GraphSym

GraphSym Graphs with Symmetries library

Version 0.1

25 May 2024

Rhys J. Evans
Antonio Montero
Primož Potočnik

Rhys J. Evans Email: rhysjevans00@gmail.com
Homepage: https://rhysje00.github.io/

Antonio Montero Email: antonio.montero@fmf.uni-lj.si
Homepage: https://anteromontonio.github.io/

Primož Potočnik Email: primoz.potocnik@fmf.uni-lj.si
Homepage: https://users.fmf.uni-lj.si/potocnik/work.htm

Copyright

Acknowledgements

The GraphSym package is based largely on the I/O functionality found in the Digraphs package (homepage: `https://digraphs.github.io/Digraphs/`). The content of this package has been extracted directly from the project webpage of Primož Potočnik, and is based on the work of himself, Marston Conder, Pablo Spiga, Gabriel Verret, and Steve Wilson.

Contents

Chapter 1

The GraphSym package

This is the manual for the GraphSym package version 0.1.

The GraphSym package contains various collections of graphs with interesting symmetry properties. Each collection of graphs are attained from complete or partial enumerations published in international journals. The papers containing these enumerations are referenced throughout this document, and their authors are included as contributors (unless they are an author of this package). The GraphSym package provides functionality enabling easy access to these graphs, along with several precomputed properties related to many of the graphs stored within.

Currently, all of the functions in this package deal with finite graphs in Digraphs format [DBJM+19], and much of the functionality provided is based on the very nice code found in (**Digraphs: Visualising and IO**).

1.1 Installing GraphSym

To install the GraphSym package, you first need to download or clone the package, found at https://rhysje00.github.io/graphsym/. Once downloaded, there is no need for any extra compiling. If possible, it is recommended that you place the GraphSym directory into the package directory of your GAP distribution. For guidance on how to load GraphSym package if you cannot move the directory directly into your GAP package directory, see Section 1.3.

The GraphSym package requires the following GAP packages:

- GAPDoc [LN19], version 1.6.6 or higher;
- Digraphs [DBJM+19], version 1.6.1 or higher.

Each of the above packages are part of the standard GAP distribution.

1.2 Building the documentation for GraphSym

Once downloaded, you can build all package documentation by running `gap path/to/graphsym/makedoc.g` from the command line. This will build a pdf version of this manual and save it in the directory `path/to/graphsym/doc/`, as well as provide the GAP help viewer with the content of the manual.

1.3 Loading GraphSym

If the GraphSym package has been downloaded and placed in the GAP package directory, it can by loaded at the GAP prompt by typing the following.

Example

```
gap> LoadPackage("graphsym");
true
```

Otherwise, you can follow the methods found in the GAP reference manual, chapter 76. For example, if `path/to/graphsym` is the path to the GraphSym package directory relative to your GAP session, it can be loaded at the GAP prompt by typing the following.

Example

```
gap> SetPackagePath("graphsym","path/to/graphsym");
gap> LoadPackage("graphsym");
true
```

1.4 Citing GraphSym

If you use the GraphSym package in your research, please tell us about it by emailing rhysjevans00@gmail.com. We are interested in any research involving the use of the GraphSym package and might refer to your work in the future. If you wish to refer to the GraphSym package in a published work, please cite GraphSym like a journal article. The following is a BibTeX entry for the current GraphSym version:

bibtex

```
@Manual{cvt,
        author = {Evans, Rhys J. and Montero, Antonio and
                  Poto\v{c}nik, Primo\v{z}},
        key = {cvt},
        title = {{GraphSym -- GRaphs with SYmmetries LIbrary for GAP,
                  Version 0.1}},
        url = {\verb+(https://rhysje00.github.io/graphsym/)+},
        year = {2024}
```

Chapter 2

Cubic vertex–transitive graphs

In this Chapter we give functions for accessing the cubic vertex-transitive graphs stored in this package, and related properties. Currently, this package contains all connected cubic vertex-transitive graphs on up to 1280 vertices. For information and references on these graphs, see [PSV13b].

Let Γ be a simple graph (undirected, loopless, without multiple edges). Then Γ is *cubic* (or *trivalent*, or 3*-valent*) if each vertex of the graph has exactly 3 neighbours.

The graph Γ is *vertex-transitive* if the automorphism group of Γ acts transitively on the vertices of Γ.

2.1 Accessing the cubic vertex–transitive graphs

In this Section we introduce functions for the access to the cubic vertex-transitive graphs stored in the GraphSym package.

2.1.1 CubicVTGraph

▷ `CubicVTGraph(n, i[, data])` (function)

Returns: A digraph.

Given positive integers `n,i`, this function returns the `i`th cubic vertex-transitive graph with `n` vertices available in this package. If there is no such graph, the function returns `fail`.

When the optional argument `data` is specified, it must have value `true` or `false`. If `data=true`, the graph returned by this function will have been assigned the precomputed properties and attributes from `SetCubicVTGraphProps` (2.2.4). If `data=false` or not specified, none of these properties or attributes are given to the resulting graph.

Example

```
gap> CubicVTGraph(50,4);
<immutable symmetric digraph with 50 vertices, 150 edges>
gap> CubicVTGraph(50,10);
fail
```

2.1.2 AllCubicVTGraphs

▷ `AllCubicVTGraphs(`*`n[, data]`*`)` (function)

Returns: A list.

Given a positive integer *n*, this function returns a list containing all cubic vertex-transitive graphs with *n* vertices available in this package. If there are no such graphs, the function returns `fail`.

When the optional argument *data* is specified, it must have value `true` or `false`. If *data*=`true`, the graphs returned by this function will have been assigned the precomputed properties and attributes from `SetCubicVTGraphProps` (2.2.4). If *data*=`false` or not specified, none of these properties or attributes are given to the resulting graphs.

Example

```
gap> gammas:=AllCubicVTGraphs(50,true);;
gap> List(gammas,IsSplitPraegerXuGraph);
[ false, false, false, false, false, false, false, false, false ]
gap> gammas:=AllCubicVTGraphs(100,true);;
gap> ForAny(gammas,IsSplitPraegerXuGraph);
true
```

2.1.3 CubicVTGraphIterator

▷ `CubicVTGraphIterator(`*`n[, data]`*`)` (function)

Returns: A list

Given a positive integer *n*, this function returns an iterator over all cubic vertex-transitive graphs with *n* vertices available in this package. If there are such no graphs, the function returns an empty iterator.

When the optional argument *data* is specified, it must have value `true` or `false`. If *data*=`true`, the graphs returned by this function will have been assigned the precomputed properties and attributes from `SetCubicVTGraphProps` (2.2.4). If *data*=`false` or not specified, none of these properties or attributes are given to the resulting graphs.

Example

```
gap> cnt:=0;; iter:=CubicVTGraphIterator(1152,true);
<iterator>
gap> for gamma in iter do
> if IsSplitPraegerXuGraph(gamma) then cnt:=cnt+1; fi;
> od;
gap> cnt;
6
```

2.2 Properties of the cubic vertex–transitive graphs and library

In this Section we give the functions which give information about the cubic vertex-transitive graphs library, and the properties and attributes of the graphs it contains.

2.2.1 Precomputed attributes of the cubic vertex–transitive graphs

For a given cubic vertex-transitive graph Γ stored in this package, there are several precomputed attributes stored in the GraphSym package. These include the following:

- The diameter of Γ (`DigraphDiameter` (**Digraphs: DigraphDiameter**)).
- The girth of the Γ (`DigraphUndirectedGirth` (**Digraphs: DigraphUndirectedGirth**)).
- The bipartiteness of Γ (`IsBipartiteDigraph` (**Digraphs: IsBipartiteDigraph**)).
- The Cayleyness of Γ (`IsCayleyGraph` (2.2.7)).
- The arc-transitiveness of Γ (`IsArcTransitiveDigraph` (2.2.8)).
- The property of Γ being isomorphic to a a split Praeger-Xu graph (`IsSplitPraegerXuGraph` (2.2.9)).

Now we introduce functions which are used to find information about the library and each of the graphs it stores.

2.2.2 NrCubicVTGraphs

▷ `NrCubicVTGraphs(`*n*`)` (function)
▷ `NumberCubicVTGraphs(`*n*`)` (function)

Returns: An integer.

Given a positive integer *n*, this function returns the number of cubic vertex-transitive graphs with *n* vertices stored in this package.

For any positive integers *n* up to 1280, the current package stores all cubic vertex-transitive graphs with *n* vertices.

Example

```
gap> NrCubicVTGraphs(50);
9
```

2.2.3 CubicVTGraphId

▷ `CubicVTGraphId(`*gamma*`)` (attribute)

Returns: An integer.

Given a digraph *gamma*, if *gamma* is isomorphic to a graph stored in this package, this function returns the index of the graph isomorphic to *gamma*. Otherwise, this function returns `fail`.

The index `i` of a graph `gamma` in this library is the position at which the graph is stored relative to its number of vertices. In particular, if `gamma` has `n` vertices, then `gamma` will be the `i`th entry of `AllCubicVTGraphs(n)` and the `i`th graph found when iterating through `CubicVTGraphIterator(n)`.

