)`, `InNeigh(G, x)`, ...

# Graph families

| Command | Graph |
|---|---|
| `PathGraph(5)` | $P_5$ |
| `CycleGraph(5)` | $C_5$ |
| `CompleteGraph(5)` | $K_5$ |
| `DiscreteGraph(5)` | $\overline{K_5}$ |
| `CompleteBipartiteGraph(3,3)` | $K_{3,3}$ |
| `OctahedralGraph(5)` | $O_5$ |

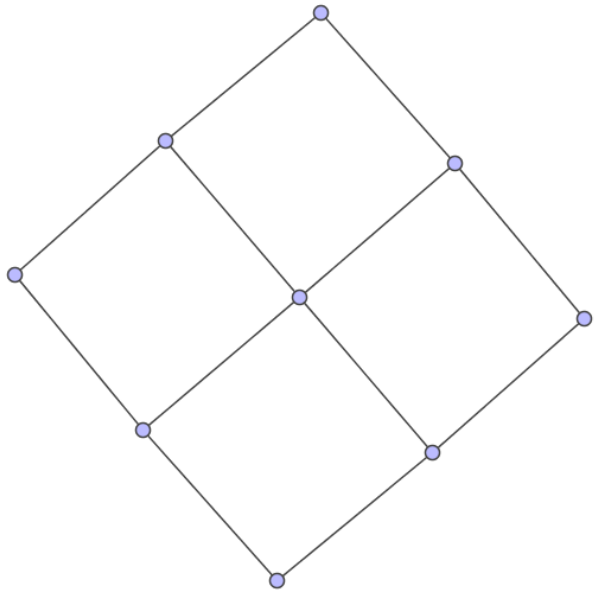
Also:

`Circulant(n,J)`, `CompleteBipartiteGraph(n,m)`, `CubeGraph(n)`, `CompleteMultipartiteGraph(n1,n2[, n3 ...])`, `TorusGraph(n,m)`, `WheelGraph(n)`, `FanGraph(n)`, `SunGraph(n)`, `SpikyGraph(n)`, ...

# Products (Cartesian Product)

## Cartesian product $g \square h$

```
gap> g:=PathGraph(3);
gap> h:=PathGraph(3);
gap> p:=BoxProduct(g,h);
gap> Draw(p);
```



# Products

| Command | Result |
|---|---|
| `TimesProduct(g,h)` | Direct product $g \times h$ |
| `BoxProduct(g,h)` | Cartesian product $g \square h$ |
| `BoxTimesProduct(g,h)` | Strong product $g \boxtimes h$ |
| `Join(g,h)` | Zykov sum $g + h$ |
| `DisjointUnion(g,h)` | Disjoint union $g \cup h$ |

# Graphs of given order

It is possible to return a list of all graphs of order n (up to isomorphism).This operation uses Brendan McKay's data: https://cs.anu.edu.au/people/Brendan.McKay/data/graphs.html.

## Graphs of given order ($n \leq 9$)

```
gap> GraphsOfGivenOrder(2);
 [ Graph( Category := SimpleGraphs, Order := 2, Size :=
0, Adjacencies := [ [ ], [ ] ] ),
Graph( Category := SimpleGraphs, Order := 2, Size :=
1, Adjacencies := [ [ 2 ], [ 1 ] ] ) ]
gap> Length(GraphsOfGivenOrder(9));
274668
```

# Graphs of given order (filtered list)

## Graphs of given order ($n \leq 9$)

```
gap> l:=GraphsOfGivenOrder(6);;
gap> f:=Filtered(l,g->Girth(g)=3);;
```

# Random graphs

## Statistics for random graphs

```
gap> GraphAttributeStatistics(10,1/2, Diameter);
```
$[[2,34],[3,59],[4,5],[5,1],[infinity,1]]$

## Statistics for random graphs (boolean value)

```
gap> GraphAttributeStatistics(10,1/2, IsCliqueHelly);
```
32

```
gap> GraphAttributeStatistics(10,1/10*[1..9], IsCliqueHelly);
```
$[100,100,94,63,34,16,30,76,95]$

# Morphisms of graphs

# Morphisms of Graphs

## Full mono morphisms

```
gap> p3:=PathGraph(3);
gap> c4:=CycleGraph(4);
gap> FullMonoMorphism(p3,c4);
[1,2,3]
gap> FullMonoMorphisms(p3,c4);
[[1,2,3],[1,4,3],[2,1,4],[2,3,4],[3,2,1],
[3,4,1],[4,1,2],[4,3,2]]
```

# Morphism of graphs (example)

## Example

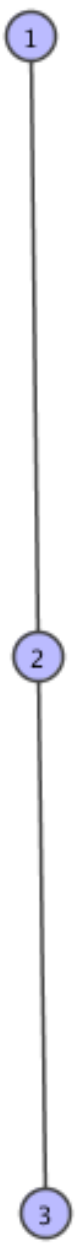
```
gap> IsInducedSubgraph:=function(h,g)
> return FullMonoMorphism(h,g)<>fail; end;
function( h, g ) ... end
gap>IsInducedSubgraph(PathGraph(3),CycleGraph(4));
true
```

