AGT

GAP Algebraic Graph Theory package

Version 0.2

2 March 2020

Rhys J. Evans

Rhys J. Evans Email: r.evans@qmul.ac.uk
Homepage: https://www.qmul.ac.uk/maths/profiles/evansr.html

Copyright

Acknowledgements

I would like to thank Leonard Soicher for his continued support throughout the development of this package.

Contents

Chapter 1

The AGT package

This is the manual for the AGT package version 0.2, developed at Queen Mary University of London by Rhys J. Evans.

The AGT package contains a methods used for the determination of various algebraic and regularity properties of graphs, as well as certain substructures of graphs. The package also contains a library of strongly regular graphs, intended to be a useful resource for computational experiments.

All of the functions in this package deal with finite simple graphs in Grape format [Soi18]. Behind the scenes, we also use the Digraphs package [DBJM+19] to format and efficiently store and access the graphs in the strongly regular graph library.

1.1 Installing AGT

To install the AGT package, you will need to download the most recent `tar.gz` file, found at https://gap-packages.github.io/agt/. Once downloaded, you can install the package by following the instructions found in the GAP reference manual, chapter 76.

The AGT package requires the following GAP packages:

- GAPDoc [LN19], version 1.6 or higher;
- DESIGN [Soi19], version 1.7 or higher;
- GRAPE [Soi18], version 4.8 or higher;
- Digraphs [DBJM+19], version 0.12.2 or higher.

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

1.2 Loading AGT

Once correctly installed, you can load the AGT package at the GAP prompt by typing the following.

Example

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

1.3 Citing AGT

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

bibtex

```
@Manual{agt,
        author = {Evans, Rhys J.},
        key = {agt},
        title = {{AGT -- Algebraic Graph Theory package for GAP, Version 0.2}},
        url = {\verb+(https://gap-packages.github.io/agt/)+},
        year = {2020}
```

1.4 Examples of the use of AGT

We will give a simple example of how to use the AGT package here. Further applications of the package can be found in [Eva20]. In this example, we will examine the properties and subgraphs of the strongly regular graphs with parameters $(16,6,2,2)$.

Example

```
gap> LoadPackage("agt");
true
gap> IsFeasibleSRGParameters([16,6,2,2]);
true
gap> NrSRGs([16,6,2,2]);
2
gap> IsEnumeratedSRGParameterTuple([16,6,2,2]);
true
gap> graphs:=AllSRGs([16,6,2,2]);;
gap> LeastEigenvalueFromSRGParameters([16,6,2,2]);
-2
gap> LeastEigenvalueInterval(graphs[1],1/10);
[ -2, -2 ]
gap> HoffmanCliqueBound([16,6,2,2]);
4
gap> CliqueAdjacencyBound([16,6,2]);
4
gap> S:=[1,2,3,4];
[ 1, 2, 3, 4 ]
gap> Nexus(graphs[1],S);
1
gap> RegularSetParameters(graphs[1],S);
[ 3, 1 ]
```

Chapter 2

Regular graphs

In this chapter we give functions used to identify graphs with various regularity properties and determine their parameters.

2.1 Regular graphs

A graph Γ is *regular* with *parameters* (v,k) if Γ is simple and undirected, it has order v, and every vertex of Γ has degree k.

2.1.1 RGParameters

▷ `RGParameters(`*gamma*`)` (function)

Returns: A list or `fail`.

Given a graph *gamma*, this function returns the regular graph parameters of *gamma*. If *gamma* is not a regular graph, the function returns `fail`.

Example

```
gap> gamma:=EdgeOrbitsGraph(Group((2,3,4,5)),[[1,2],[2,1]]);;
gap> RGParameters(gamma);
fail
gap> gamma:=HammingGraph(3,4);;
gap> RGParameters(gamma);
[ 64, 9 ]
```

2.1.2 IsRG

▷ `IsRG(`*gamma*`)` (function)

Returns: `true` or `false`.

Given a graph *gamma*, this function returns `true` if *gamma* is a regular graph, and `false` otherwise.

Example

```
gap> gamma:=NullGraph(Group(()),5);;
gap> IsRG(gamma);
true
```

```
gap> gamma:=EdgeOrbitsGraph(Group((2,3,4,5)),[[1,2],[2,1]]);;
gap> IsRG(gamma);
false
gap> gamma:=TriangularGraph(6);;
gap> IsRG(gamma);
true
```

2.1.3 IsFeasibleRGParameters

▷ `IsFeasibleRGParameters(`*`[v, k]`*`)` (function)

Returns: `true` or `false`.

Given a list of integers of length 2, *`[v,k]`*, this function returns `true` if (v,k) is a feasible parameter tuple for a regular graph. Otherwise, the function returns `false`.

The tuple (v,k) is a *feasible* parameter tuple for a regular graph if it satisfies the following well-known conditions:

- $v > k \geq 0$;
- 2 divides vk.

Any regular graph must have parameters that satisfy these conditions (see [BCN89]).

Example

```
gap> IsFeasibleRGParameters([15,9]);
false
gap> IsFeasibleRGParameters([16,9]);
true
```

2.2 Edge-regular graphs

A graph Γ is *edge-regular* with *parameters* (v,k,a) if it is regular with parameters (v,k), it has at least one edge, and every pair of adjacent vertices have exactly a common neighbours.

2.2.1 ERGParameters

▷ `ERGParameters(`*`gamma`*`)` (function)

Returns: A list or `fail`.

Given a graph *`gamma`*, this function returns the edge-regular graph parameters of *`gamma`*. If *`gamma`* is not an edge-regular graph, the function returns `fail`.

Example

```
gap> gamma:=NullGraph(Group(()),5);;
gap> ERGParameters(gamma);
fail
gap> gamma:=JohnsonGraph(7,3);;
```

```
gap> ERGParameters(gamma);
[ 35, 12, 5 ]
```

2.2.2 IsERG

▷ `IsERG(gamma)` (function)

Returns: `true` or `false`.

Given a graph *gamma*, this function returns `true` if *gamma* is an edge-regular graph, and `false` otherwise.

Example

```
gap> gamma:=NullGraph(Group(()),5);;
gap> IsERG(gamma);
false
gap> gamma:=JohnsonGraph(7,3);;
gap> IsERG(gamma);
true
```

2.2.3 IsFeasibleERGParameters

▷ `IsFeasibleERGParameters([v, k, a])` (function)

Returns: `true` or `false`.

Given a list of integers of length 3, *[v,k,a]*, this function returns `true` if $(v,\ k,\ a)$ is a feasible parameter tuple for an edge-regular graph. Otherwise, the function returns `false`.

The tuple (v,k,a) is a *feasible* parameter tuple for an edge-regular graph if it satisfies the following well-known conditions:

- (v,k) is a feasible regular graph parameter tuple;
- $k > a \geq 0$;
- 2 divides ka and 6 divides vka;
- $v - 2k + a \geq 0$.

