Constructing Graphs by Voltage Assignment

Hebert Pérez-Rosés [1,2]

[1] Department of Mathematics
University of Lleida, Spain

[2] Conjoint fellow
Dept. Software Eng. and Comp. Science
The University *of* Newcastle, Australia

GAP days 2014

Outline

Voltage graphs

- Given a digraph $G = (V, A)$, and a finite group Γ, a **voltage assignment** of G in Γ is a function $\alpha : A \to \Gamma$, that labels the arcs of G with elements of Γ.

- The **derived graph** (**lift**, or **covering**) $G' = (V', A')$ (also denoted G^α), is constructed as follows:
  - $V' = V \times \Gamma$, and $A' = A \times \Gamma$
  - If $a = (v, w) \in A$, then $(a, g) = a_g = (v_g, w_{g\alpha(a)}) \in A'$

Voltage graphs

- Given a digraph $G = (V, A)$, and a finite group Γ, a **voltage assignment** of G in Γ is a function $\alpha : A \to \Gamma$, that labels the arcs of G with elements of Γ.
- The **derived graph** (**lift**, or **covering**) $G' = (V', A')$ (also denoted G^α), is constructed as follows:
  - $V' = V \times \Gamma$, and $A' = A \times \Gamma$
  - If $a = (v, w) \in A$, then $(a, g) = a_g = (v_g, w_{g\alpha(a)}) \in A'$

Voltage graphs

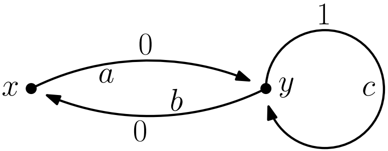
- Given a digraph $G = (V, A)$, and a finite group Γ, a **voltage assignment** of G in Γ is a function $\alpha : A \to \Gamma$, that labels the arcs of G with elements of Γ.

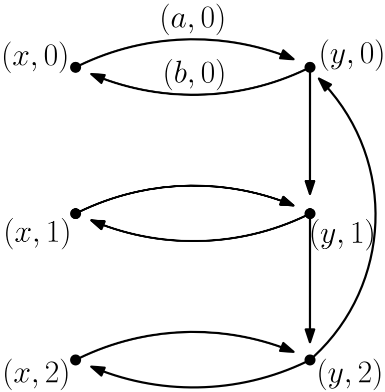
- The **derived graph** (**lift**, or **covering**) $G' = (V', A')$ (also denoted G^α), is constructed as follows:
  - $V' = V \times \Gamma$, and $A' = A \times \Gamma$
  - If $a = (v, w) \in A$, then $(a, g) = a_g = (v_g, w_{g\alpha(a)}) \in A'$

Voltage graphs

- Given a digraph $G = (V, A)$, and a finite group Γ, a **voltage assignment** of G in Γ is a function $\alpha : A \to \Gamma$, that labels the arcs of G with elements of Γ.
- The **derived graph** (**lift**, or **covering**) $G' = (V', A')$ (also denoted G^α), is constructed as follows:
  - $V' = V \times \Gamma$, and $A' = A \times \Gamma$
  - If $a = (v, w) \in A$, then $(a, g) = a_g = (v_g, w_{g\alpha(a)}) \in A'$

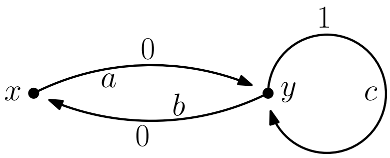
Example 1: Voltages in $\mathbb{Z}_3$

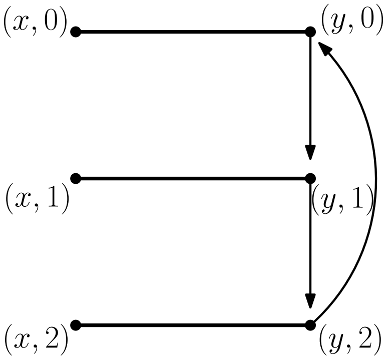




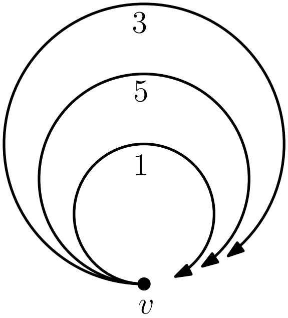
- ▸ Return to GAP session

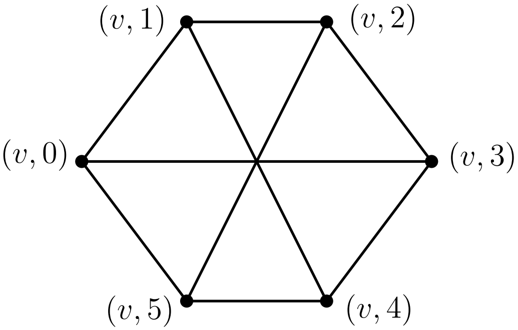
Example 1: Collapsing mutually reverse arcs