Example

```
gap> gamma:=CubicVTGraph(8,2);;
gap> CubicVTGraphId(gamma);
```

```
2
gap> gamma:=CycleDigraph(8);;
gap> CubicVTGraphId(gamma);
fail
```

2.2.4 SetCubicVTGraphProps

▷ SetCubicVTGraphProps(*gamma*) (function)

Returns:

Given a digraph *gamma*, if this graph is isomorphic to a graph stored in this library, this function sets the properties and attributes of *gamma* precomputed in this package. This includes

- All properties and attributes found in Subsection 2.2.1.
- CubicVTGraphId (2.2.3).
- IsCubicGraph (2.2.6).
- IsVertexTransitive (**Digraphs: IsVertexTransitive**).

Example

```
gap> gamma:=CompleteDigraph(4);;
gap> SetCubicVTGraphProps(gamma);
gap> IsArcTransitiveDigraph(gamma);
true
```

2.2.5 SmallCubicVTGraphsInfo

▷ SmallCubicVTGraphsInfo(*n*) (function)

Returns:

Given a positive integer *n*, this function prints information on the cubic vertex-transitive graphs on *n* vertices currently stored in this package. This includes the total number of graphs, the enumeration status of these graphs, and the number of graphs with several properties precomputed and stored in this library.

Example

```
gap> SmallCubicVTGraphsInfo(50);
CVT: There are 9 cubic vertex-transitive graphs on 50 vertices.

Of these 9 graphs, there are;
   - 6 bipartite graphs,
   - 8 Cayley graphs,
   - 1 arc-transitive graphs,
   - 0 split Praeger-Xu graphs,
   - 9 graphs with solvable automorphism group.
```

2.2.6 IsCubicGraph

▷ IsCubicGraph(*gamma*) (property)

Returns: true or false.

Given a digraph *gamma*, this function returns true if *gamma* is a cubic digraph, and otherwise it returns false.

Example

```
gap> IsCubicGraph(CycleDigraph(5));
false
gap> IsCubicGraph(CompleteDigraph(4));
true
```

2.2.7 IsCayleyGraph

▷ IsCayleyGraph(*gamma*) (property)

Returns: true or false.

Given a cubic vertex-transitive graph *gamma* from the GraphSym package such that its properties and attributes have been assigned, this function returns true if *gamma* is a Cayley (di)graph and false otherwise.

The properties and attributes of a cubic vertex-transitive graph that can be found in the GraphSym package can be assigned using the function SetCubicVTGraphProps (2.2.4), or loaded automatically by the functions CubicVTGraph (2.1.1), AllCubicVTGraphs (2.1.2) or CubicVTGraphIterator (2.1.3). This property is not equivalent to IsCayleyDigraph (**Digraphs: IsCayleyDigraph**), as a digraph with this property has associated group and generators.

Example

```
gap> gamma:=CubicVTGraph(50,2,true);;
gap> IsCayleyGraph(gamma);
true
gap> gamma:=CubicVTGraph(50,9,true);;
gap> IsCayleyGraph(gamma);
false
```

2.2.8 IsArcTransitiveDigraph

▷ IsArcTransitiveDigraph(*gamma*) (property)

Returns: true or false.

Given a digraph *gamma*, this function returns true if *gamma* is arc-transitive and false otherwise. This is a synonym for the function IsEdgeTransitive (**Digraphs: IsEdgeTransitive**).

Example

```
gap> gamma:=CubicVTGraph(50,4,true);;
gap> IsArcTransitiveDigraph(gamma);
false
gap> gamma:=CubicVTGraph(50,8,true);;
```

```
gap> IsArcTransitiveDigraph(gamma);
true
```

2.2.9 IsSplitPraegerXuGraph

▷ IsSplitPraegerXuGraph(*gamma*) (property)

Returns: true or false.

Given a cubic vertex-transitive graph *gamma* from the GraphSym package such that its properties and attributes have been assigned, this function returns true if *gamma* is a split Praeger-Xu graph and false otherwise.

The properties and attributes of a cubic vertex-transitive graph that can be found in the GraphSym package can be assigned using the function SetCubicVTGraphProps (2.2.4), or loaded automatically by the functions CubicVTGraph (2.1.1), AllCubicVTGraphs (2.1.2) or CubicVTGraphIterator (2.1.3).

Example

```
gap> gamma:=CubicVTGraph(48,6,true);;
gap> IsSplitPraegerXuGraph(gamma);
false
gap> gamma:=CubicVTGraph(48,7,true);;
gap> IsSplitPraegerXuGraph(gamma);
true
```

Chapter 3

Cubic edge-transitive graphs

In this Chapter we give functions for accessing the cubic edge-transitive graphs stored in this package, and related properties. Currently, this package contains all connected cubic edge-transitive graphs on up to 10000 vertices. For information and references on these graphs, see [CD02] and [CMMP06] (note: these references do not cover the enumeration of graphs on more than 768 vertices, but the authors are currently writing up the extended computations for publishing).

Let Γ be a simple graph (undirected, loopless, without multiple edges). Then Γ is *cubic* (or *trivalent*, or 3*-valent*) if each vertex of the graph has exactly 3 neighbours.

The graph Γ is *edge-transitive* if the automorphism group of Γ acts transitively on the edges of Γ.

A regular edge-transitive graph can be either vertex-transitive or non-vertex-transitive. In the former case the graph is arc-transitive, and in the latter case it is said to be *semisymmetric*. Currently in the GraphSym package, if a user wants access to a cubic edge-transitive graph they must also specify if the graph is arc-transitive or semisymmetric.

3.1 Accessing the cubic edge–transitive graphs

In this Section we introduce functions for the access to the cubic edge-transitive graphs stored in the GraphSym package.

3.1.1 CubicATGraph

▷ `CubicATGraph(n, i[, data])` (function)

Returns: A digraph.

Given positive integers `n,i`, this function returns the `i`th cubic arc-transitive graph with `n` vertices available in this package. If there is no such graph, the function returns `fail`.

When the optional argument `data` is specified, it must have value `true` or `false`. If `data=true`, the graph returned by this function will have been assigned the precomputed properties and attributes from `SetCubicATGraphProps` (3.2.6). If `data=false` or not specified, none of these properties or attributes are given to the resulting graph.

Example

```
gap> CubicATGraph(2048,4);
<immutable symmetric digraph with 2048 vertices, 6144 edges>
gap> CubicATGraph(2048,26);
fail
```

3.1.2 CubicSSGraph

▷ `CubicSSGraph(n, i[, data])` (function)

Returns: A digraph.

Given positive integers `n,i`, this function returns the `i`th cubic semisymmetric graph with `n` vertices available in this package. If there is no such graph, the function returns `fail`.

When the optional argument `data` is specified, it must have value `true` or `false`. If `data=true`, the graph returned by this function will have been assigned the precomputed properties and attributes from `SetCubicSSGraphProps` (3.2.7). If `data=false` or not specified, none of these properties or attributes are given to the resulting graph.

Example

```
gap> CubicSSGraph(4374,10);
<immutable symmetric digraph with 4374 vertices, 13122 edges>
gap> CubicSSGraph(4374,30);
fail
```

3.1.3 AllCubicATGraphs

▷ `AllCubicATGraphs(n[, data])` (function)

Returns: A list.

Given a positive integer `n`, this function returns a list containing all cubic arc-transitive graphs with `n` vertices available in this package. If there are no such graphs, the function returns `fail`.

When the optional argument `data` is specified, it must have value `true` or `false`. If `data=true`, the graphs returned by this function will have been assigned the precomputed properties and attributes from `SetCubicATGraphProps` (3.2.6). If `data=false` or not specified, none of these properties or attributes are given to the resulting graphs.

Example

```
gap> gammas:=AllCubicATGraphs(3072,true);;
gap> Length(gammas);
21
```

3.1.4 AllCubicSSGraphs

▷ `AllCubicSSGraphs(n[, data])` (function)

Returns: A list.

Given a positive integer `n`, this function returns a list containing all cubic semisymmetric graphs with `n` vertices available in this package. If there are no such graphs, the function returns `fail`.

When the optional argument `data` is specified, it must have value `true` or `false`. If `data=true`, the graphs returned by this function will have been assigned the precomputed properties and attributes

from SetCubicSSGraphProps (3.2.7). If *data*=false or not specified, none of these properties or attributes are given to the resulting graphs.

Example

```
gap> gammas:=AllCubicSSGraphs(7680,true);;
gap> Length(gammas);
21
```

3.1.5 CubicATGraphIterator

▷ CubicATGraphIterator(*n*[, *data*]) (function)

Returns: A list

Given a positive integer *n*, this function returns an iterator over all cubic arc-transitive graphs with *n* vertices available in this package. If there are such no graphs, the function returns an empty iterator.

When the optional argument *data* is specified, it must have value true or false. If *data*=true, the graphs returned by this function will have been assigned the precomputed properties and attributes from SetCubicATGraphProps (3.2.6). If *data*=false or not specified, none of these properties or attributes are given to the resulting graphs.

Example

3.1.6 CubicSSGraphIterator

▷ CubicSSGraphIterator(*n*[, *data*]) (function)

Returns: A list

Given a positive integer *n*, this function returns an iterator over all cubic semisymmetric graphs with *n* vertices available in this package. If there are such no graphs, the function returns an empty iterator.

When the optional argument *data* is specified, it must have value true or false. If *data*=true, the graphs returned by this function will have been assigned the precomputed properties and attributes from SetCubicSSGraphProps (3.2.7). If *data*=false or not specified, none of these properties or attributes are given to the resulting graphs.