Any edge-regular graph must have parameters which satisfy these conditions (see [BCN89]).

Example

```
gap> IsFeasibleERGParameters([15,9,6]);
false
gap> IsFeasibleERGParameters([16,9,4]);
true
```

2.3 Strongly regular graphs

A graph Γ is *strongly regular* with *parameters* (v,k,a,b) if it is edge-regular with parameters (v,k,a), it has at least one pair of distinct non-adjacent vertices, and every pair of distinct non-adjacent vertices have exactly b common neighbours.

2.3.1 SRGParameters

▷ `SRGParameters(gamma)` (function)

Returns: A list or `fail`.

Given a graph `gamma`, this function returns the strongly regular graph parameters of `gamma`. If `gamma` is not a strongly regular graph, the function returns `fail`.

Example

```
gap> gamma:=CompleteGraph(Group(()),5);;
gap> SRGParameters(gamma);
fail
gap> gamma:=JohnsonGraph(5,3);;
gap> SRGParameters(gamma);
[ 10, 6, 3, 4 ]
```

2.3.2 IsSRG

▷ `IsSRG(gamma)` (function)

Returns: `true` or `false`.

Given a graph `gamma`, this function returns `true` if `gamma` is a strongly regular graph, and `false` otherwise.

Example

```
gap> gamma:=CompleteGraph(Group(()),5);;
gap> IsSRG(gamma);
false
gap> gamma:=JohnsonGraph(5,3);;
gap> IsSRG(gamma);
true
```

2.3.3 IsFeasibleSRGParameters

▷ `IsFeasibleSRGParameters([v, k, a, b])` (function)

Returns: `true` or `false`.

Given a list of integers of length 4, `[v,k,a,b]`, this function returns `true` if `(v, k, a, b)` is a feasible parameter tuple for a strongly regular graph. Otherwise, this function returns `false`.

The tuple (v,k,a,b) is a *feasible* parameter tuple for a strongly regular graph if it satisfies the following well-known conditions:

- (v,k,a) is a feasible edge-regular graph parameter tuple;

- $k \geq b$;
- $(v-k-1)b = k(k-a-1)$;
- $v-2-2k+b \geq 0$;
- the formulae for the multiplicities of the eigenvalues of a strongly regular graph with these parameters evaluate to positive integers (see [BH11]).

Any strongly regular graph must have parameters which satisfy these conditions (see [BCN89]).

Example

```
gap> IsFeasibleSRGParameters([15,9,4,7]);
false
gap> IsFeasibleSRGParameters([10,3,0,1]);
true
```

Chapter 3

Spectra of graphs

In this chapter we give methods for investigating the eigenvalues of a graph.

Let Γ be a graph of order v. The *adjacency matrix* of Γ, $A(\Gamma)$, is the $v \times v$ matrix indexed by $V(\Gamma)$ such that $A(\Gamma)_{xy} = 1$ if $xy \in E(\Gamma)$, and $A(\Gamma)_{xy} = 0$ otherwise.

The *spectrum* of Γ, $Spec(\Gamma)$, is the multiset of eigenvalues of its adjacency matrix, and an *eigenvalue of* Γ is a member of $Spec(\Gamma)$. The *multiplicity* of an eigenvalue α of Γ is the number of times α appears in $Spec(\Gamma)$. For information on most of the objects and results discussed in this chapter, see [BH11].

3.1 Eigenvalues of regular graphs

In this section, we introduce methods for investigating eigenvalues of regular graphs. The input for these methods will be a specific graph or the parameters of a graph.

Let Γ be a regular graph with parameters (v,k). Then Γ has largest eigenvalue k (see [BH11]). Therefore we do not implement a "LargestEigenvalue" function for regular graphs.

Let Γ be a strongly regular graph with parameters (v,k,a,b). The eigenvalues of Γ and their corresponding multiplicites are uniquely determined by the parameters (v,k,a,b) (see [BH11]). Using this knowledge, we provide methods which take as input feasible strongly regular graph parameters (v,k,a,b). We also give methods which return an exact representation of the eigenvalues of a strongly regular graph with parameters (v,k,a,b), and their multiplicities.

3.1.1 LeastEigenvalueInterval

▷ `LeastEigenvalueInterval(`*`gamma, eps`*`)` (operation)

▷ `LeastEigenvalueInterval(`*`parms, eps`*`)` (operation)

Returns: A list.

Given a graph *`gamma`* and rational number *`eps`*, this function returns an interval containing the least eigenvalue of *`gamma`*.

Given feasible strongly regular graph parameters *`parms`* and rational number *`eps`*, this function returns an interval containing the least eigenvalue of a strongly regular graph with parameters *`parms`*.

The interval returned is in the form of a list, *`[y,z]`* of rationals $y \leq z$ with the property that $z - y \leq$ *`eps`*. If the eigenvalue is rational this function will return a list *`[y,z]`*, where $y = z$.

Example

```
gap> gamma:=EdgeOrbitsGraph(Group((1,2,3,4,5)),[[1,2],[2,1]]);;
```

```
gap> LeastEigenvalueInterval(gamma,1/10);
[ -13/8, -25/16 ]
gap> parms:=SRGParameters(gamma);
[ 5, 2, 0, 1 ]
gap> LeastEigenvalueInterval(parms,1/10);
[ -13/8, -25/16 ]
gap> LeastEigenvalueInterval(JohnsonGraph(7,3),1/20);
[ -3, -3 ]
```

3.1.2 SecondEigenvalueInterval

▷ SecondEigenvalueInterval(*gamma, eps*) (operation)
▷ SecondEigenvalueInterval(*parms, eps*) (operation)

Returns: A list.

Given a regular graph *gamma* and rational number *eps*, this function returns an interval containing the second largest eigenvalue of *gamma*.

Given feasible strongly regular graph parameters *parms* and rational number *eps*, this function returns an interval containing the second largest eigenvalue of a strongly regular graph with parameters *parms*.

The interval returned is in the form of a list, *[y,z]* of rationals $y \leq z$ with the property that $z - y \leq$ *eps*. If the eigenvalue is rational this function will return a list *[y,z]*, where $y = z$.

Example

```
gap> gamma:=EdgeOrbitsGraph(Group((1,2,3,4,5)),[[1,2],[2,1]]);;
gap> SecondEigenvalueInterval(gamma,1/10);
[ 9/16, 5/8 ]
gap> parms:=SRGParameters(gamma);
[ 5, 2, 0, 1 ]
gap> SecondEigenvalueInterval(parms,1/10);
[ 9/16, 5/8 ]
gap> SecondEigenvalueInterval(JohnsonGraph(7,3),1/20);
[ 5, 5 ]
```

3.1.3 LeastEigenvalueFromSRGParameters

▷ LeastEigenvalueFromSRGParameters(*[v, k, a, b]*) (function)

Returns: An integer or an element of a cyclotomic field.