Example 2: Cayley graphs (voltages in $\mathbb{Z}_6$)





Some elementary properties

- The **net voltage** of a walk W in G is the product of the voltages of every edge in the W
- Every cycle C' of G' corresponds to a closed non-reversing walk W in G, with net voltage equal to the identity of Γ
- The girth of G' is equal to the length of the shortest closed non-reversing walk W of G, with net voltage equal to the identity
- The **local group** at vertex v is the group generated by the net voltages of all closed walks based at v. G' is connected if and only if the local group at every vertex v is equal to Γ.

Some elementary properties

- The **net voltage** of a walk W in G is the product of the voltages of every edge in the W
- Every cycle C' of G' corresponds to a closed non-reversing walk W in G, with net voltage equal to the identity of Γ
- The girth of G' is equal to the length of the shortest closed non-reversing walk W of G, with net voltage equal to the identity
- The **local group** at vertex v is the group generated by the net voltages of all closed walks based at v. G' is connected if and only if the local group at every vertex v is equal to Γ.

Some elementary properties

- The **net voltage** of a walk W in G is the product of the voltages of every edge in the W
- Every cycle C' of G' corresponds to a closed non-reversing walk W in G, with net voltage equal to the identity of Γ
- The girth of G' is equal to the length of the shortest closed non-reversing walk W of G, with net voltage equal to the identity
- The **local group** at vertex v is the group generated by the net voltages of all closed walks based at v. G' is connected if and only if the local group at every vertex v is equal to Γ.

Some elementary properties

- The **net voltage** of a walk W in G is the product of the voltages of every edge in the W
- Every cycle C' of G' corresponds to a closed non-reversing walk W in G, with net voltage equal to the identity of Γ
- The girth of G' is equal to the length of the shortest closed non-reversing walk W of G, with net voltage equal to the identity
- The **local group** at vertex v is the group generated by the net voltages of all closed walks based at v. G' is connected if and only if the local group at every vertex v is equal to Γ.

The Degree–Diameter Problem (DDP)

- Construct the largest possible network (or graph) with a given maximum degree Δ, and a given diameter D (Elspas, 1964)
- $N_{\Delta,D} \leq 1 + \Delta + \Delta(\Delta - 1) + \cdots + \Delta(\Delta - 1)^{D-1}$ (Moore bound)
- Moore graphs (i.e. graphs attaining the Moore bound) only exist for a few combinations of Δ and D
- Main research directions:
  - Obtain sharper upper bounds
  - Construct larger graphs

The Degree–Diameter Problem (DDP)

- Construct the largest possible network (or graph) with a given maximum degree Δ, and a given diameter D (Elspas, 1964)
- $N_{\Delta,D} \leq 1 + \Delta + \Delta(\Delta-1) + \cdots + \Delta(\Delta-1)^{D-1}$ (Moore bound)
- Moore graphs (i.e. graphs attaining the Moore bound) only exist for a few combinations of Δ and D
- Main research directions:
  - Obtain sharper upper bounds
  - Construct larger graphs

The Degree−Diameter Problem (DDP)

- Construct the largest possible network (or graph) with a given maximum degree Δ, and a given diameter D (Elspas, 1964)
- $N_{\Delta,D} \leq 1 + \Delta + \Delta(\Delta - 1) + \cdots + \Delta(\Delta - 1)^{D-1}$ (Moore bound)
- Moore graphs (i.e. graphs attaining the Moore bound) only exist for a few combinations of Δ and D
- Main research directions:
  - Obtain sharper upper bounds
  - Construct larger graphs

The Degree–Diameter Problem (DDP)

- Construct the largest possible network (or graph) with a given maximum degree Δ, and a given diameter D (Elspas, 1964)
- $N_{\Delta,D} \leq 1 + \Delta + \Delta(\Delta - 1) + \cdots + \Delta(\Delta - 1)^{D-1}$ (Moore bound)
- Moore graphs (i.e. graphs attaining the Moore bound) only exist for a few combinations of Δ and D
- Main research directions:
  - Obtain sharper upper bounds
  - Construct larger graphs

The Degree–Diameter Problem (DDP)

- Construct the largest possible network (or graph) with a given maximum degree Δ, and a given diameter D (Elspas, 1964)
- $N_{\Delta,D} \leq 1 + \Delta + \Delta(\Delta - 1) + \cdots + \Delta(\Delta - 1)^{D-1}$ (Moore bound)
- Moore graphs (i.e. graphs attaining the Moore bound) only exist for a few combinations of Δ and D
- Main research directions:
  - Obtain sharper upper bounds
  - Construct larger graphs