Example

3.2 Properties of the cubic edge–transitive graphs and library

In this Section we give the functions which give information about the cubic edge-transitive graphs library, and the properties and attributes of the graphs it contains.

3.2.1 Precomputed attributes of the cubic edge–transitive graphs

Currently, there are no precomputed attributes available for the cubic edge-transitive graphs stored in this package. In the near future, we plan to provide several attributes for each of the cubic edge-transitive graphs stored, including diameter, girth and bipartiteness.

Now we introduce functions which are used to find information about the library and each of the graphs it stores.

3.2.2 NrCubicATGraphs

▷ `NrCubicATGraphs(n)` (function)
▷ `NumberCubicATGraphs(n)` (function)

Returns: An integer.

Given a positive integer `n`, this function returns the number of cubic arc-transitive graphs with `n` vertices stored in this package.

For any positive integers n up to 1280, the current package stores all cubic arc-transitive graphs with n vertices.

Example

```
gap> NrCubicATGraphs(8192);
93
```

3.2.3 NrCubicSSGraphs

▷ `NrCubicSSGraphs(n)` (function)
▷ `NumberCubicSSGraphs(n)` (function)

Returns: An integer.

Given a positive integer `n`, this function returns the number of cubic semisymmetric graphs with `n` vertices stored in this package.

For any positive integers n up to 1280, the current package stores all cubic semisymmetric graphs with n vertices.

Example

```
gap> NrCubicSSGraphs(3888);
35
```

3.2.4 CubicATGraphId

▷ `CubicATGraphId(gamma)` (attribute)

Returns: An integer.

Given a digraph `gamma`, if `gamma` is isomorphic to a cubic arc-transitive graph stored in this package, this function returns the index of the graph isomorphic to `gamma`. Otherwise, this function returns `fail`.

The index `i` of a cubic arc-transitive graph `gamma` in this library is the position at which the graph is stored relative to its number of vertices. In particular, if `gamma` has `n` vertices, then `gamma`

will be the ith entry of AllCubicATGraphs(n) and the ith graph found when iterating through CubicATGraphIterator(n).

Example

```
gap> gamma:=CubicATGraph(8192,40);;
gap> CubicATGraphId(gamma);
40
gap> gamma:=PetersenGraph();;
gap> CubicATGraphId(gamma);
1
```

3.2.5 CubicSSGraphId

▷ CubicSSGraphId(*gamma*) (attribute)

Returns: An integer.

Given a digraph *gamma*, if *gamma* is isomorphic to a cubic semisymmetric graph stored in this package, this function returns the index of the graph isomorphic to *gamma*. Otherwise, this function returns fail.

The index i of a cubic semisymmetric graph gamma in this library is the position at which the graph is stored relative to its number of vertices. In particular, if gamma has n vertices, then gamma will be the ith entry of AllCubicSSGraphs(n) and the ith graph found when iterating through CubicSSGraphIterator(n).

Example

```
gap> gamma:=CubicSSGraph(6912,20);;
gap> CubicSSGraphId(gamma);
20
```

3.2.6 SetCubicATGraphProps

▷ SetCubicATGraphProps(*gamma*) (function)

Returns:

Given a digraph *gamma*, if this graph is isomorphic to a cubic arc-transitive graph stored in this library, this function sets the properties and attributes of *gamma* precomputed in this package. This includes

- All properties and attributes found in Subsection 3.2.1.
- CubicATGraphId (3.2.4).
- IsCubicGraph (2.2.6).
- IsVertexTransitive (**Digraphs: IsVertexTransitive**).

Example

```
gap> gamma:=CompleteDigraph(4);;
```

```
gap> SetCubicATGraphProps(gamma);
gap> IsArcTransitiveDigraph(gamma);
true
```

3.2.7 SetCubicSSGraphProps

▷ SetCubicSSGraphProps(*gamma*) (function)

Returns:

Given a digraph *gamma*, if this graph is isomorphic to a cubic semisymmetric graph stored in this library, this function sets the properties and attributes of *gamma* precomputed in this package. This includes

- All properties and attributes found in Subsection 3.2.1.
- CubicSSGraphId (3.2.5).
- IsCubicGraph (2.2.6).
- IsVertexTransitive (**Digraphs: IsVertexTransitive**).

Example

```
gap> gamma:=CubicSSGraph(6912,25);;
gap> SetCubicSSGraphProps(gamma);
gap> IsVertexTransitive(gamma);
false
```

Chapter 4

Arc–transitive 2–valent digraphs

In this Chapter we give functions for accessing the arc-transitive 2-valent asymmetric digraphs stored in this package, and related properties. Currently, this package contains all arc-transitive 2-valent connected asymmetric digraphs on up to 1000 vertices. For information and references about these digraphs, see [PSV13a].

Let Γ be a digraph. Then Γ is *asymmetric* if it has no arcs for which the reverse arc is also in Γ, and is *connected* if the underlying graph of Γ is connected. The digraph Γ is *arc-transitive* if the automorphism group of Γ acts transitively on the arcs of Γ. Further, the digraph Γ is 2*-valent* if each vertex of Γ has 2 in-neighbours and 2 out-neighbours.

4.1 Accessing the arc–transitive 2–valent digraphs

In this Section we introduce functions for the access to the cubic vertex-transitive graphs stored in the GraphSym package.

4.1.1 AT2ValentDigraph

▷ `AT2ValentDigraph(n, i[, data])` (function)

Returns: A digraph.

Given positive integers `n,i`, this function returns the `i`th arc-transitive 2-valent digraph with `n` vertices available in this package. If there is no such graph, the function returns `fail`.

When the optional argument `data` is specified, it must have value `true` or `false`. If `data=true`, the graph returned by this function will have been assigned the precomputed properties and attributes from `SetAT2ValentDigraphProps` (4.2.4). If `data=false` or not specified, none of these properties or attributes are given to the resulting graph.

Example

```
gap> AT2ValentDigraph(300,40);
<immutable digraph with 300 vertices, 600 edges>
gap> AT2ValentDigraph(300,100);
fail
```

4.1.2 AllAT2ValentDigraphs

▷ `AllAT2ValentDigraphs(`*`n[, data]`*`)` (function)

Returns: A list.

Given a positive integer *n*, this function returns a list containing all arc-transitive digraphs with *n* vertices available in this package. If there are no such graphs, the function returns `fail`.

When the optional argument *data* is specified, it must have value `true` or `false`. If *data*=`true`, the graphs returned by this function will have been assigned the precomputed properties and attributes from `SetAT2ValentDigraphProps` (4.2.4). If *data*=`false` or not specified, none of these properties or attributes are given to the resulting graphs.

Example

```
gap> gammas:=AllAT2ValentDigraphs(300,true);;
gap> ForAny(gammas,HasAbelianVertexStabilizer);
true
```

4.1.3 AT2ValentDigraphIterator

▷ `AT2ValentDigraphIterator(`*`n[, data]`*`)` (function)

Returns: A list.

Given a positive integer *n*, this function returns an iterator over all arc-transitive 2-valent digraphs with *n* vertices available in this package. If there are such no graphs, the function returns an empty iterator.

When the optional argument *data* is specified, it must have value `true` or `false`. If *data*=`true`, the graphs returned by this function will have been assigned the precomputed properties and attributes from `SetAT2ValentDigraphProps` (4.2.4). If *data*=`false` or not specified, none of these properties or attributes are given to the resulting graphs.

Example

```
gap> cnt:=0;; iter:=AT2ValentDigraphIterator(100,true);;
gap> for gamma in iter do
> if HasSolvableAutGroup(gamma) then
> cnt:=cnt+1;
> fi;
> od;
gap> cnt;
15
```

4.2 Properties of the arc–transitive 2–valent digraphs and library

In this Section we give the functions which give information about the arc-transitive 2-valent digraph library, and the properties and attributes of the graphs it contains.

4.2.1 Precomputed attributes of the arc–transitive 2–valent digraphs

For a given arc-transitive 2-valent digraph Γ stored in this package, there are several precomputed attributes stored in the GraphSym package. These include the following:

- The self-reverseness of Γ (`IsSelfReverseDigraph` (4.2.5)).
- The index of the reverse graph of Γ (`AT2ValentReverseDigraphId` (4.2.8)).
- The arc-transitiveness of the underlying graph of Γ (`HasATUnderlyingGraph` (4.2.6)).
- The underlying graph of Γ. This is a string giving the position of the underlying graph in one of the collections of graphs found in this package (`NameOfUnderlyingGraph` (4.2.9)).
- The maximum s such that Γ is s-arc-transitive (`MaximumArcTransitivity` (4.2.10)).
- The generalised wreath digraph-ness of Γ (`IsGeneralizedWreathDigraph` (4.2.7)).

Now we introduce functions which are used to find information about the library and each of the graphs it stores.

4.2.2 NrAT2ValentDigraphs

▷ `NrAT2ValentDigraphs(n)` (function)
▷ `NumberAT2ValentDigraphs(n)` (function)

Returns: An integer.

Given a positive integer `n`, this function returns the number of arc-transitive 2-valent digraphs with `n` vertices stored in this package.

For any positive integers n up to 1000, arc-transitive 2-valent connected asymmetric digraphs with n vertices.

Example

```
gap> NrAT2ValentDigraphs(500);
63
```

4.2.3 AT2ValentDigraphId

▷ `AT2ValentDigraphId(gamma)` (attribute)

Returns: An integer.