Given feasible strongly regular graph parameters *[v, k, a, b]* this function returns the least eigenvalue of a strongly regular graph with parameters (*v,k,a,b*). If the eigenvalue is integer, the object returned is an integer. If the eigenvalue is irrational, the object returned lies in a cyclotomic field.

Example

```
gap> LeastEigenvalueFromSRGParameters([5,2,0,1]);
E(5)^2+E(5)^3
gap> LeastEigenvalueFromSRGParameters([10,3,0,1]);
```

```
-2
```

3.1.4 SecondEigenvalueFromSRGParameters

▷ SecondEigenvalueFromSRGParameters(*[v, k, a, b]*) (function)

Returns: An integer or an element of a cyclotomic field.

Given feasible strongly regular graph parameters *[v, k, a, b]*, this function returns the second largest eigenvalue of a strongly regular graph with parameters (v,k,a,b). If the eigenvalue is integer, the object returned is an integer. If the eigenvalue is irrational, the object returned lies in a cyclotomic field.

Example

```
gap> SecondEigenvalueFromSRGParameters([5,2,0,1]);
E(5)+E(5)^4
gap> SecondEigenvalueFromSRGParameters([10,3,0,1]);
1
```

3.1.5 LeastEigenvalueMultiplicity

▷ LeastEigenvalueMultiplicity(*[v, k, a, b]*) (operation)

Returns: An integer.

Given feasible strongly regular graph parameters *[v, k, a, b]* this function returns the multiplicity of the least eigenvalue of a strongly regular graph with parameters (v,k,a,b).

Example

```
gap> LeastEigenvalueMultiplicity([16,9,4,6]);
6
gap> LeastEigenvalueMultiplicity([25,12,5,6]);
12
```

3.1.6 SecondEigenvalueMultiplicity

▷ SecondEigenvalueMultiplicity(*[v, k, a, b]*) (operation)

Returns: An integer.

Given feasible strongly regular graph parameters *[v, k, a, b]* this function returns the multiplicity of the second eigenvalue of a strongly regular graph with parameters (v,k,a,b).

Example

```
gap> SecondEigenvalueMultiplicity([16,9,4,6]);
9
gap> SecondEigenvalueMultiplicity([25,12,5,6]);
12
```

Chapter 4

Regular induced subgraphs

In this chapter we give methods to investigate regular induced subgraphs of regular graphs.

Let Γ be a graph, and consider a subset U of its vertices. The *induced subgraph* of Γ on U, $\Gamma[U]$, is the graph with vertex set U, and vertices in $\Gamma[U]$ are adjacent if and only if they are adjacent in Γ.

4.1 Spectral bounds

In this section, we introduce some bounds on regular induced subgraphs of regular graphs, which depend on the spectrum of the graph.

Let Γ be a graph. A *coclique*, or *independent set*, of Γ is a subset of vertices for which each pair of distinct vertices are non-adjacent. A *clique* of Γ is a subset of vertices for which each pair of distinct vertices are adjacent.

4.1.1 HoffmanCocliqueBound

▷ `HoffmanCocliqueBound(`*`gamma`*`)` (operation)
▷ `HoffmanCocliqueBound(`*`parms`*`)` (operation)

Returns: An integer.

Given a non-null regular graph *`gamma`*, this function returns the Hoffman coclique bound of *`gamma`*.

Given feasible strongly regular graph parameters *`parms`*, this function returns the Hoffman coclique bound of a strongly regular graph with parameters *`parms`*.

Let Γ be a non-null regular graph with parameters (v,k) and least eigenvalue s. The *Hoffman coclique bound*, or *ratio bound* of Γ, is defined as

$$\delta = \lfloor \left(\frac{v}{k-s} \right) \rfloor.$$

It is known that any coclique in Γ must contain at most δ vertices (see [BH11]).

Example

```
gap> HoffmanCocliqueBound(HammingGraph(3,5));
25
gap> HoffmanCocliqueBound(HammingGraph(2,5));
5
gap> parms:=SRGParameters(HammingGraph(2,5));
```

```
[ 25, 8, 3, 2 ]
gap> HoffmanCocliqueBound(parms);
5
```

4.1.2 HoffmanCliqueBound

▷ HoffmanCliqueBound(*gamma*) (operation)
▷ HoffmanCliqueBound(*parms*) (operation)

Returns: An integer.

Given a non-null, non-complete regular graph *gamma*, this function returns the Hoffman clique bound of *gamma*.

Given feasible strongly regular graph parameters *parms*, this function returns the Hoffman clique bound of a strongly regular graph with parameters *parms*.

Let Γ be a non-null, non-complete regular graph. The *Hoffman clique bound* of Γ, is defined as the Hoffman coclique bound of its complement (see HoffmanCocliqueBound (4.1.1)). It is known that the Hoffman clique bound is an upper bound on the number of vertices in any clique of Γ (see [BH11]). Note that in the case that Γ is a strongly regular graph, this function returns the value of the well-known *Delsarte-Hoffman clique bound* (see [Del73]).

Example

```
gap> gamma:=EdgeOrbitsGraph(CyclicGroup(IsPermGroup,15),[[1,2],[2,1]]);;
gap> HoffmanCliqueBound(gamma);
2
gap> gamma:=JohnsonGraph(7,2);;
gap> HoffmanCliqueBound(gamma);
6
gap> parms:=SRGParameters(gamma);
[ 21, 10, 5, 4 ]
gap> HoffmanCliqueBound(parms);
6
```

4.1.3 HaemersRegularUpperBound

▷ HaemersRegularUpperBound(*gamma, d*) (operation)
▷ HaemersRegularUpperBound(*parms, d*) (operation)

Returns: An integer.

Given a non-null regular graph *gamma* and non-negative integer *d*, this function returns the Haemers upper bound on *d*-regular induced subgraphs of *gamma*.

Given feasible strongly regular graph parameters *parms* and non-negative integer *d*, this function returns the Haemers upper bound on *d*-regular induced subgraphs of a strongly regular graph with parameters *parms*.

Let Γ be a non-null regular graph with parameters (v,k) and least eigenvalue s and let d be a

non-negative integer. The *Haemers upper bound* on d-regular induced subgraphs of Γ, is defined as

$$\delta = \lfloor \left(\frac{v(d-s)}{k-s} \right) \rfloor.$$

It is known that any d-regular induced subgraph in Γ has order at most δ (see [Eva20]).

Example

```
gap> HaemersRegularUpperBound(SimsGerwitzGraph(),3);
28
gap> HaemersRegularUpperBound([56,10,0,2],0);
16
```

4.1.4 HaemersRegularLowerBound

▷ `HaemersRegularLowerBound(gamma, d)` (operation)
▷ `HaemersRegularLowerBound(parms, d)` (operation)

Returns: An integer.

Given a connected regular graph `gamma` and non-negative integer `d`, this function returns the Haemers lower bound on `d`-regular induced subgraphs of `gamma`.