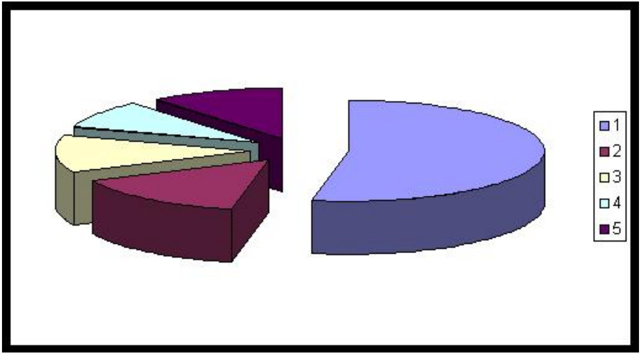
The Degree–Diameter Problem (DDP)

- Construct the largest possible network (or graph) with a given maximum degree Δ, and a given diameter D (Elspas, 1964)
- $N_{\Delta,D} \leq 1 + \Delta + \Delta(\Delta - 1) + \cdots + \Delta(\Delta - 1)^{D-1}$ (Moore bound)
- Moore graphs (i.e. graphs attaining the Moore bound) only exist for a few combinations of Δ and D
- Main research directions:
  - Obtain sharper upper bounds
  - Construct larger graphs

Largest Known Graphs (combinatoricswiki.org)

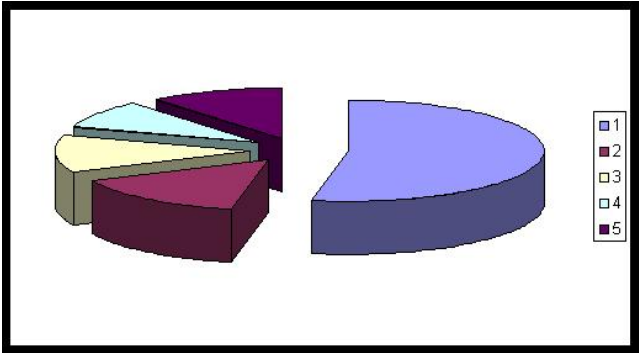
d\k	2	3	4	5	6	7	8	9	10
3	10	20	38	70	132	196	336	600	1 250
4	15	41	96	364	740	1 320	3 243	7 575	17 703
5	24	72	210	624	2 772	5 516	17 030	57 840	187 056
6	32	110	390	1 404	7 917	19 383	76 461	307 845	1 253 615
7	50	168	672	2 756	11 988	52 768	249 660	1 223 050	6 007 230
8	57	253	1 100	5 060	39 672	131 137	734 820	4 243 100	24 897 161
9	74	585	1 550	8 200	75 893	279 616	1 686 600	12 123 288	65 866 350
10	91	650	2 286	13 140	134 690	583 083	4 293 452	27 997 191	201 038 922
11	104	715	3 200	19 500	156 864	1 001 268	7 442 328	72 933 102	600 380 000
12	133	786	4 680	29 470	359 772	1 999 500	15 924 326	158 158 875	1 506 252 500
13	162	851	6 560	40 260	531 440	3 322 080	29 927 790	249 155 760	3 077 200 700
14	183	916	8 200	57 837	816 294	6 200 460	55 913 932	600 123 780	7 041 746 081
15	186	1 215	11 712	76 518	1 417 248	8 599 986	90 001 236	1 171 998 164	10 012 349 898
16	198	1 600	14 640	132 496	1 771 560	14 882 658	140 559 416	2 025 125 476	12 951 451 931
17	274	1 610	19 040	133 144	3 217 872	18 495 162	220 990 700	3 372 648 954	15 317 070 720
18	307	1 620	23 800	171 828	4 022 340	26 515 120	323 037 476	5 768 971 167	16 659 077 632
19	338	1 638	23 970	221 676	4 024 707	39 123 116	501 001 000	8 855 580 344	18 155 097 232
20	381	1 958	34 952	281 820	8 947 848	55 625 185	762 374 779	12 951 451 931	78 186 295 824

Relative contribution of different techniques



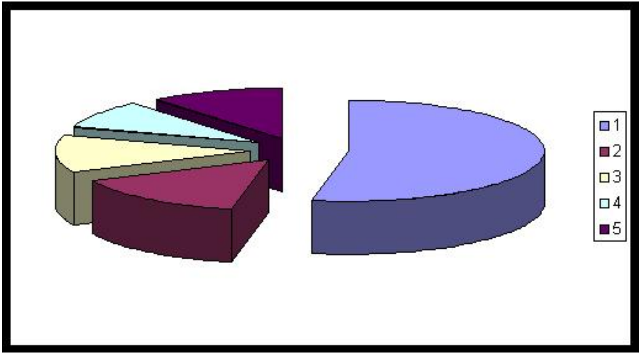
1. Voltage assignment (analytic and computer-based) - 53%
2. Graph compounding - 15%
3. Polarity graphs of generalized polygons - 12%
4. Other computer-based techniques - 9%
5. Moore graphs and others - 11%