Given a digraph `gamma`, if `gamma` is isomorphic to a graph stored in this package, this function returns the index of the graph isomorphic to `gamma`. Otherwise, this function returns `fail`.

The index `i` of a graph `gamma` in this library is the position at which the graph is stored relative to its number of vertices. In particular, if `gamma` has `n` vertices, then `gamma` will be the `i`th entry of `AllAT2ValentDigraphs(n)` and the `i`th graph found when iterating through `AT2ValentDigraphIterator(n)`.

Example

```
gap> gamma:=AT2ValentDigraph(100,5);;
gap> AT2ValentDigraphId(gamma);
```

```
5
```

4.2.4 SetAT2ValentDigraphProps

▷ `SetAT2ValentDigraphProps(`*gamma*`)` (function)

Returns:

Given a digraph *gamma*, if this graph is isomorphic to a graph stored in this library, this function sets the properties and attributes of *gamma* precomputed in this package. This includes

- All properties and attributes found in Subsection 4.2.1.
- `AT2ValentDigraphId` (4.2.3).
- `IsArcTransitiveDigraph` (2.2.8).
- `IsVertexTransitive` (**Digraphs: IsVertexTransitive**).
- `DigraphReverse` (**Digraphs: DigraphReverse**), which stores the reverse graph of *gamma* with it's library id as defined in `AT2ValentDigraphId` (4.2.3)).
- `DigraphSymmetricClosure` (**Digraphs: DigraphSymmetricClosure**), which stores the underlying graph of *gamma* as described in `NameOfUnderlyingGraph` (4.2.9).

Example

```
gap> gamma:=AT2ValentDigraph(150,5);;
gap> SetAT2ValentDigraphProps(gamma);
gap> IsGeneralizedWreathDigraph(gamma);
false
```

4.2.5 IsSelfReverseDigraph

▷ `IsSelfReverseDigraph(`*gamma*`)` (property)

Returns: `true` or `false`.

Given a digraph *gamma*, this function returns `true` if *gamma* is isomorphic to the reverse (or reverse) of *gamma*, and otherwise it returns `false` (see `DigraphReverse` (**Digraphs: DigraphReverse**)).

Example

```
gap> gamma:=AT2ValentDigraph(300,1);;
gap> IsSelfReverseDigraph(gamma);
false
gap> gamma:=AT2ValentDigraph(300,20);;
gap> IsSelfReverseDigraph(gamma);
true
```

4.2.6 HasATUnderlyingGraph

▷ `HasATUnderlyingGraph(`*gamma*`)` (property)

Returns: `true` or `false`.

Given a digraph *gamma*, this function returns `true` if the underlying graph of *gamma* is arc-transitive, and otherwise it returns `false` (see `DigraphSymmetricClosure` (**Digraphs: DigraphSymmetricClosure**), `IsEdgeTransitive` (**Digraphs: IsEdgeTransitive**) and `IsArcTransitiveDigraph` (2.2.8)).

Example

```
gap> gamma:=AT2ValentDigraph(300,10);;
gap> HasATUnderlyingGraph(gamma);
false
gap> gamma:=AT2ValentDigraph(300,30,true);;
gap> HasATUnderlyingGraph(gamma);
true
```

4.2.7 IsGeneralizedWreathDigraph

▷ `IsGeneralizedWreathDigraph(`*gamma*`)` (property)

Returns: `true` or `false`.

Given an arc-transitive 2-valent digraph *gamma* from the GraphSym package such that its properties and attributes have been assigned, this function returns `true` if it is isomorphic to a generalized wreath digraph, and otherwise it returns `false`.

The properties and attributes of an arc-transitive 2-valent digraph that can be found in the GraphSym package can be assigned using the function `SetAT2ValentDigraphProps` (4.2.4), or loaded automatically by the functions `AT2ValentDigraph` (4.1.1), `AllAT2ValentDigraphs` (4.1.2) or `AT2ValentDigraphIterator` (4.1.3).

Let n be an integer such that $n > 2$. The *generalized wreath digraph*, $\overrightarrow{W}_n$, has vertex-set $\mathbb{Z}_n \times \mathbb{Z}_2$. Then, two distinct vertices $(i,a),(j,b)$ are adjacent if and only if $j - i \equiv 1 \pmod{n}$.

For more information on the generalized wreath digraphs, see [PSV15].

Example

```
gap> gamma:=AT2ValentDigraph(700,30,true);;
gap> IsGeneralizedWreathDigraph(gamma);
false
gap> gamma:=AT2ValentDigraph(700,43,true);;
gap> IsGeneralizedWreathDigraph(gamma);
true
```

4.2.8 AT2ValentReverseDigraphId

▷ `AT2ValentReverseDigraphId(`*gamma*`)` (attribute)

Returns: An integer.

Given an arc-transitive 2-valent digraph *gamma* from the GraphSym package such that its properties and attributes have been assigned, this function returns the position, `i`, at which the reverse (or reverse) of *gamma* is stored (see `DigraphReverse` (**Digraphs: DigraphReverse**)). In particular, if *gamma* has order n, then `AT2ValentDigraph(`n`,i)` is the reverse graph of *gamma*.

The properties and attributes of an arc-transitive 2-valent digraph that can be found in the GraphSym package can be assigned using the function `SetAT2ValentDigraphProps` (4.2.4), or loaded automatically by the functions `AT2ValentDigraph` (4.1.1), `AllAT2ValentDigraphs` (4.1.2) or `AT2ValentDigraphIterator` (4.1.3).

Example

```
gap> gamma:=AT2ValentDigraph(700,10,true);;
gap> AT2ValentReverseDigraphId(gamma);
9
```

4.2.9 NameOfUnderlyingGraph

▷ `NameOfUnderlyingGraph(`*gamma*`)` (attribute)

Returns: A string.

Given an arc-transitive 2-valent digraph *gamma* from the GraphSym package such that its properties and attributes have been assigned, this function returns the name of the underlying graph of *gamma* (see `DigraphSymmetricClosure` (**Digraphs: DigraphSymmetricClosure**)).

The properties and attributes of an arc-transitive 2-valent digraph that can be found in the GraphSym package can be assigned using the function `SetAT2ValentDigraphProps` (4.2.4), or loaded automatically by the functions `AT2ValentDigraph` (4.1.1), `AllAT2ValentDigraphs` (4.1.2) or `AT2ValentDigraphIterator` (4.1.3).

The string returned will start with 3 or 4 letters, and then contain 2 numbers enclosed within "[" and "]" characters and separated by the character ";". In the following we give the 3 possible strings of letters found in these names, and their meaning:

`"GWD"`
: the underlying graph is a generalized wreath graph.

`"HAT"`
: the underlying graph is half-arc-transitive, and not a generalised wreath graph (see Chapter 6).

`"GHAT"`
: the underlying graph is G-half-arc-transitive for some subgroup of its automorphism group G, but not a half-arc-transitive or generalised wreath graph (see Chapter 5).

The underlying graph corresponding to this string is then the graph in the list found in [PSV15] with the same name as the string, and indexed by the remaining two integers in the string.

Example

```
gap> gamma:=AT2ValentDigraph(700,40,true);;
gap> NameOfUnderlyingGraph(gamma);
"GHAT[700;12]"
```

4.2.10 MaximumArcTransitivity

▷ `MaximumArcTransitivity(`*gamma*`)` (attribute)

Returns: An integer.

Given an arc-transitive 2-valent digraph *gamma* from the GraphSym package such that its properties and attributes have been assigned, this function returns the maximum integer s such that *gamma* is s-arc-transitive.

The properties and attributes of an arc-transitive 2-valent digraph that can be found in the GraphSym package can be assigned using the function `SetAT2ValentDigraphProps` (4.2.4), or loaded automatically by the functions `AT2ValentDigraph` (4.1.1), `AllAT2ValentDigraphs` (4.1.2) or `AT2ValentDigraphIterator` (4.1.3).

An *s-arc* of a digraph Γ is an $(s+1)$-tuple $(v_0, v_1, \ldots, v_s)$ of vertices of Γ, such that (v_{i-1}, v_i) is an arc of Γ for every $i \in \{1, \ldots, s\}$ and $v_{i-1} \neq v_{i+1}$ for every $i \in \{1, \ldots, s-1\}$. A digraph Γ is *s-arc-transitive* if the automorphism group of Γ acts transitively on the set of s-arcs of Γ.

For more information on s-arc-transitive graphs, see [GR01].

Example

```
gap> gamma:=AT2ValentDigraph(748,18,true);;
gap> MaximumArcTransitivity(gamma);
373
```

Chapter 5

G–half–arc–transitive 4–valent graphs

In this Chapter we give functions for accessing the G-half-arc-transitive 4-valent graphs stored in this package, and related properties. Currently, this package contains all G-half-arc-transitive 4-valent graphs on up to 1000 vertices. For information and references on these graphs, see [PSV13a].

Let Γ be a simple graph (undirected, loopless, without multiple edges). Then Γ is *4-valent* (or *tetravalent*) if each vertex of the graph has exactly 4 neighbours.

The graph Γ is *G-half-arc-transitive* if G is a group of automorphisms of Γ which acts transitively on the vertices and edges of Γ, but not on the arcs of Γ.

5.1 Accessing the *G*–half–arc–transitive 4–valent graphs

In this Section we introduce functions for the access to the G-half-arc-transitive 4-valent graphs stored in the GraphSym package.