Given the parameters of a connected strongly regular graph, `parms`, and non-negative integer `d`, this function returns the Haemers lower bound on `d`-regular induced subgraphs of a strongly regular graph with parameters `parms`.

Let Γ be a connected regular graph with parameters (v,k) and second eigenvalue r and let d be a non-negative integer. The *Haemers lower bound* on d-regular induced subgraphs of Γ, is defined as

$$\delta = \lfloor \left(\frac{v(d-r)}{k-r} \right) \rfloor.$$

It is known that any d-regular induced subgraph in Γ has order at least δ (see [Eva20]).

Example

```
gap> HaemersRegularLowerBound(HoffmanSingletonGraph(),4);
20
gap> HaemersRegularLowerBound([50,7,0,1],3);
10
```

4.2 Block intersection polynomials and bounds

In this section, we introduce functions related to the block intersection polynomials, defined in [Soi10]. If you would like to know more about the properties of these polynomials, see [Soi10], [Soi15] and [GS16].

4.2.1 CliqueAdjacencyPolynomial

▷ `CliqueAdjacencyPolynomial(parms, x, y)` (function)

Returns: A polynomial.

Given feasible edge-regular graph parameters `parms` and indeterminates `x,y`, this function returns the clique adjacency polynomial with respect to the parameters `parms` and indeterminates `x,y`, defined in [Soi15].

Let Γ be an edge-regular graph with parameters (v,k,a). The *clique adjacency polynomial* of Γ is defined as

$$C(x,y) := x(x+1)(v-y) - 2xy(k-y+1) + y(y-1)(a-y+2).$$

Example

```
gap> x:=Indeterminate(Rationals,"x");
x
gap> y:=Indeterminate(Rationals,"y");
y
gap> CliqueAdjacencyPolynomial([21,8,3],x,y);
-x^2*y-y^3+21*x^2-x*y+8*y^2+21*x-23*y
```

4.2.2 CliqueAdjacencyBound

▷ `CliqueAdjacencyBound(parms)` (function)

Returns: An integer.

Given feasible edge-regular graph parameters `parms`, this function returns the clique adjacency bound with respect to the parameters `parms`, defined in [Soi10].

Let Γ be an edge-regular graph with parameters (v,k,a), and let C be its corresponding clique adjacency poylnomial (see `CliqueAdjacencyPolynomial` (4.2.1)). The *clique adjacency bound* of Γ is defined as the smallest integer $y \geq 2$ such that there exists an integer m for which $C(m,y+1) < 0$. It is known that the clique adjacency bound is an upper bound on the number of vertices in any clique of Γ.

Example

```
gap> CliqueAdjacencyBound([16,6,2]);
4
```

4.2.3 RegularAdjacencyPolynomial

▷ `RegularAdjacencyPolynomial(parms, x, y, d)` (function)

Returns: A polynomial.

Given feasible strongly regular graph parameters `parms` and indeterminates `x,y,d`, this function returns the regular adjacency polynomial with respect to the parameters `parms` and indeterminates `x,y,d`, as defined in [Eva20].

Let Γ be a strongly regular graph with parameters (v,k,a,b). The *regular adjacency polynomial* of Γ is defined as

$$R(x,y,d) := x(x+1)(v-y) - 2xyk + (2x+a-b+1)yd + y(y-1)b - yd^2.$$

Example

```
gap> RegularAdjacencyPolynomial([16,6,2,2],"x","y","d");
-x^2*y+2*x*y*d-y*d^2+16*x^2-x*y+2*y^2+y*d+4*x-2*y
```

4.2.4 RegularAdjacencyUpperBound

▷ `RegularAdjacencyUpperBound(`*parms*`, `*d*`)` (function)

Returns: An integer.

Given strongly regular graph parameters *parms* and non-negative integer *d*, this function returns the regular adjacency upper bound with respect to the parameters *parms* and integer *d*, defined in [Eva20].

Let Γ be a strongly regular graph with parameters (v,k,a,b), and let R be its corresponding regular adjacency poylnomial (see `RegularAdjacencyPolynomial` (4.2.3)). For fixed d, the *regular adjacency upper bound* of Γ is defined as the largest integer $d+1 \leq y \leq v$ such that for all integers m, we have $R(m,y,d) \geq 0$ if such a y exists, and 0 otherwise. It is known that the regular adjacency upper bound is an upper bound on the number of vertices in any d-regular induced subgraph of Γ.

Example

```
gap> RegularAdjacencyUpperBound([56,10,0,2],3);
28
```

4.2.5 RegularAdjacencyLowerBound

▷ `RegularAdjacencyLowerBound(`*parms*`, `*d*`)` (function)

Returns: An integer.

Given the parameters of a connected strongly regular graph, *parms*, and non-negative integer *d*, this function returns the regular adjacency lower bound with respect to the parameters *parms* and integer *d*, defined in [Eva20].

Let Γ be a strongly regular graph with parameters (v,k,a,b), and let R be its corresponding regular adjacency poylnomial (see `RegularAdjacencyPolynomial` (4.2.3)). For fixed d, the *regular adjacency lower bound* of Γ is defined as the smallest integer $d+1 \leq y \leq v$ such that for all integers m, we have $R(m,y,d) \geq 0$ if such a y, and $v+1$ otherwise. It is known that the regular adjacency lower bound is a lower bound on the number of vertices in any d-regular induced subgraph of Γ.

Example

```
gap> RegularAdjacencyLowerBound([50,7,0,1],2);
5
```

4.3 Regular sets

In this section we give functions to investigate regular sets, with a focus on regular sets in strongly regular graphs.

Let Γ be a graph and U be a subset of its vertices. Then U is *m-regular* if every vertex in $V(\Gamma)\backslash U$ is adjacent to the same number $m > 0$ of vertices in U. In this case we say that U has *nexus* m.

The set U is a (d,m)-*regular set* if U is an m-regular set and $\Gamma[U]$ is a d-regular graph. Then we call (d,m) the *regular set parameters* of U.

4.3.1 Nexus

▷ `Nexus(gamma, U)` (function)

Returns: An integer or `fail`.

Given a graph `gamma` and a subset `U` of its vertices, this function returns the nexus of `U`. If `U` is not an m-regular set for some $m > 0$, the function returns `fail`.

Example

```
gap> Nexus(SquareLatticeGraph(5),[1,2,3,4,6]);
fail
gap> Nexus(SquareLatticeGraph(5),[1,2,3,4,5]);
1
```

4.3.2 RegularSetParameters

▷ `RegularSetParameters(gamma, U)` (function)

Returns: A list or `fail`.

Given a graph `gamma` and a subset `U` of its vertices, this function returns a list `[d,m]` such that `U` is a (d,m)-regular set. If `U` is not an (d,m)-regular set for some d,m, the function returns `fail`.