Relative contribution of different techniques



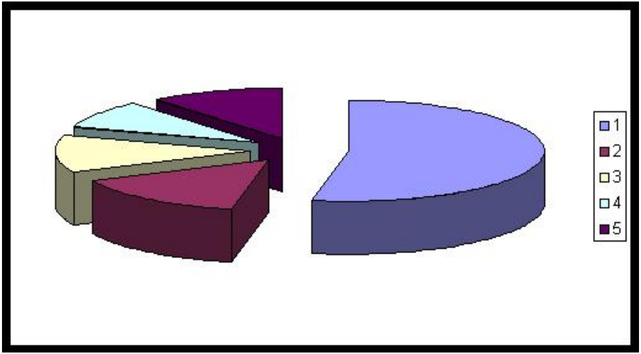
1. Voltage assignment (analytic and computer-based) - 53%
2. Graph compounding - 15%
3. Polarity graphs of generalized polygons - 12%
4. Other computer-based techniques - 9%
5. Moore graphs and others - 11%

Relative contribution of different techniques



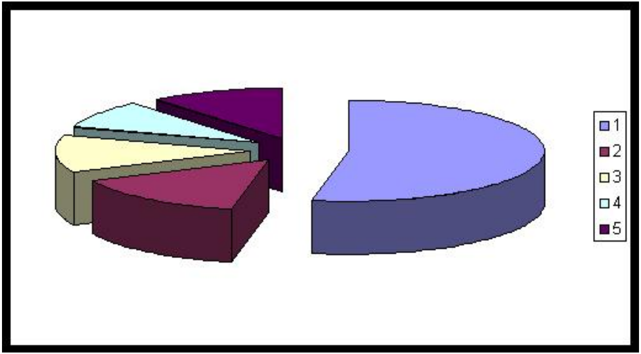
1. Voltage assignment (analytic and computer-based) - 53%
2. Graph compounding - 15%
3. Polarity graphs of generalized polygons - 12%
4. Other computer-based techniques - 9%
5. Moore graphs and others - 11%

Relative contribution of different techniques



1. Voltage assignment (analytic and computer-based) - 53%
2. Graph compounding - 15%
3. Polarity graphs of generalized polygons - 12%
4. Other computer-based techniques - 9%
5. Moore graphs and others - 11%

Relative contribution of different techniques



1. Voltage assignment (analytic and computer-based) - 53%
2. Graph compounding - 15%
3. Polarity graphs of generalized polygons - 12%
4. Other computer-based techniques - 9%
5. Moore graphs and others - 11%

Random voltage search

```
Choose a base graph G and a family of groups Ω,
and initialize MAX;
Label the arcs of a BFS spanning tree of the base
graph G with the identity element;

for every unexplored group Γ in Ω do
    for i:=1 to MAX do begin
        generate a random voltage assignment α;
        compute the girth and diameter of G';
        if diameter ≤ k then begin
            save Γ and α;
            break;
        end;
    end;
```

Construction of large general graphs

- The 'less abelian' a group is, the better
- The groups that have been used more extensively are $\mathbb{Z}_m \rtimes_r \mathbb{Z}_n$, $(\mathbb{Z}_m \times \mathbb{Z}_m) \rtimes_r \mathbb{Z}_n$, and $(\mathbb{Z}_m \rtimes_r \mathbb{Z}_n) \rtimes (\mathbb{Z}_m \rtimes_r \mathbb{Z}_n)$
- The method is expected to give better results with simple groups, and other non-solvable groups (e.g. perfect groups)
- A group Γ is perfect if it equals its commutator (or derived) subgroup $[\Gamma, \Gamma]$. E.g. $SL(2, 5)$

Construction of large general graphs

- The ‘less abelian’ a group is, the better
- The groups that have been used more extensively are $\mathbb{Z}_m \rtimes_r \mathbb{Z}_n$, $(\mathbb{Z}_m \times \mathbb{Z}_m) \rtimes_r \mathbb{Z}_n$, and $(\mathbb{Z}_m \rtimes_r \mathbb{Z}_n) \rtimes (\mathbb{Z}_m \rtimes_r \mathbb{Z}_n)$
- The method is expected to give better results with simple groups, and other non-solvable groups (e.g. perfect groups)
- A group Γ is perfect if it equals its commutator (or derived) subgroup $[\Gamma, \Gamma]$. E.g. $SL(2,5)$