5.1.1 GHAT4ValentGraph

▷ `GHAT4ValentGraph(n, i[, data])` (function)

Returns: A digraph.

Given positive integers `n,i`, this function returns the `i`th G-half-arc-transitive 4-valent graph with `n` vertices available in this package. If there is no such graph, the function returns `fail`.

When the optional argument `data` is specified, it must have value `true` or `false`. If `data=true`, the graph returned by this function will have been assigned the precomputed properties and attributes from `SetGHAT4ValentGraphProps` (5.2.4). If `data=false` or not specified, none of these properties or attributes are given to the resulting graph.

Example

```
gap> GHAT4ValentGraph(600,20);
<immutable symmetric digraph with 600 vertices, 2400 edges>
gap> GHAT4ValentGraph(600,81);
fail
```

5.1.2 AllGHAT4ValentGraphs

▷ `AllGHAT4ValentGraphs(`*`n[, data]`*`)` (function)

Returns: A list.

Given a positive integer *n*, this function returns a list containing all G-half-arc-transitive 4-valent graphs with *n* vertices available in this package. If there are no such graphs, the function returns `fail`.

When the optional argument *data* is specified, it must have value `true` or `false`. If *data*=`true`, the graphs returned by this function will have been assigned the precomputed properties and attributes from `SetGHAT4ValentGraphProps` (5.2.4). If *data*=`false` or not specified, none of these properties or attributes are given to the resulting graphs.

Example

```
gap> gammas:=AllGHAT4ValentGraphs(600);;
gap> Length(gammas);
80
```

5.1.3 GHAT4ValentGraphIterator

▷ `GHAT4ValentGraphIterator(`*`n[, data]`*`)` (function)

Returns: An iterator.

Given a positive integer *n*, this function returns an iterator over all G-half-arc-transitive 4-valent graphs with *n* vertices available in this package. If there are such no graphs, the function returns an empty iterator.

When the optional argument *data* is specified, it must have value `true` or `false`. If *data*=`true`, the graphs returned by this function will have been assigned the precomputed properties and attributes from `SetGHAT4ValentGraphProps` (5.2.4). If *data*=`false` or not specified, none of these properties or attributes are given to the resulting graphs.

Example

```
gap> cnt:=0;; iter:=GHAT4ValentGraphIterator(600);;
gap> for gamma in iter do
> if HasSolvableAutGroup(gamma) then
> cnt:=cnt+1;
> fi;
> od;
gap> cnt;
57
```

5.2 Properties of the G–half–arc–transitive 4–valent graphs and library

In this Section we give the functions which give information about the G-half-arc-transitive 4-valent graph library, and the properties and attributes of the graphs it contains.

5.2.1 Precomputed attributes of the G–half–arc–transitive 4–valent graphs

For a given G-half-arc-transitive 4-valent graph Γ stored in this package, there are several precomputed attributes stored in the GraphSym package. These include the following:

- Diameter of Γ (`DigraphDiameter` (**Digraphs: DigraphDiameter**)).
- Girth of Γ (`DigraphUndirectedGirth` (**Digraphs: DigraphUndirectedGirth**)).
- Bipartiteness of Γ (`IsBipartiteDigraph` (**Digraphs: IsBipartiteDigraph**)).
- Arc-transitivity of Γ (`IsArcTransitiveDigraph` (2.2.8)).
- The Cayley type of Γ (`CayleyType` (5.2.5)).
- The consistent cycle types of Γ (`ConsistentCycleTypes` (5.2.6)).

Now we introduce functions which are used to find information about the library and each of the graphs it stores.

5.2.2 NrGHAT4ValentGraphs

▷ `NrGHAT4ValentGraphs(`*n*`)` (function)
▷ `NumberGHAT4ValentGraphs(`*n*`)` (function)

Returns: An integer.

Given a positive integer *n*, this function returns the number of G-half-arc-transitive 4-valent graphs with *n* vertices stored in this package.

For any positive integers *n* up to 1000, the current package stores all G-half-arc-transitive 4-valent graphs with *n* vertices.

Example

```
gap> NrGHAT4ValentGraphs(600);
80
```

5.2.3 GHAT4ValentGraphId

▷ `GHAT4ValentGraphId(`*gamma*`)` (attribute)

Returns: An integer.

Given a digraph *gamma*, if *gamma* is isomorphic to a graph stored in this package, this function returns the index of the graph isomorphic to *gamma*. Otherwise, this function returns `fail`.

The index `i` of a graph `gamma` in this library is the position at which the graph is stored relative to its number of vertices. In particular, if `gamma` has `n` vertices, then `gamma` will be the `i`th entry of `AllGHAT4ValentGraphs(n)` and the `i`th graph found when iterating through `GHAT4ValentGraphIterator(n)`.

Example

```
gap> gamma:=GHAT4ValentGraph(768,20);;
gap> GHAT4ValentGraphId(gamma);
20
```

5.2.4 SetGHAT4ValentGraphProps

▷ `SetGHAT4ValentGraphProps(`*gamma*`)` (function)

Returns:

Given a digraph *gamma*, if this graph is isomorphic to a graph stored in this library, this function sets the properties and attributes of *gamma* precomputed in this package. This includes:

- All properties and attributes found in Subsection 5.2.1.
- `GHAT4ValentGraphId` (5.2.3).
- `IsCayleyGraph` (2.2.7).
- `IsVertexTransitive` (**Digraphs: IsVertexTransitive**).

Example

```
gap> gamma:=GHAT4ValentGraph(768,20);;
gap> SetGHAT4ValentGraphProps(gamma);
gap> SizeStabAut(gamma);
4
```

5.2.5 CayleyType

▷ `CayleyType(`*gamma*`)` (attribute)

Returns: A string.

Given a G-half-arc-transitive 4-valent graph *gamma* from the GraphSym package such that its properties and attributes have been assigned, this function returns the Cayley type of *gamma*.

Let Γ be a graph. Then `CayleyType(`Γ`)` takes value as one of the following strings:

`"Circ"`
: if Γ is isomorphic to a circulant graph (a Cayley graph defined on a cyclic group).

`"AbCay"`
: if Γ is isomorphic to a Cayley graph defined on an abelian group, but not isomorphic to a circulant graph.

`"Cay"`
: if Γ is isomorphic to a Cayley graph defined on a nonabelian group, but not isomorphic to a Cayley graph defined on an abelian group.

`"n-Cay"`
: if Γ is not isomorphic to a Cayley graph.

Example

```
gap> gamma:=GHAT4ValentGraph(768,20,true);;
gap> CayleyType(gamma);
"Cay"
```

5.2.6 ConsistentCycleTypes

▷ `ConsistentCycleTypes(`*gamma*`)` (attribute)

Returns: A list.

Given a G-half-arc-transitive 4-valent graph *gamma* from the GraphSym package such that its properties and attributes have been assigned, this function returns the list of consistent cycle types of the graph *gamma*, with exactly one entry for each orbit of consistent cycles under the action of the automorphism group of *gamma*. For a G-consistent cycle `C` in *gamma*, the consistent cycle type of `C` is denoted as a string, starting with length of `C`, and with last character `s` if `C` is G-symmetric, or `c` if `C` is G-chiral.

The properties and attributes of a G-half-arc-transitive 4-valent graph that can be found in the GraphSym package can be assigned using the function `SetGHAT4ValentGraphProps` (5.2.4), or loaded automatically by the functions `GHAT4ValentGraph` (5.1.1), `AllGHAT4ValentGraphs` (5.1.2) or `GHAT4ValentGraphIterator` (5.1.3).

Let Γ be a graph with group of automorphisms G, and cycle C. The cycle C is *G-consistent* if there is an element of G which induces a 1-step rotation on C. A G-consistent cycle C is *G-symmetric* if there is an element of G which reverses the orientation of C. A G-consistent cycle C is *G-chiral* if there is no element of G which reverses the orientation of C.

For information and references on consistent cycles, see [PSV13a].

Example

```
gap> gamma:=GHAT4ValentGraph(768,20,true);;
gap> ConsistentCycleTypes(gamma);
[ "4s", "6s", "12s" ]
```

Chapter 6

Half–arc–transitive 4–valent graphs

In this Chapter we give functions for accessing the half-arc-transitive 4-valent graphs stored in this package, and related properties. Currently, this package contains all half-arc-transitive 4-valent graphs on up to 1000 vertices. For information and references on these graphs, see [PSV13a].

Let Γ be a simple graph (undirected, loopless, without multiple edges). Then Γ is *4-valent* (or *tetravalent*) if each vertex of the graph has exactly 4 neighbours.

The graph Γ is *half-arc-transitive* if the automorphism group of Γ acts transitively on the vertices and edges of Γ, but not on the arcs of Γ.

6.1 Accessing the half–arc–transitive 4–valent graphs

In this Section we introduce functions for the access to the half-arc-transitive 4-valent graphs stored in the GraphSym package.

6.1.1 HAT4ValentGraph

▷ `HAT4ValentGraph(n, i[, data])` (function)

Returns: A digraph.

Given positive integers `n,i`, this function returns the `i`th half-arc-transitive 4-valent graph with `n` vertices available in this package. If there is no such graph, the function returns `fail`.