Example

```
gap> RegularSetParameters(SquareLatticeGraph(5),[6,11,16,21]);
fail
gap> RegularSetParameters(SquareLatticeGraph(5),[1,6,11,16,21]);
[ 4, 1 ]
```

4.3.3 IsRegularSet

▷ `IsRegularSet(gamma, U, opt)` (function)

Returns: `true` or `false`.

Given a graph `gamma` and a subset `U` of its vertices, this function returns `true` if `U` is a regular set, and `false` otherwise.

The input `opt` should take a boolean value `true` or `false`. This option effects the output of the function in the following way.

`true`
: this function will return `true` if and only if `U` is a (d,m)-regular set for some d,m.

`false`
: this function will return `true` if and only if `U` is a m-regular set for some m.

Example

```
gap> IsRegularSet(HoffmanSingletonGraph(),[11..50],false);
true
gap> IsRegularSet(HoffmanSingletonGraph(),[11..50],true);
false
```

4.3.4 RegularSetSRGParameters

▷ `RegularSetSRGParameters(parms, d)` (function)

Returns: A list.

Given feasible strongly regular graph parameters `parms` and non-negative integer `d`, this function returns a list of pairs `[s,m]` with the following properties:

- (d,m) are feasible regular set parameters for a strongly regular graph with parameters `parms`.
- s is the order of any (d,m)-regular set in a strongly regular graph with parameters `parms`.

Let Γ be a strongly regular graph with parameters (v,k,a,b) and let R be its corresponding regular adjacency polynomial (see `RegularAdjacencyPolynomial` (4.2.3)). Then the tuple (d,m) is a *feasible regular set parameter tuple* for Γ if d,m are non-negative integers and there exists a positive integer s such that

$$R(m-1,s,d) = R(m,s,d) = 0.$$

It is known that any (d,m)-regular set of size s in Γ must satisfy the above conditions (see [Eva20]).

Example

```
gap> RegularSetSRGParameters([16,6,2,2],4);
[ [ 8, 2 ], [ 12, 6 ] ]
```

4.4 Neumaier graphs

In this section, we give functions to investigate regular cliques in edge-regular graphs.

A clique S in Γ is *m-regular*, for some $m > 0$, if S is an m-regular set. A graph Γ is a *Neumaier graph* with *parameters* $(v,k,a;s,m)$ if it is edge-regular with parameters (v,k,a), and Γ contains an m-regular clique of size s. For more information on Neumaier graphs, see [Eva20].

4.4.1 NGParameters

▷ `NGParameters(gamma)` (function)

Returns: A list or `fail`.

Given a graph *gamma*, this function returns the Neumaier graph parameters of *gamma*. If *gamma* is not a Neumaier graph, the function returns `fail`.

Example

```
gap> NGParameters(HigmanSimsGraph());
fail
gap> NGParameters(TriangularGraph(10));
[ [ 45, 16, 8, 9, 2 ] ]
```

4.4.2 IsNG

▷ `IsNG(gamma)` (function)

Returns: `true` or `false`.

Given a graph *gamma*, this function returns `true` if *gamma* is a Neumaier graph, and `false` otherwise.

Example

```
gap> IsNG(HammingGraph(3,7));
false
gap> IsNG(HammingGraph(2,7));
true
```

4.4.3 IsFeasibleNGParameters

▷ `IsFeasibleNGParameters([v, k, a, s, m])` (function)

Returns: `true` or `false`.

Given a list of integers of length 5, `[v,k,a,s,m]`, this function returns `true` if $(v,k,a;s,m)$ is a feasible parameter tuple for a Neumaier graph. Otherwise, the function returns `false`.

The tuple $(v,k,a;s,m)$ is a *feasible* parameter tuple for a Neumaier graph if it satisfies the following conditions:

- (v,k,a) is a feasible edge-regular graph parameter tuple;
- $0 < m \leq s$ and $2 \leq s \leq a+2$;
- $(v-s)m = (k-s+1)s$;
- $(k-s+1)(m-1) = (a-s+2)(s-1)$.

Any Neumaier graph must have parameters which satisfy these conditions (see [Eva20]).

Example

```
gap> IsFeasibleNGParameters([35,16,6,5,2]);
true
gap> IsFeasibleNGParameters([37,18,8,5,2]);
false
```

4.4.4 RegularCliqueERGParameters

▷ RegularCliqueERGParameters(*parms*) (function)

Returns: A list.

Given feasible edge-regular graph parameters *parms*=[v,k,a], this function returns a list of pairs [s,m], such that (*v*,*k*,*a*;*s*,*m*) are feasible Neumaier graph parameters (as defined in IsFeasibleNGParameters (4.4.3)).

Example

```
gap> RegularCliqueERGParameters([8,7,6]);
[ [ 1, 1 ], [ 2, 2 ], [ 3, 3 ], [ 4, 4 ], [ 5, 5 ], [ 6, 6 ], [ 7, 7 ] ]
gap> RegularCliqueERGParameters([8,6,4]);
[ [ 4, 3 ] ]
gap> RegularCliqueERGParameters([16,9,4]);
[ [ 4, 2 ] ]
```

Chapter 5

Strongly regular graphs

In this chapter we give functions to investigate strongly regular graphs. In particular, we provide a collection of strongly regular graphs which can be a useful computational resource.

5.1 Strongly regular graph parameter tuples

In this section, we give functions to investigate the parameters of a strongly regular graph. For the definition of feasible strongly regular graph parameters, see `IsFeasibleSRGParameters` (2.3.3).

5.1.1 ComplementSRGParameters

▷ `ComplementSRGParameters(`*parms*`)` (function)

Returns: A list.

Given feasible strongly regular graph parameters *parms*, this function returns the complement parameters of *parms*.

Suppose Γ is a strongly regular graph with parameters (v,k,a,b). Then the complement of Γ is a strongly regular graph with parameters $(v, v-k-1, v-2-2k+b, v-2k+a)$ (see [BCN89]). We define the *complement parameters* of the feasible strongly regular graph parameter tuple (v,k,a,b) as the tuple $(v, v-k-1, v-2-2k+b, v-2k+a)$.

Example

```
gap> ComplementSRGParameters([16,9,4,6]);
[ 16, 6, 2, 2 ]
```

5.1.2 SRGToGlobalParameters

▷ `SRGToGlobalParameters(`*parms*`)` (function)

Returns: A list.

Given feasible strongly regular graph parameters *parms*, this function returns the global parameters of a graph with strongly regular graph parameters *parms*. For information on global parameters of a graph, see [Soi18].

Example

```
gap> SRGToGlobalParameters([50,7,0,1]);
[ [ 0, 0, 7 ], [ 1, 0, 6 ], [ 1, 6, 0 ] ]
```

5.1.3 GlobalToSRGParameters

▷ `GlobalToSRGParameters(parms)` (function)

Returns: A list.