# Morphism of graphs (fragments)

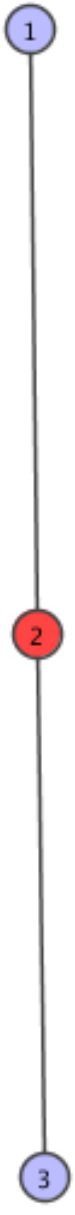
- Morphism: The images of adjacent vertices are adjacent.
- Epi: The morphism is vertex-surjective.
- Mono: The morphism is vertex-injective.
- Bi: The morphism is vertex-bijective.
- Metric: The image of any pair of vertices are at the same distance from each other as the original pair of vertices.
- Full:
  $f(x)f(y) \in E(H) \Rightarrow \exists x'y' \in E(G)$ with $f(x'y') = f(x)f(y)$.

# Backtracking

# Example: proper colorings

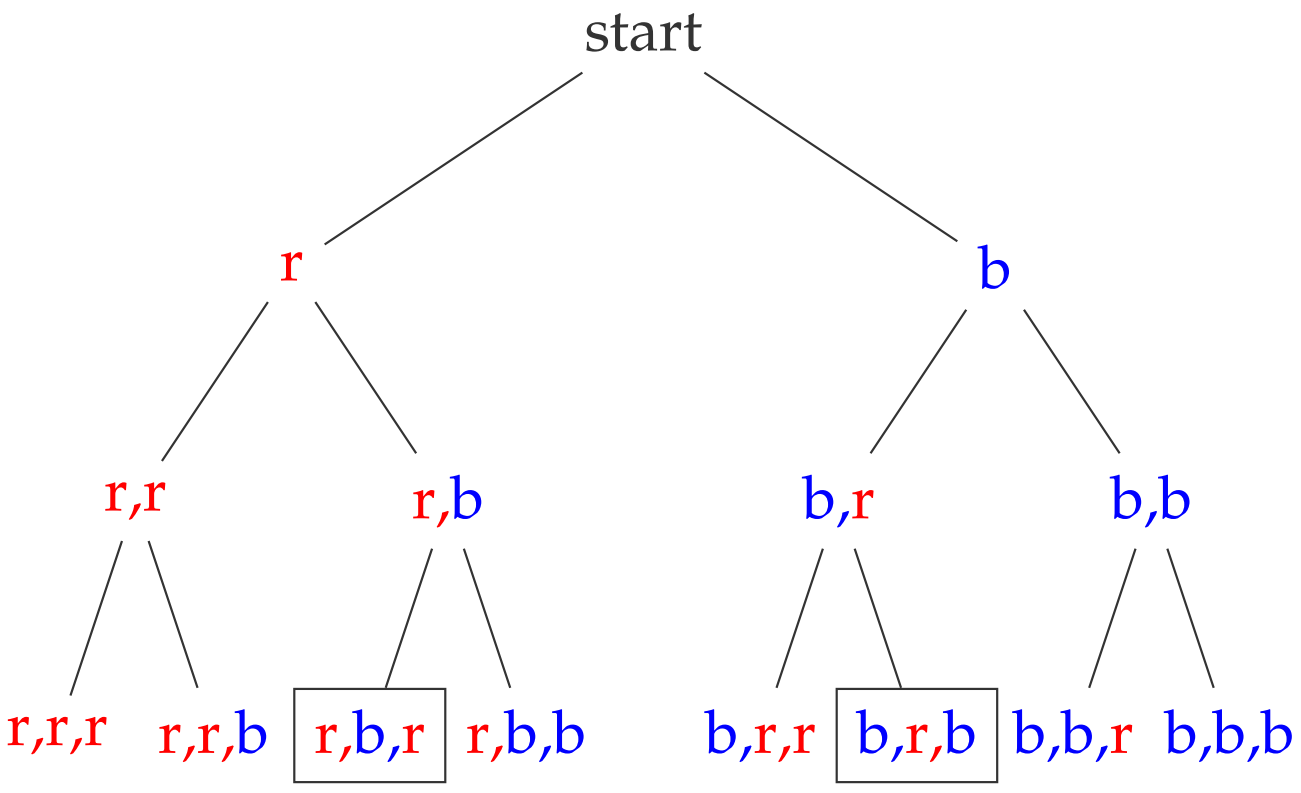






# Example: proper colorings

A coloring is a function $f : \{1, 2, 3\} \Rightarrow \{r, b\}$



# Backtracking (example)

## Example

```
gap> g:=PathGraph(3);;
gap> chk:=function(L,g)
>       local x,y;
>       if L=[] then return true; fi;
>       x:=Length(L);
>       for y in [1..x-1] do
>         if IsEdge(g,[x,y]) and L[x]=L[y] then
>               return false;
>         fi;
>       od;
>       return true;
>     end;
function( L, g ) ... end
gap> BacktrackBag(["r","a"],chk,Order(g),g);
[ [ "r", "b", "r" ], [ "b", "r", "b" ] ]
```

# Questions?