When the optional argument `data` is specified, it must have value `true` or `false`. If `data=true`, the graph returned by this function will have been assigned the precomputed properties and attributes from `SetHAT4ValentGraphProps` (6.2.4). If `data=false` or not specified, none of these properties or attributes are given to the resulting graph.

Example

```
gap> HAT4ValentGraph(600,20);
<immutable symmetric digraph with 600 vertices, 2400 edges>
gap> HAT4ValentGraph(600,28);
fail
```

6.1.2 AllHAT4ValentGraphs

▷ `AllHAT4ValentGraphs(`*`n[, data]`*`)` (function)

Returns: A list.

Given a positive integer *n*, this function returns a list containing all half-arc-transitive 4-valent graphs with *n* vertices available in this package. If there are no such graphs, the function returns `fail`.

When the optional argument *data* is specified, it must have value `true` or `false`. If *data*=`true`, the graphs returned by this function will have been assigned the precomputed properties and attributes from `SetHAT4ValentGraphProps` (6.2.4). If *data*=`false` or not specified, none of these properties or attributes are given to the resulting graphs.

Example

```
gap> gammas:=AllHAT4ValentGraphs(600);;
gap> Length(gammas);
27
```

6.1.3 HAT4ValentGraphIterator

▷ `HAT4ValentGraphIterator(`*`n[, data]`*`)` (function)

Returns: An iterator.

Given a positive integer *n*, this function returns an iterator over all half-arc-transitive 4-valent graphs with *n* vertices available in this package. If there are such no graphs, the function returns an empty iterator.

When the optional argument *data* is specified, it must have value `true` or `false`. If *data*=`true`, the graphs returned by this function will have been assigned the precomputed properties and attributes from `SetHAT4ValentGraphProps` (6.2.4). If *data*=`false` or not specified, none of these properties or attributes are given to the resulting graphs.

Example

```
gap> cnt:=0;; iter:=HAT4ValentGraphIterator(600);;
gap> for gamma in iter do
> if HasSolvableAutGroup(gamma) then
> cnt:=cnt+1;
> fi;
> od;
gap> cnt;
25
```

6.2 Properties of the half–arc–transitive 4–valent graphs and library

In this Section we give the functions which give information about the half-arc-transitive 4-valent graph library, and the properties and attributes of the graphs it contains.

6.2.1 Precomputed attributes of the half–arc–transitive 4–valent graphs

For a given half-arc-transitive 4-valent graph Γ stored in this package, there are several precomputed attributes stored in the GraphSym package. These include the following:

- Girth of Γ (`DigraphUndirectedGirth` (**Digraphs: DigraphUndirectedGirth**)).
- Bipartiteness of Γ (`IsBipartiteDigraph` (**Digraphs: IsBipartiteDigraph**)).
- Arc-transitivity of Γ (`IsArcTransitiveDigraph` (2.2.8)).
- The Cayleyness of Γ (`IsCayleyGraph` (2.2.7)).
- The length of the shortest consistent cycle in Γ (see `ConsistentCycleTypes` (5.2.6)).
- The length of the longest consistent cycle in Γ (see `ConsistentCycleTypes` (5.2.6)).

Now we introduce functions which are used to find information about the library and each of the graphs it stores.

6.2.2 NrHAT4ValentGraphs

▷ `NrHAT4ValentGraphs(`*n*`)` (function)
▷ `NumberHAT4ValentGraphs(`*n*`)` (function)

Returns: An integer.

Given a positive integer *n*, this function returns the number of half-arc-transitive 4-valent graphs with *n* vertices stored in this package.

For any positive integers *n* up to 1000, the current package stores all half-arc-transitive 4-valent graphs with *n* vertices.

Example

```
gap> NrHAT4ValentGraphs(600);
27
```

6.2.3 HAT4ValentGraphId

▷ `HAT4ValentGraphId(`*gamma*`)` (attribute)

Returns: An integer.

Given a digraph *gamma*, if *gamma* is isomorphic to a graph stored in this package, this function returns the index of the graph isomorphic to *gamma*. Otherwise, this function returns `fail`.

The index `i` of a graph `gamma` in this library is the position at which the graph is stored relative to its number of vertices. In particular, if `gamma` has `n` vertices, then `gamma` will be the `i`th entry of `AllHAT4ValentGraphs(n)` and the `i`th graph found when iterating through `HAT4ValentGraphIterator(n)`.

Example

```
gap> gamma:=HAT4ValentGraph(768,20);;
gap> HAT4ValentGraphId(gamma);
20
```

6.2.4 SetHAT4ValentGraphProps

▷ `SetHAT4ValentGraphProps(`*gamma*`)` (function)

Returns:

Given a digraph *gamma*, if this graph is isomorphic to a graph stored in this library, this function sets the properties and attributes of *gamma* precomputed in this package. This includes

- All properties and attributes found in Subsection 6.2.1.
- `HAT4ValentGraphId` (6.2.3).
- `IsCayleyGraph` (2.2.7).
- `IsVertexTransitive` (**Digraphs: IsVertexTransitive**).

Example

```
gap> gamma:=HAT4ValentGraph(768,20);;
gap> SetHAT4ValentGraphProps(gamma);
gap> SizeStabAut(gamma);
8
```

Chapter 7

Arc–transitive 4–valent graphs

In this Chapter we give functions for accessing the arc-transitive 4-valent graphs stored in this package, and related properties. Currently, this package contains all arc-transitive 4-valent graphs on up to 640 vertices. For information and references about these graphs, see [PSV15] and [PSV13b].

Let Γ be a simple graph (undirected, loopless, without multiple edges). Then Γ is *4-valent* (or *tetravalent*) if each vertex of the graph has exactly 4 neighbours.

The graph Γ is *arc-transitive* if the automorphism group of Γ acts transitively on the arcs of Γ.

7.1 Accessing the arc–transitive 4–valent graphs

In this Section we introduce functions for the access to the arc-transitive 4-valent graphs stored in the GraphSym package.

7.1.1 AT4ValentGraph

▷ `AT4ValentGraph(n, i[, data])` (function)

Returns: A digraph.

Given positive integers `n,i`, this function returns the `i`th arc-transitive 2-valent digraph with `n` vertices available in this package. If there is no such graph, the function returns `fail`.

When the optional argument `data` is specified, it must have value `true` or `false`. If `data=true`, the graph returned by this function will have been assigned the precomputed properties and attributes from `SetAT4ValentGraphProps` (7.2.3). If `data=false` or not specified, none of these properties or attributes are given to the resulting graph.

Example

```
gap> AT4ValentGraph(640,20);
<immutable symmetric digraph with 640 vertices, 2560 edges>
gap> AT4ValentGraph(640,159);
fail
```

7.1.2 AllAT4ValentGraphs

▷ `AllAT4ValentGraphs(`*`n[, data]`*`)` (function)

Returns: A list.

Given a positive integer *n*, this function returns a list containing all arc-transitive digraphs with *n* vertices available in this package. If there are no such graphs, the function returns `fail`.

When the optional argument *data* is specified, it must have value `true` or `false`. If *data*=`true`, the graphs returned by this function will have been assigned the precomputed properties and attributes from `SetAT4ValentGraphProps` (7.2.3). If *data*=`false` or not specified, none of these properties or attributes are given to the resulting graphs.

Example

```
gap> gammas:=AllAT4ValentGraphs(640);;
gap> Length(gammas);
158
```

7.1.3 AT4ValentGraphIterator

▷ `AT4ValentGraphIterator(`*`n[, data]`*`)` (function)

Returns: An iterator.

Given a positive integer *n*, this function returns an iterator over all arc-transitive 2-valent digraphs with *n* vertices available in this package. If there are such no graphs, the function returns an empty iterator.

When the optional argument *data* is specified, it must have value `true` or `false`. If *data*=`true`, the graphs returned by this function will have been assigned the precomputed properties and attributes from `SetAT4ValentGraphProps` (7.2.3). If *data*=`false` or not specified, none of these properties or attributes are given to the resulting graphs.

Example

```
gap> cnt:=0;; iter:=AT4ValentGraphIterator(640);;
gap> for gamma in iter do
> if HasSolvableAutGroup(gamma) then
> cnt:=cnt+1;
> fi;
> od;
gap> cnt;
155
```

7.2 Properties of the arc–transitive 4–valent graphs and library

In this Section we give the functions which give information about the arc-transitive 4-valent graph library, and the properties and attributes of the graphs it contains.

7.2.1 NrAT4ValentGraphs

▷ `NrAT4ValentGraphs(n)` (function)
▷ `NumberAT4ValentGraphs(n)` (function)

Returns: An integer.

Given a positive integer `n`, this function returns the number of arc-transitive 2-valent digraphs with `n` vertices stored in this package.

For any positive integers n up to 1000, the current package stores all arc-transitive 2-valent digraphs with n vertices.

Example

```
gap> NrAT4ValentGraphs(640);
158
```

7.2.2 AT4ValentGraphId

▷ `AT4ValentGraphId(gamma)` (attribute)

Returns: An integer.

Given a digraph `gamma`, if `gamma` is isomorphic to a graph stored in this package, this function returns the index of the graph isomorphic to `gamma`. Otherwise, this function returns `fail`.

The index `i` of a graph `gamma` in this library is the position at which the graph is stored relative to its number of vertices. In particular, if `gamma` has `n` vertices, then `gamma` will be the `i`th entry of `AllAT4ValentGraphs(n)` and the `i`th graph found when iterating through `AT4ValentGraphIterator(n)`.