Given the global parameters *parms* of a connected strongly regular graph, this function returns the strongly regular graph parameters of the graph. For information on global parameters of a graph, see [Soi18].

Example

```
gap> parms:=GlobalParameters(JohnsonGraph(5,3));
[ [ 0, 0, 6 ], [ 1, 3, 2 ], [ 4, 2, 0 ] ]
gap> GlobalToSRGParameters(parms);
[ 10, 6, 3, 4 ]
```

5.1.4 IsPrimitiveSRGParameters

▷ `IsPrimitiveSRGParameters([v, k, a, b])` (function)

Returns: `true` or `false`.

Given a list of integers of length 4, *[v,k,a,b]*, this function returns `true` if (v,k,a,b) is a feasible parameter tuple for a primitive strongly regular graph. Otherwise, this function returns `false`.

Let (v,k,a,b) be feasible strongly regular parameters with complement parameters (v',k',a',b'). Then the parameter tuple (v,k,a,b) is called *primitive* if $b \neq 0$ and $b' \neq 0$.

A strongly regular graph Γ is called *primitive* if Γ and its complement is connected. It is known that a non-primitive strongly regular graph is a union of cliques, or the complement of a union of cliques. From our definition, it follows that a strongly regular graph Γ is primitive if and only if Γ has primitive strongly regular graph parameters (see [BCN89]).

Example

```
gap> IsFeasibleSRGParameters([8,6,4,6]);
true
gap> IsPrimitiveSRGParameters([8,6,4,6]);
false
gap> IsPrimitiveSRGParameters([10,6,3,4]);
true
```

5.1.5 IsTypeISRGParameters

▷ `IsTypeISRGParameters([v, k, a, b])` (function)

Returns: `true` or `false`.

Given a list of integers of length 4, `[v,k,a,b]`, this function returns `true` if `(v,k,a,b)` is a feasible parameter tuple for a type I strongly regular graph. Otherwise, this function returns `false`.

A feasible strongly regular parameter tuple (v,k,a,b) is of *type I* (or a *conference graph*) if there exists a positive integer t such that $v = 4t+1, k = 2t, a = t-1, b = t$.

There are two types of strongly regular graphs, called type I and type II (see [BCN89]). Let Γ be a strongly regular graph with parameters (v,k,a,b). Then Γ is of *type I* if and only if (v,k,a,b) is of type I.

Example

```
gap> IsTypeISRGParameters([5,2,0,1]);
true
gap> IsTypeISRGParameters([9,4,1,2]);
true
gap> IsTypeISRGParameters([10,3,0,1]);
false
```

5.1.6 IsTypeIISRGParameters

▷ `IsTypeIISRGParameters([v, k, a, b])` (function)

Returns: `true` or `false`.

Given a list of integers of length 4, `[v,k,a,b]`, this function returns `true` if `(v,k,a,b)` is a feasible parameter tuple for a type II strongly regular graph. Otherwise, this function returns `false`.

A feasible strongly regular parameter tuple (v,k,a,b) is of *type II* if the polynomial $x^2-(a-b)x+b-k$ has integer zeros.

There are two types of strongly regular graphs, called type I and *type II* (see [BCN89]). Let Γ be a strongly regular graph with parameters (v,k,a,b). Then Γ is of *type II* if and only if all of its eigenvalues are integer. The eigenvalues of Γ are k and the zeros of the polynomial $x^2-(a-b)x+b-k$ (see [BCN89]). From our definition, it follows that Γ is of type II if and only if (v,k,a,b) is of type II.

Example

```
gap> IsTypeIISRGParameters([5,2,0,1]);
false
gap> IsTypeIISRGParameters([9,4,1,2]);
true
gap> IsTypeIISRGParameters([10,3,0,1]);
true
```

5.1.7 KreinParameters

▷ `KreinParameters([v, k, a, b])` (operation)

Returns: A list.

Given feasible strongly regular graph parameters `[v, k, a, b]`, this function returns a list of (non-trivial) Krein parameters of a strongly regular graph with parameters `(v,k,a,b)`.

If the eigenvalues of a strongly regular graph are integer, the object returned is a list of integers. If the eigenvalues are irrational, the object returned will be a list of cyclotomic numbers.

Let Γ be a strongly regular graph with parameters (v,k,a,b) and eigenvalues $k \geq r > s$. Then the *Krein parameters* of Γ are the numbers

$$K_1 = (k+r)(s+1)^2 - (r+1)(k+r+2rs),$$

$$K_2 = (k+s)(r+1)^2 - (s+1)(k+s+2rs).$$

For information on the Krein parameters of strongly regular graphs, see [BCN89].

Example

```
gap> KreinParameters([10,6,3,4]);
[ 1, 16 ]
gap> KreinParameters([13,6,2,3]);
[ -14*E(13)-12*E(13)^2-14*E(13)^3-14*E(13)^4-12*E(13)^5-12*E(13)^6-12*E(13)^7
   -12*E(13)^8-14*E(13)^9-14*E(13)^10-12*E(13)^11-14*E(13)^12,
-12*E(13)-14*E(13)^2-12*E(13)^3-12*E(13)^4-14*E(13)^5-14*E(13)^6-14*E(13)^7
   -14*E(13)^8-12*E(13)^9-12*E(13)^10-14*E(13)^11-12*E(13)^12 ]
```

5.1.8 IsKreinConditionsSatisfied

▷ `IsKreinConditionsSatisfied([v, k, a, b])` (operation)

Returns: `true` or `false`.

Given feasible strongly regular graph parameters `[v, k, a, b]`, this function returns `true` if the parameters satisfy the Krein conditions. Otherwise, this function returns `false`.

Let Γ be a strongly regular graph with parameters (v,k,a,b) and Krein parameters K_1, K_2 (see `KreinParameters` (5.1.7)). The *Krein conditions* of Γ are the inequalities

$$K_1 \geq 0, K_2 \geq 0.$$

It is known that any strongly regular graph must have parameters that satisfy the Krein conditions. For information on the Krein conditions of strongly regular graphs, see [BCN89].

Example

```
gap> IsKreinConditionsSatisfied([28,9,0,4]);
false
gap> IsKreinConditionsSatisfied([13,6,2,3]);
true
gap> IsKreinConditionsSatisfied([10,6,3,4]);
true
```

5.1.9 IsAbsoluteBoundSatisfied

▷ `IsAbsoluteBoundSatisfied( [v, k, a, b] )` (function)

Returns: `true` or `false`.

Given primitive strongly regular graph parameters `[v, k, a, b]`, this function returns `true` if the parameters satisfy the absolute bound. Otherwise, this function returns `false`.

Let Γ be a primitive strongly regular graph with parameters (v,k,a,b) and eigenvalues $k \geq r > s$, with multiplicities $1, f, g$. The *absolute bound* for the number of vertices of Γ is

$$v \leq f(f+3)/2, v \leq g(g+3)/2.$$

For information on the absolute bound of strongly regular graphs, see [BCN89].