Construction of large general graphs

- The 'less abelian' a group is, the better
- The groups that have been used more extensively are $\mathbb{Z}_m \rtimes_r \mathbb{Z}_n$, $(\mathbb{Z}_m \times \mathbb{Z}_m) \rtimes_r \mathbb{Z}_n$, and $(\mathbb{Z}_m \rtimes_r \mathbb{Z}_n) \rtimes (\mathbb{Z}_m \rtimes_r \mathbb{Z}_n)$
- The method is expected to give better results with simple groups, and other non-solvable groups (e.g. perfect groups)
- A group Γ is perfect if it equals its commutator (or derived) subgroup $[\Gamma, \Gamma]$. E.g. $SL(2, 5)$

Construction of large general graphs

- The 'less abelian' a group is, the better
- The groups that have been used more extensively are $\mathbb{Z}_m \rtimes_r \mathbb{Z}_n$, $(\mathbb{Z}_m \times \mathbb{Z}_m) \rtimes_r \mathbb{Z}_n$, and $(\mathbb{Z}_m \rtimes_r \mathbb{Z}_n) \rtimes (\mathbb{Z}_m \rtimes_r \mathbb{Z}_n)$
- The method is expected to give better results with simple groups, and other non-solvable groups (e.g. perfect groups)
- A group Γ is perfect if it equals its commutator (or derived) subgroup $[\Gamma, \Gamma]$. E.g. $SL(2, 5)$

GAP functionality for graphs

- Package GRAPE, by Leonard Soicher
- Supports directed graphs with loops
- Does not support multiple edges (or multiple loops)
- Does not support edge labels
- Makes heavy use of the automorphism group of the graph
- Calls *nauty*

GAP functionality for graphs

- Package GRAPE, by Leonard Soicher
- Supports directed graphs with loops
- Does not support multiple edges (or multiple loops)
- Does not support edge labels
- Makes heavy use of the automorphism group of the graph
- Calls *nauty*

GAP functionality for graphs

- Package GRAPE, by Leonard Soicher
- Supports directed graphs with loops
- Does not support multiple edges (or multiple loops)
- Does not support edge labels
- Makes heavy use of the automorphism group of the graph
- Calls *nauty*

GAP functionality for graphs

- Package GRAPE, by Leonard Soicher
- Supports directed graphs with loops
- Does not support multiple edges (or multiple loops)
- Does not support edge labels
- Makes heavy use of the automorphism group of the graph
- Calls *nauty*

GAP functionality for graphs

- Package GRAPE, by Leonard Soicher
- Supports directed graphs with loops
- Does not support multiple edges (or multiple loops)
- Does not support edge labels
- Makes heavy use of the automorphism group of the graph
- Calls *nauty*

GAP functionality for graphs

- Package GRAPE, by Leonard Soicher
- Supports directed graphs with loops
- Does not support multiple edges (or multiple loops)
- Does not support edge labels
- Makes heavy use of the automorphism group of the graph
- Calls *nauty*

Our implementation (EULER)

- Graph data structure: Adjacency list
- Group specification
- Functions:
  - Traversals: *BFS* and *DFS*
  - Function *Diameter*, to compute the diameter of a directed graph
  - Function *Lift*, to perform the voltage assignment construction
  - Function *DirectedCayDiameter*: Computes the Cayley digraph of a given group Γ that gives the smallest diameter, among all generating sets with given cardinality k
  - Function *UndirectedCayDiameter*: the same, for undirected Cayley graphs
  - Various input/output functions

Our implementation (EULER)

- Graph data structure: Adjacency list
- Group specification
- Functions:
  - Traversals: *BFS* and *DFS*
  - Function *Diameter*, to compute the diameter of a directed graph
  - Function *Lift*, to perform the voltage assignment construction
  - Function *DirectedCayDiameter*: Computes the Cayley digraph of a given group Γ that gives the smallest diameter, among all generating sets with given cardinality k
  - Function *UndirectedCayDiameter*: the same, for undirected Cayley graphs
  - Various input/output functions

Our implementation (EULER)

- Graph data structure: Adjacency list
- Group specification
- Functions:
  - Traversals: *BFS* and *DFS*
  - Function *Diameter*, to compute the diameter of a directed graph
  - Function *Lift*, to perform the voltage assignment construction
  - Function *DirectedCayDiameter*: Computes the Cayley digraph of a given group Γ that gives the smallest diameter, among all generating sets with given cardinality k
  - Function *UndirectedCayDiameter*: the same, for undirected Cayley graphs
  - Various input/output functions

Our implementation (EULER)