Example

```
gap> gamma:=CompleteDigraph(5);;
gap> AT4ValentGraphId(gamma);
1
gap> gamma:=AT4ValentGraph(50,2);;
gap> AT4ValentGraphId(gamma);
2
```

7.2.3 SetAT4ValentGraphProps

▷ `SetAT4ValentGraphProps(gamma)` (function)

Returns:

Given a digraph `gamma`, if this graph is isomorphic to a graph stored in this library, this function sets the properties and attributes of `gamma` precomputed in this package. This includes:

- `AT4ValentGraphId` (7.2.2).

Example

```
gap> gamma:=CompleteDigraph(5);;
gap> SetAT4ValentGraphProps(gamma);
```

```
gap> AT4ValentGraphId(gamma);
1
```

Chapter 8

2–arc–transitive 4–valent graphs

In this Chapter we give functions for accessing the 2-arc-transitive 4-valent graphs stored in this package, and related properties. Currently, this package contains all 2-arc-transitive 4-valent graphs on up to 727 vertices, and many more on up to 2000 vertices. For information and references about these graphs, see [Pot09].

Let Γ be a simple graph (undirected, loopless, without multiple edges). Then Γ is *4-valent* (or *tetravalent*) if each vertex of the graph has exactly 4 neighbours.

The graph Γ is *2-arc-transitive* if the automorphism group of Γ acts transitively on the 2-arcs of Γ.

8.1 Accessing the 2–arc–transitive 4–valent graphs

In this Section we introduce functions for the access to the 2-arc-transitive 4-valent graphs stored in the GraphSym package.

8.1.1 2AT4ValentGraph

▷ `2AT4ValentGraph(n, i[, data])` (function)

Returns: A digraph.

Given positive integers `n,i`, this function returns the `i`th arc-transitive 2-valent digraph with `n` vertices available in this package. If there is no such graph, the function returns `fail`.

When the optional argument `data` is specified, it must have value `true` or `false`. If `data=true`, the graph returned by this function will have been assigned the precomputed properties and attributes from `Set2AT4ValentGraphProps` (8.2.3). If `data=false` or not specified, none of these properties or attributes are given to the resulting graph.

Example

```
gap> 2AT4ValentGraph(1920,3);
<immutable symmetric digraph with 1920 vertices, 7680 edges>
gap> 2AT4ValentGraph(1920,12);
fail
```

8.1.2 All2AT4ValentGraphs

▷ `All2AT4ValentGraphs(`*`n[, data]`*`)` (function)

Returns: A list.

Given a positive integer *n*, this function returns a list containing all arc-transitive digraphs with *n* vertices available in this package. If there are no such graphs, the function returns `fail`.

When the optional argument *data* is specified, it must have value `true` or `false`. If *data*=`true`, the graphs returned by this function will have been assigned the precomputed properties and attributes from `Set2AT4ValentGraphProps` (8.2.3). If *data*=`false` or not specified, none of these properties or attributes are given to the resulting graphs.

Example

```
gap> gammas:=All2AT4ValentGraphs(1920);;
gap> Length(gammas);
11
```

8.1.3 2AT4ValentGraphIterator

▷ `2AT4ValentGraphIterator(`*`n[, data]`*`)` (function)

Returns: An iterator.

Given a positive integer *n*, this function returns an iterator over all arc-transitive 2-valent digraphs with *n* vertices available in this package. If there are such no graphs, the function returns an empty iterator.

When the optional argument *data* is specified, it must have value `true` or `false`. If *data*=`true`, the graphs returned by this function will have been assigned the precomputed properties and attributes from `Set2AT4ValentGraphProps` (8.2.3). If *data*=`false` or not specified, none of these properties or attributes are given to the resulting graphs.

Example

```
gap> cnt:=0;; iter:=2AT4ValentGraphIterator(1920);;
gap> for gamma in iter do
> if HasSolvableAutGroup(gamma) then
> cnt:=cnt+1;
> fi;
> od;
gap> cnt;
0
```

8.2 Properties of the 2–arc–transitive 4–valent graphs and library

In this Section we give the functions which give information about the 2-arc-transitive 4-valent graph library, and the properties and attributes of the graphs it contains.

8.2.1 Nr2AT4ValentGraphs

▷ `Nr2AT4ValentGraphs(n)` (function)
▷ `Number2AT4ValentGraphs(n)` (function)

Returns: An integer.

Given a positive integer `n`, this function returns the number of arc-transitive 2-valent digraphs with `n` vertices stored in this package.

For any positive integers n up to 1000, the current package stores all arc-transitive 2-valent digraphs with n vertices.

Example

```
gap> Nr2AT4ValentGraphs(1920);
11
```

8.2.2 2AT4ValentGraphId

▷ `2AT4ValentGraphId(gamma)` (attribute)

Returns: An integer.

Given a digraph `gamma`, if `gamma` is isomorphic to a graph stored in this package, this function returns the index of the graph isomorphic to `gamma`. Otherwise, this function returns `fail`.

The index `i` of a graph `gamma` in this library is the position at which the graph is stored relative to its number of vertices. In particular, if `gamma` has `n` vertices, then `gamma` will be the `i`th entry of `All2AT4ValentGraphs(n)` and the `i`th graph found when iterating through `2AT4ValentGraphIterator(n)`.

Example

```
gap> gamma:=CompleteDigraph(5);;
gap> 2AT4ValentGraphId(gamma);
1
gap> gamma:=2AT4ValentGraph(1920,10);;
gap> 2AT4ValentGraphId(gamma);
10
```

8.2.3 Set2AT4ValentGraphProps

▷ `Set2AT4ValentGraphProps(gamma)` (function)

Returns:

Given a digraph `gamma`, if this graph is isomorphic to a graph stored in this library, this function sets the properties and attributes of `gamma` precomputed in this package. This includes:

- `2AT4ValentGraphId` (8.2.2).

Example

```
gap> gamma:=2AT4ValentGraph(1920,5);;
gap> Set2AT4ValentGraphProps(gamma);
```

```
gap> 2AT4ValentGraphId(gamma);
5
```

Chapter 9

Edge–transitive 4–valent graphs

In this chapter we give functions for accessing the edge-transitive 4-valent graphs stored in this package, and related properties. Currently, this package contains an incomplete list of edge-transitive 4-valent graphs on up to 512 vertices. For information and references about these graphs, see [PW16].

Let Γ be a simple graph (undirected, loopless, without multiple edges). Then Γ is *4-valent* (or *tetravalent*) if each vertex of the graph has exactly 4 neighbours.

The graph Γ is *edge-transitive* if the automorphism group of Γ acts transitively on the edges of Γ.

9.1 Accessing the edge–transitive 4–valent graphs

In this Section we introduce functions for the access to the edge-transitive 4-valent graphs stored in the GraphSym package.

9.1.1 ET4ValentGraph

▷ `ET4ValentGraph(n, i[, data])` (function)

Returns: A digraph.

Given positive integers `n,i`, this function returns the `i`th arc-transitive 2-valent digraph with `n` vertices available in this package. If there is no such graph, the function returns `fail`.

When the optional argument `data` is specified, it must have value `true` or `false`. If `data=true`, the graph returned by this function will have been assigned the precomputed properties and attributes from `SetET4ValentGraphProps` (9.2.3). If `data=false` or not specified, none of these properties or attributes are given to the resulting graph.

Example

```
gap> ET4ValentGraph(512,100);
<immutable symmetric digraph with 512 vertices, 2048 edges>
gap> ET4ValentGraph(512,700);
fail
```

9.1.2 AllET4ValentGraphs

▷ `AllET4ValentGraphs(`*`n[, data]`*`)` (function)

Returns: A list.

Given a positive integer *n*, this function returns a list containing all arc-transitive digraphs with *n* vertices available in this package. If there are no such graphs, the function returns `fail`.

When the optional argument *data* is specified, it must have value `true` or `false`. If *data*=`true`, the graphs returned by this function will have been assigned the precomputed properties and attributes from `SetET4ValentGraphProps` (9.2.3). If *data*=`false` or not specified, none of these properties or attributes are given to the resulting graphs.

Example

```
gap> gammas:=AllET4ValentGraphs(400);;
gap> Length(gammas);
107
```

9.1.3 ET4ValentGraphIterator

▷ `ET4ValentGraphIterator(`*`n[, data]`*`)` (function)

Returns: An iterator.

Given a positive integer *n*, this function returns an iterator over all arc-transitive 2-valent digraphs with *n* vertices available in this package. If there are such no graphs, the function returns an empty iterator.

When the optional argument *data* is specified, it must have value `true` or `false`. If *data*=`true`, the graphs returned by this function will have been assigned the precomputed properties and attributes from `SetET4ValentGraphProps` (9.2.3). If *data*=`false` or not specified, none of these properties or attributes are given to the resulting graphs.

Example

```
gap> cnt:=0;; iter:=ET4ValentGraphIterator(400);;
gap> for gamma in iter do
> if HasSolvableAutGroup(gamma) then
> cnt:=cnt+1;
> fi;
> od;
gap> cnt;
106
```

9.2 Properties of the edge–transitive 4–valent graphs and library

In this Section we give the functions which give information about the edge-transitive 4-valent graph library, and the properties and attributes of the graphs it contains.

9.2.1 NrET4ValentGraphs

▷ NrET4ValentGraphs(*n*) (function)
▷ NumberET4ValentGraphs(*n*) (function)

Returns: An integer.