Example

```
gap> IsAbsoluteBoundSatisfied([13,6,3,4]);
false
gap> IsAbsoluteBoundSatisfied([50,21,4,12]);
false
gap> IsAbsoluteBoundSatisfied([50,21,8,9]);
true
```

5.2 Small strongly regular graphs

In this section, we give functions to access and use the library of strongly regular graphs currently stored in this package. The information on small strongly regular graphs in this section is based on the tables of Andries Brouwer [Bro18a].

5.2.1 AGT_Brouwer_Parameters_MAX

▷ `AGT_Brouwer_Parameters_MAX` (global variable)

This variable stores the largest value v for which the current package contains information about the parameters of all strongly regular graphs with at most v vertices. This information is stored in the list `AGT_Brouwer_Parameters` (5.2.2).

5.2.2 AGT_Brouwer_Parameters

▷ `AGT_Brouwer_Parameters` (global variable)

This variable stores information about the feasible strongly regular graph parameters found in Brouwer [Bro18a] and the available strongly regular graphs. *`AGT_Brouwer_Parameters`* is a list, each element of which is a list of length 4. For an element `[parms,status,stored,num]`, each entry denotes the following;

`parms`
: A feasible strongly regular graph parameter tuple `[v,k,a,b]`, where `v` is less than `AGT_Brouwer_Parameters_MAX` (5.2.1).

`status`
: the status of the known strongly regular graphs with parameters `parms`. This entry is

  - 0 if there exists a strongly regular graph with parameters `parms`, and all such graphs have been enumerated,
  - 1 if there exists a strongly regular graph with parameters `parms`, but all such graphs have not been enumerated,
  - 2 if it is not known if a strongly regular graph with parameters `parms` exists,
  - 3 if it has been proven that no strongly regular graph with parameters `parms` exists.

`stored`
: the number of non-isomoprhic strongly regular graphs with parameters `parms` that are available in the current package.

`num` the number of non-isomorphic strongly regular graphs with parameters `parms` that exist. If this has not been determined, the value of `num` is set to `-1`.

Example

```
gap> AGT_Brouwer_Parameters[34];
[ [ 36, 20, 10, 12 ], 0, 32548, 32548 ]
gap> AGT_Brouwer_Parameters[35];
[ [ 37, 18, 8, 9 ], 1, 6760, -1 ]
gap> AGT_Brouwer_Parameters[2530];
[ [ 765, 176, 28, 44 ], 2, 0, -1 ]
gap> AGT_Brouwer_Parameters[4530];
[ [ 1293, 646, 322, 323 ], 3, 0, 0 ]
```

5.2.3 IsSRGAvailable

▷ `IsSRGAvailable(`*parms*`)` (function)

Returns: `true` or `false`.

Given feasible strongly regular graph parameters *parms*, this function returns `true` if there is a strongly regular graph with parameters *parms* stored in this package. If *parms* is a feasible parameter tuple but there is no strongly regular graphs with parameters *parms* available, the function returns `false`.

Example

```
gap> IsSRGAvailable([28,12,6,4]);
true
gap> IsSRGAvailable([28,9,0,4]);
false
```

5.2.4 SRGLibraryInfo

▷ SRGLibraryInfo(*parms*) (function)

Returns: A list.

Given feasible strongly regular graph parameters *parms*, with first parameter *v* at most AGT_Brouwer_Parameters_MAX (5.2.1), this function returns the element of AGT_Brouwer_Parameters (5.2.2) corresponding to *parms*.

Example

```
gap> SRGLibraryInfo([37,18,8,9]);
[ [ 37, 18, 8, 9 ], 1, 6760, -1 ]
gap> SRGLibraryInfo([36,15,6,6]);
[ [ 36, 15, 6, 6 ], 0, 32548, 32548 ]
```

5.2.5 SRG

▷ SRG(*parms*, *n*) (function)

Returns: A record or fail.

Given feasible strongly regular graph parameters *parms* and positive integer *n*, this function returns the *n*th strongly regular graph with parameters *parms* available in this package. If there there is no *n*th strongly regular graph with parameters *parms* available, the function returns fail.

Example

```
gap> SRG([16,6,2,2],1);
rec( adjacencies := [ [ 2, 3, 4, 5, 6, 7 ] ],
group := <permutation group with 6 generators>, isGraph := true,
names := [ 1 .. 16 ], order := 16, representatives := [ 1 ],
schreierVector := [ -1, 6, 4, 3, 5, 5, 5, 6, 6, 6, 4, 4, 4, 3, 3, 3 ] )
gap> SRG([16,6,2,2],2);
rec( adjacencies := [ [ 2, 3, 4, 5, 6, 7 ] ], group := Group([ (3,4)(5,6)(8,9)
(11,14)(12,13)(15,16), (2,3)(4,5)(6,7)(9,11)(10,12)(14,15), (1,2)(5,8)(6,9)
(7,10)(11,12)(13,14) ]), isGraph := true, names := [ 1 .. 16 ],
order := 16, representatives := [ 1 ],
schreierVector := [ -1, 3, 2, 1, 2, 1, 2, 3, 3, 3, 2, 2, 1, 1, 2, 1 ] )
gap> SRG([28,9,0,4],1);
fail
```

5.2.6 NrSRGs

▷ NrSRGs(*parms*) (function)

Returns: An integer.

Given feasible strongly regular graph parameters *parms*, this function returns the number of pairwise non-isomorphic strongly regular graphs with parameters *parms* currently stored in this package.

Example

```
gap> NrSRGs([36,15,6,6]);
```

```
32548
gap> NrSRGs([28,9,0,4]);
0
```

5.2.7 OneSRG

▷ `OneSRG(`*parms*`)` (function)

Returns: A record or `fail`.

Given feasible strongly regular graph parameters *parms*, this function returns one strongly regular graph with parameters *parms*. If there there is no strongly regular graph with parameters *parms* available, the function returns `fail`.

Example

```
gap> OneSRG([16,9,4,6]);
rec( adjacencies := [ [ 8, 9, 10, 11, 12, 13, 14, 15, 16 ] ],
group := Group([ (6,7)(9,10)(12,13)(15,16), (5,6)(8,9)(11,12)(14,15), (2,5)
(3,6)(4,7)(9,11)(10,14)(13,15), (1,2)(5,8)(6,9)(7,10) ]), isGraph := true,
names := [ 1 .. 16 ], order := 16, representatives := [ 1 ],
schreierVector := [ -1, 4, 3, 3, 3, 2, 1, 4, 4, 4, 3, 2, 1, 3, 2, 1 ] )
gap> OneSRG([28,9,0,4]);
fail
```

5.2.8 AllSRGs

▷ `AllSRGs(`*parms*`)` (function)

Returns: A list.

Given feasible strongly regular graph parameters *parms*, this function returns a list of all pairwise non-isomorphic strongly regular graphs with parameters *parms* available in this package.