- Graph data structure: Adjacency list
- Group specification
- Functions:
  - Traversals: *BFS* and *DFS*
  - Function *Diameter*, to compute the diameter of a directed graph
  - Function *Lift*, to perform the voltage assignment construction
  - Function *DirectedCayDiameter*: Computes the Cayley digraph of a given group Γ that gives the smallest diameter, among all generating sets with given cardinality k
  - Function *UndirectedCayDiameter*: the same, for undirected Cayley graphs
  - Various input/output functions

Our implementation (EULER)

- Graph data structure: Adjacency list
- Group specification
- Functions:
  - Traversals: *BFS* and *DFS*
  - Function *Diameter*, to compute the diameter of a directed graph
  - Function *Lift*, to perform the voltage assignment construction
  - Function *DirectedCayDiameter*: Computes the Cayley digraph of a given group Γ that gives the smallest diameter, among all generating sets with given cardinality k
  - Function *UndirectedCayDiameter*: the same, for undirected Cayley graphs
  - Various input/output functions

Our implementation (EULER)

- Graph data structure: Adjacency list
- Group specification
- Functions:
  - Traversals: *BFS* and *DFS*
  - Function *Diameter*, to compute the diameter of a directed graph
  - Function *Lift*, to perform the voltage assignment construction
  - Function *DirectedCayDiameter*: Computes the Cayley digraph of a given group Γ that gives the smallest diameter, among all generating sets with given cardinality k
  - Function *UndirectedCayDiameter*: the same, for undirected Cayley graphs
  - Various input/output functions

Our implementation (EULER)

- Graph data structure: Adjacency list
- Group specification
- Functions:
  - Traversals: *BFS* and *DFS*
  - Function *Diameter*, to compute the diameter of a directed graph
  - Function *Lift*, to perform the voltage assignment construction
  - Function *DirectedCayDiameter*: Computes the Cayley digraph of a given group Γ that gives the smallest diameter, among all generating sets with given cardinality k
  - Function *UndirectedCayDiameter*: the same, for undirected Cayley graphs
  - Various input/output functions

Our implementation (EULER)

- Graph data structure: Adjacency list
- Group specification
- Functions:
  - Traversals: *BFS* and *DFS*
  - Function *Diameter*, to compute the diameter of a directed graph
  - Function *Lift*, to perform the voltage assignment construction
  - Function *DirectedCayDiameter*: Computes the Cayley digraph of a given group Γ that gives the smallest diameter, among all generating sets with given cardinality k
  - Function *UndirectedCayDiameter*: the same, for undirected Cayley graphs
  - Various input/output functions

Our implementation (EULER)

- Graph data structure: Adjacency list
- Group specification
- Functions:
  - Traversals: *BFS* and *DFS*
  - Function *Diameter*, to compute the diameter of a directed graph
  - Function *Lift*, to perform the voltage assignment construction
  - Function *DirectedCayDiameter*: Computes the Cayley digraph of a given group Γ that gives the smallest diameter, among all generating sets with given cardinality k
  - Function *UndirectedCayDiameter*: the same, for undirected Cayley graphs
  - Various input/output functions

Our implementation (EULER)

- Graph data structure: Adjacency list
- Group specification
- Functions:
  - Traversals: *BFS* and *DFS*
  - Function *Diameter*, to compute the diameter of a directed graph
  - Function *Lift*, to perform the voltage assignment construction
  - Function *DirectedCayDiameter*: Computes the Cayley digraph of a given group Γ that gives the smallest diameter, among all generating sets with given cardinality k
  - Function *UndirectedCayDiameter*: the same, for undirected Cayley graphs
  - Various input/output functions

Input/output functions

- Euler to/from Matgraph's SGF
- Euler to/from Pajek's .NET
- Euler to/from Gephi
- Other possible formats
  - Sage ?
  - Mathematica ?
  - R ?

Input/output functions

- Euler to/from Matgraph's SGF
- Euler to/from Pajek's .NET
- Euler to/from Gephi
- Other possible formats
  - Sage ?
  - Mathematica ?
  - R ?

- Euler to/from Matgraph's SGF
- Euler to/from Pajek's .NET
- Euler to/from Gephi
- Other possible formats
  - Sage ?
  - Mathematica ?
  - R ?

Input/output functions

- Euler to/from Matgraph's SGF
- Euler to/from Pajek's .NET
- Euler to/from Gephi
- Other possible formats
  - Sage ?
  - Mathematica ?
  - R ?

Input/output functions

- Euler to/from Matgraph's SGF
- Euler to/from Pajek's .NET
- Euler to/from Gephi
- Other possible formats
  - Sage ?
  - Mathematica ?
  - R ?