Given a positive integer *n*, this function returns the number of arc-transitive 2-valent digraphs with *n* vertices stored in this package.

For any positive integers *n* up to 1000, the current package stores all arc-transitive 2-valent digraphs with *n* vertices.

Example

```
gap> NrET4ValentGraphs(400);
107
```

9.2.2 ET4ValentGraphId

▷ ET4ValentGraphId(*gamma*) (attribute)

Returns: An integer.

Given a digraph *gamma*, if *gamma* is isomorphic to a graph stored in this package, this function returns the index of the graph isomorphic to *gamma*. Otherwise, this function returns fail.

The index i of a graph gamma in this library is the position at which the graph is stored relative to its number of vertices. In particular, if gamma has n vertices, then gamma will be the ith entry of AllET4ValentGraphs(n) and the ith graph found when iterating through ET4ValentGraphIterator(n).

Example

```
gap> gamma:=CompleteDigraph(5);;
gap> ET4ValentGraphId(gamma);
1
gap> gamma:=ET4ValentGraph(512,40);;
gap> ET4ValentGraphId(gamma);
40
```

9.2.3 SetET4ValentGraphProps

▷ SetET4ValentGraphProps(*gamma*) (function)

Returns:

Given a digraph *gamma*, if this graph is isomorphic to a graph stored in this library, this function sets the properties and attributes of *gamma* precomputed in this package. This includes:

- ET4ValentGraphId (9.2.2).

Example

```
gap> gamma:=ET4ValentGraph(512,40);;
gap> SetET4ValentGraphProps(gamma);
```

```
gap> ET4ValentGraphId(gamma);
40
```

Chapter 10

Locally arc–transitive $\{3,4\}$–valent graphs

In this chapter we give functions for accessing the locally arc-transitive $\{3,4\}$-valent graphs stored in this package, and related properties.

In this package, we collect bipartite biregular graphs of valence $\{3,4\}$ that are locally G-arc-transitive for some group of automorphisms G with the edge stabiliser acting faithfully on the union of the neighbourhoods of the vertices of the edge. Currently, this package stores all such graphs on up to 350 vertices, and many more on up to 1050. For more information and references about these graphs, see [Pot12].

10.1 Accessing the locally arc–transitive $\{3,4\}$–valent graphs

In this Section we introduce functions for the access to the locally arc-transitive $\{3,4\}$-valent graphs stored in the GraphSym package.

10.1.1 LAT34ValentGraph

▷ `LAT34ValentGraph(n, i[, data])` (function)

Returns: A digraph.

Given positive integers `n,i`, this function returns the `i`th arc-transitive 2-valent digraph with `n` vertices available in this package. If there is no such graph, the function returns `fail`.

When the optional argument `data` is specified, it must have value `true` or `false`. If `data=true`, the graph returned by this function will have been assigned the precomputed properties and attributes from `SetLAT34ValentGraphProps` (10.2.3). If `data=false` or not specified, none of these properties or attributes are given to the resulting graph.

Example

```
gap> LAT34ValentGraph(896,200);
<immutable symmetric digraph with 896 vertices, 3072 edges>
gap> LAT34ValentGraph(896,300);
fail
```

10.1.2 AllLAT34ValentGraphs

▷ `AllLAT34ValentGraphs(`*`n[, data]`*`)` (function)

Returns: A list.

Given a positive integer *n*, this function returns a list containing all arc-transitive digraphs with *n* vertices available in this package. If there are no such graphs, the function returns `fail`.

When the optional argument *data* is specified, it must have value `true` or `false`. If *data*=`true`, the graphs returned by this function will have been assigned the precomputed properties and attributes from `SetLAT34ValentGraphProps` (10.2.3). If *data*=`false` or not specified, none of these properties or attributes are given to the resulting graphs.

Example

```
gap> gammas:=AllLAT34ValentGraphs(224);;
gap> Length(gammas);
22
```

10.1.3 LAT34ValentGraphIterator

▷ `LAT34ValentGraphIterator(`*`n[, data]`*`)` (function)

Returns: An iterator.

Given a positive integer *n*, this function returns an iterator over all arc-transitive 2-valent digraphs with *n* vertices available in this package. If there are such no graphs, the function returns an empty iterator.

When the optional argument *data* is specified, it must have value `true` or `false`. If *data*=`true`, the graphs returned by this function will have been assigned the precomputed properties and attributes from `SetLAT34ValentGraphProps` (10.2.3). If *data*=`false` or not specified, none of these properties or attributes are given to the resulting graphs.

Example

```
gap> cnt:=0;; iter:=LAT34ValentGraphIterator(224);;
gap> for gamma in iter do
> if HasSolvableAutGroup(gamma) then
> cnt:=cnt+1;
> fi;
> od;
gap> cnt;
22
```

10.2 Properties of the locally arc–transitive $\{3,4\}$–valent graphs and library

In this Section we give the functions which give information about the locally arc-transitive $\{3,4\}$-valent graph library, and the properties and attributes of the graphs it contains.

10.2.1 NrLAT34ValentGraphs

▷ `NrLAT34ValentGraphs(n)` (function)
▷ `NumberLAT34ValentGraphs(n)` (function)

Returns: An integer.

Given a positive integer `n`, this function returns the number of arc-transitive 2-valent digraphs with `n` vertices stored in this package.

For any positive integers *n* up to 1000, the current package stores all arc-transitive 2-valent digraphs with *n* vertices.

Example

```
gap> NrLAT34ValentGraphs(896);
249
```

10.2.2 LAT34ValentGraphId

▷ `LAT34ValentGraphId(gamma)` (attribute)

Returns: An integer.

Given a digraph `gamma`, if `gamma` is isomorphic to a graph stored in this package, this function returns the index of the graph isomorphic to `gamma`. Otherwise, this function returns `fail`.

The index `i` of a graph `gamma` in this library is the position at which the graph is stored relative to its number of vertices. In particular, if `gamma` has `n` vertices, then `gamma` will be the `i`th entry of `AllLAT34ValentGraphs(n)` and the `i`th graph found when iterating through `LAT34ValentGraphIterator(n)`.

Example

```
gap> gamma:=LAT34ValentGraph(896,20);;
gap> LAT34ValentGraphId(gamma);
20
```

10.2.3 SetLAT34ValentGraphProps

▷ `SetLAT34ValentGraphProps(gamma)` (function)

Returns:

Given a digraph `gamma`, if this graph is isomorphic to a graph stored in this library, this function sets the properties and attributes of `gamma` precomputed in this package. This includes:

- `LAT34ValentGraphId` (10.2.2).

Example

```
gap> gamma:=LAT34ValentGraph(896,20);;
gap> SetLAT34ValentGraphProps(gamma);
gap> LAT34ValentGraphId(gamma);
20
```

References

[CD02] M. Conder and P. Dobcsányi. Trivalent symmetric graphs on up to 768 vertices. *JCMCC. The Journal of Combinatorial Mathematics and Combinatorial Computing*, 40:41–63, 2002. 13

[CMMP06] M. Conder, A. Malnič, D. Marušič, and P. Potočnik. A census of semisymmetric cubic graphs on up to 768 vertices. *Journal of Algebraic Combinatorics*, 23(3):255–294, 2006. 13

[DBJM+19] J. De Beule, J. Jonušas, J. D. Mitchell, M. C. Torpey, and W. A. Wilson. Digraphs – a GAP package, Version 0.15.3, 2019. Refereed GAP package, available at https://gap-packages.github.io/Digraphs/. 5

[GR01] Chris Godsil and Gordon Royle. *Algebraic Graph Theory*, volume 207 of *Graduate Texts in Mathematics*. Springer, 2001. 25

[LN19] F. Lübeck and M. Neunhöffer. GAPDoc – a GAP package, Version 1.6.1, 2019. Refereed GAP package, available at http://www.math.rwth-aachen.de/ Frank.Luebeck/GAPDoc/index.html. 5

[Pot09] Primož Potočnik. A list of 4-valent 2-arc-transitive graphs and finite faithful amalgams of index (4, 2). *European Journal of Combinatorics*, 30(5):1323–1336, 2009. Part Special Issue on Metric Graph Theory. 39

[Pot12] Primož Potočnik. Locally arc-transitive graphs of valence $\{3,4\}$ with trivial edge kernel. *Journal of Algebraic Combinatorics*, 38, 2012. 47

[PSV13a] Primož Potočnik, Pablo Spiga, and Gabriel Verret. A census of 4-valent half-arc-transitive graphs and arc-transitive digraphs of valence two. *Ars Mathematica Contemporanea*, 8, 2013. 19, 26, 30, 31

[PSV13b] Primož Potočnik, Pablo Spiga, and Gabriel Verret. Cubic vertex-transitive graphs on up to 1280 vertices. *Journal of Symbolic Computation*, 50:465–477, 2013. 7, 35

[PSV15] Primož Potočnik, Pablo Spiga, and Gabriel Verret. Bounding the order of the vertex-stabiliser in 3-valent vertex-transitive and 4-valent arc-transitive graphs. *Journal of Combinatorial Theory, Series B*, 111:148–180, 2015. 23, 24, 35

[PW16] Primož Potočnik and Steve Wilson. Recipes for Edge-Transitive Tetravalent Graphs. *The Art of Discrete and Applied Mathematics*, 3, 2016. 43

Index