Input/output functions

- Euler to/from Matgraph's SGF
- Euler to/from Pajek's .NET
- Euler to/from Gephi
- Other possible formats
  - Sage ?
  - Mathematica ?
  - R ?

Input/output functions

- Euler to/from Matgraph's SGF
- Euler to/from Pajek's .NET
- Euler to/from Gephi
- Other possible formats
  - Sage ?
  - Mathematica ?
  - R ?

Input/output functions

- Euler to/from Matgraph's SGF
- Euler to/from Pajek's .NET
- Euler to/from Gephi
- Other possible formats
  - Sage ?
  - Mathematica ?
  - R ?

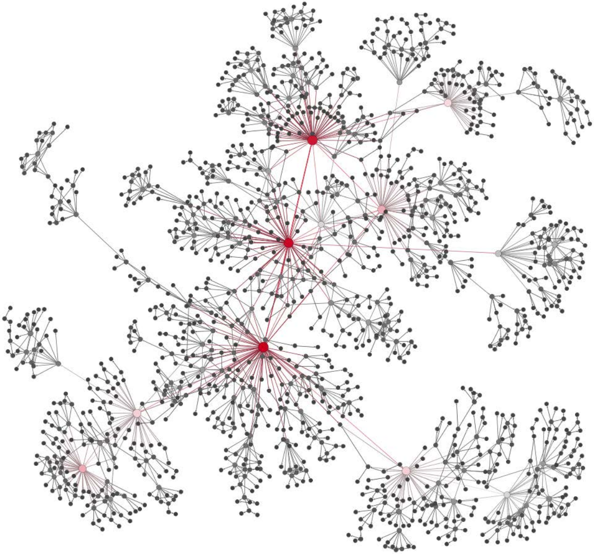
Input/output functions

- Euler to/from Matgraph's SGF
- Euler to/from Pajek's .NET
- Euler to/from Gephi
- Other possible formats
  - Sage ?
  - Mathematica ?
  - R ?

Input/output functions

- Euler to/from Matgraph's SGF
- Euler to/from Pajek's .NET
- Euler to/from Gephi
- Other possible formats
  - Sage ?
  - Mathematica ?
  - R ?

Network layout with Gephi



Example 1

Recall Example 1

```
Gamma:= CyclicGroup(3);
LGamma:= List(Gamma);
vgrafo:= [ [ [ 2 ], [ LGamma[1] ] ],
     [ [ 1, 2 ], [ LGamma[1], LGamma[2] ] ] ];
```

```
     [ [4], [5], [6], [1,5], [2,6], [3,4] ]
```

Future extensions

- Improve data structures
- Functions for:
  - Basic graph operations (girth, connectivity, graph products, etc.)
  - Estimating the diameter of a large directed non-symmetric graph
  - Find the voltage assignment that produces a lift of minimal diameter
  - Implement the random voltage search
- Adapt functions to different group specifications (permutation groups, fp groups, pc groups, etc.)

Future extensions

- Improve data structures
- Functions for:
  - Basic graph operations (girth, connectivity, graph products, etc.)
  - Estimating the diameter of a large directed non-symmetric graph
  - Find the voltage assignment that produces a lift of minimal diameter
  - Implement the random voltage search
- Adapt functions to different group specifications (permutation groups, fp groups, pc groups, etc.)

Future extensions

- Improve data structures
- Functions for:
  - Basic graph operations (girth, connectivity, graph products, etc.)
  - Estimating the diameter of a large directed non-symmetric graph
  - Find the voltage assignment that produces a lift of minimal diameter
  - Implement the random voltage search
- Adapt functions to different group specifications (permutation groups, fp groups, pc groups, etc.)

Future extensions

- Improve data structures
- Functions for:
  - Basic graph operations (girth, connectivity, graph products, etc.)
  - Estimating the diameter of a large directed non-symmetric graph
  - Find the voltage assignment that produces a lift of minimal diameter
  - Implement the random voltage search
- Adapt functions to different group specifications (permutation groups, fp groups, pc groups, etc.)

Future extensions

- Improve data structures
- Functions for:
  - Basic graph operations (girth, connectivity, graph products, etc.)
  - Estimating the diameter of a large directed non-symmetric graph
  - Find the voltage assignment that produces a lift of minimal diameter
  - Implement the random voltage search
- Adapt functions to different group specifications (permutation groups, fp groups, pc groups, etc.)

Future extensions

- Improve data structures
- Functions for:
  - Basic graph operations (girth, connectivity, graph products, etc.)
  - Estimating the diameter of a large directed non-symmetric graph
  - Find the voltage assignment that produces a lift of minimal diameter
  - Implement the random voltage search
- Adapt functions to different group specifications (permutation groups, fp groups, pc groups, etc.)

Future extensions

- Improve data structures
- Functions for:
  - Basic graph operations (girth, connectivity, graph products, etc.)
  - Estimating the diameter of a large directed non-symmetric graph
  - Find the voltage assignment that produces a lift of minimal diameter
  - Implement the random voltage search
- Adapt functions to different group specifications (permutation groups, fp groups, pc groups, etc.)

An alternative data structure

```
Gamma:=  CyclicGroup(3);
LGamma:= List(Gamma);
nverts:= 2;
narcs:= 4;

LArcs:= [ [1,2], [2,1], [2,2] ];

alpha:=  [ LGamma[1], LGamma[1], LGamma[2] ];
```

- Knuth-Bendix completion procedure for string-rewriting systems: Given a finite monoid presentation, convert it to a complete (confluent) presentation.
- Given a complete presentation, multiply two elements (reduced words)
- Compute the order of a finitely-presented group (monoid) given by a confluent presentation
- Generate confluent presentations for some classes of groups (symmetric groups, alternating groups, finite Coxeter groups, etc.)

Auxiliary functions (in GAP 3)

- Knuth-Bendix completion procedure for string-rewriting systems: Given a finite monoid presentation, convert it to a complete (confluent) presentation.
- Given a complete presentation, multiply two elements (reduced words)
- Compute the order of a finitely-presented group (monoid) given by a confluent presentation
- Generate confluent presentations for some classes of groups (symmetric groups, alternating groups, finite Coxeter groups, etc.)

Auxiliary functions (in GAP 3)

- Knuth-Bendix completion procedure for string-rewriting systems: Given a finite monoid presentation, convert it to a complete (confluent) presentation.
- Given a complete presentation, multiply two elements (reduced words)
- Compute the order of a finitely-presented group (monoid) given by a confluent presentation
- Generate confluent presentations for some classes of groups (symmetric groups, alternating groups, finite Coxeter groups, etc.)

Auxiliary functions (in GAP 3)

- Knuth-Bendix completion procedure for string-rewriting systems: Given a finite monoid presentation, convert it to a complete (confluent) presentation.
- Given a complete presentation, multiply two elements (reduced words)
- Compute the order of a finitely-presented group (monoid) given by a confluent presentation
- Generate confluent presentations for some classes of groups (symmetric groups, alternating groups, finite Coxeter groups, etc.)

Complete presentation of A_5

```
a:=AbstractGenerator("a");
b:=AbstractGenerator("b");
c:=AbstractGenerator("c");

Gens:=[a,b,c];

Rels:= [[a^3, IdWord], [b^2, IdWord], [c^2, IdWord],
        [b*a*b, a^2*b*a^2], [b*a^2*b, a*b*a],
        [c*a, a^2*c], [c*b*c, b*c*b],
        [c*b*a*c, b*c*b*a^2], [c*b*a^2*c, b*c*b*a]];
```

GAP-4 translation

```
F:= FreeGroup("a", "b", "c");
a:= F.1;
b:= F.2;
c:= F.3;

IdWord:= One(F);

Rels:= [[a^3, IdWord], [b^2, IdWord], [c^2, IdWord],
        [b*a*b, a^2*b*a^2], [b*a^2*b, a*b*a],
        [c*a, a^2*c], [c*b*c, b*c*b],
        [c*b*a*c, b*c*b*a^2], [c*b*a^2*c, b*c*b*a]];
```

For more details

- J.L.Gross and T.W.Tucker: *Topological Graph Theory.* John Wiley & Sons, 1987.
- L.H. Soicher: “Computing with graphs and groups”. In *Topics in Algebraic Graph Theory* (L.W. Beineke and R.J. Wilson, eds). Cambridge Univ. Press, 2004, pp. 250-266.
- L.H. Soicher: GRAPE Manual. http://www.gap-system.org/Packages/grape.html.

For more details

- J.L.Gross and T.W.Tucker: *Topological Graph Theory.* John Wiley & Sons, 1987.
- L.H. Soicher: "Computing with graphs and groups". In *Topics in Algebraic Graph Theory* (L.W. Beineke and R.J. Wilson, eds). Cambridge Univ. Press, 2004, pp. 250-266.
- L.H. Soicher: GRAPE Manual. http://www.gap-system.org/Packages/grape.html.

For more details

J.L.Gross and T.W.Tucker: *Topological Graph Theory.*
John Wiley & Sons, 1987.

L.H. Soicher: "Computing with graphs and groups".
In *Topics in Algebraic Graph Theory* (L.W. Beineke and R.J. Wilson, eds).
Cambridge Univ. Press, 2004, pp. 250-266.

L.H. Soicher: GRAPE Manual.
`http://www.gap-system.org/Packages/grape.html`.

END