Example

```
gap> AllSRGs([16,6,2,2]);
[ rec( adjacencies := [ [ 2, 3, 4, 5, 6, 7 ] ],
  group := <permutation group with 6 generators>, isGraph := true,
  names := [ 1 .. 16 ], order := 16, representatives := [ 1 ],
  schreierVector := [ -1, 6, 4, 3, 5, 5, 5, 6, 6, 6, 4, 4, 4, 3, 3, 3 ] ),
rec( adjacencies := [ [ 2, 3, 4, 5, 6, 7 ] ], group := Group([ (3,4)(5,6)
  (8,9)(11,14)(12,13)(15,16), (2,3)(4,5)(6,7)(9,11)(10,12)(14,15), (1,2)
  (5,8)(6,9)(7,10)(11,12)(13,14) ]), isGraph := true,
  names := [ 1 .. 16 ], order := 16, representatives := [ 1 ],
  schreierVector := [ -1, 3, 2, 1, 2, 1, 2, 3, 3, 3, 2, 2, 1, 1, 2, 1 ] )]
```

5.2.9 SRGIterator

▷ SRGIterator(*parms*) (function)

Returns: An iterator.

Given feasible strongly regular graph parameters *parms*, this function returns an iterator of all pairwise non-isomorphic strongly regular graph with parameters *parms* that are stored in this package.

Example

```
gap> SRGIterator([16,6,2,2]);
<iterator>
```

5.2.10 SmallFeasibleSRGParameterTuples

▷ SmallFeasibleSRGParameterTuples(*v*) (function)

Returns: A list.

Given an integer *v*, this function returns a list of all feasible parameter tuples with at most *v* vertices, according to the list of Brouwer [Bro18a]. The list contains parameter tuples with first parameter at most AGT_Brouwer_Parameters_MAX (5.2.1).

Example

```
gap> SmallFeasibleSRGParameterTuples(16);
[ [ 5, 2, 0, 1 ], [ 9, 4, 1, 2 ], [ 10, 3, 0, 1 ], [ 10, 6, 3, 4 ],
[ 13, 6, 2, 3 ], [ 15, 6, 1, 3 ], [ 15, 8, 4, 4 ], [ 16, 5, 0, 2 ],
[ 16, 10, 6, 6 ], [ 16, 6, 2, 2 ], [ 16, 9, 4, 6 ] ]
```

5.2.11 IsEnumeratedSRGParameterTuple

▷ IsEnumeratedSRGParameterTuple(*parms*) (function)

Returns: true or false.

Given feasible strongly regular graph parameters *parms* with first parameter *v* at most AGT_Brouwer_Parameters_MAX (5.2.1), this function returns true if the strongly regular graphs with parameters *parms* have been enumerated, according to the list of Brouwer [Bro18a]. Otherwise, this function returns false.

Example

```
gap> IsEnumeratedSRGParameterTuple([37,18,8,9]);
false
gap> IsEnumeratedSRGParameterTuple([56,10,0,2]);
true
```

5.2.12 IsKnownSRGParameterTuple

▷ IsKnownSRGParameterTuple(*parms*) (function)

Returns: true or false.

Given feasible strongly regular graph parameters *parms* with first parameter *v* at most AGT_Brouwer_Parameters_MAX (5.2.1), this function returns true if it is known that there exists a strongly regular graph with parameters *parms*, according to the list of Brouwer [Bro18a]. Otherwise, this function returns false.

Example

```
gap> IsKnownSRGParameterTuple([64,28,12,12]);
true
gap> IsKnownSRGParameterTuple([64,30,18,10]);
false
gap> IsKnownSRGParameterTuple([65,32,15,16]);
false
```

5.2.13 IsAllSRGsStored

▷ IsAllSRGsStored(*parms*) (function)

Returns: true or false.

Given feasible strongly regular graph parameters *parms* with first parameter *v* at most AGT_Brouwer_Parameters_MAX (5.2.1), this function returns true if all pairwise non-isomorphic strongly regular graphs with parameters *parms* are stored in the package. Otherwise, this function returns false.

Example

```
gap> IsAllSRGsStored([37,18,8,9]);
false
gap> IsAllSRGsStored([36,15,6,6]);
true
```

5.3 Strongly regular graph constructors

In this section, we give functions to construct certain graphs, most of which are strongly regular graphs.

5.3.1 DisjointUnionOfCliques

▷ DisjointUnionOfCliques(*n1, n2, ...*) (function)

Returns: A record.

Given positive integers *n1, n2,...*, this function returns the graph consisting of the disjoint union of cliques with orders *n1, n2,...*.

Example

```
gap> DisjointUnionOfCliques(3,5,7);
rec( adjacencies := [ [ 2, 3 ], [ 5, 6, 7, 8 ], [ 10, 11, 12, 13, 14, 15 ] ],
autGroup := <permutation group with 12 generators>,
group := <permutation group with 12 generators>, isGraph := true,
```

```
isSimple := true, order := 15, representatives := [ 1, 4, 9 ],
schreierVector := [ -1, 12, 11, -2, 10, 9, 8, 7, -3, 6, 5, 4, 3, 2, 1 ] )
```

5.3.2 CompleteMultipartiteGraph

▷ `CompleteMultipartiteGraph(n1, n2, ...)` (function)

Returns: A record.

Given positive integers `n1, n2,...`, this function returns the complete multipartite graph with parts of orders `n1, n2,...`.

Let $n_1, n_2, \ldots, n_t$ be positive integers. Then the *complete multipartite graph*, $K_{n_1,n_2,\ldots,n_t}$, has vertex set that can be partitioned into t disjoint sets $X_1, X_2, \ldots, X_t$ of sizes $n_1, n_2, \ldots, n_t$ such that distinct vertices are adjacent if and only if they belong to different X_i.

Example

```
gap> CompleteMultipartiteGraph(4,2,1);
rec( adjacencies := [ [ 5, 6, 7 ], [ 1, 2, 3, 4, 7 ], [ 1, 2, 3, 4, 5, 6 ] ],
autGroup := Group([ (5,6), (3,4), (2,3), (1,2) ]), group := Group([ (5,6),
  (3,4), (2,3), (1,2) ]), isGraph := true, isSimple := true, order := 7,
representatives := [ 1, 5, 7 ],
schreierVector := [ -1, 4, 3, 2, -2, 1, -3 ] )
```

5.3.3 TriangularGraph

▷ `TriangularGraph(n)` (function)

Returns: A record.

Given an integer `n`, where $n \geq 3$, this function returns the triangular graph on `n` points.

Let n be an integer, where $n \geq 3$. The *triangular graph* , $T(n)$, has vertex set consisting of the subsets of $\{1,2,...,n\}$ of size 2, and two distinct vertices A, B are joined by an edge precisely when $|A \cap B| = 1$.

The graph $T(n)$ is strongly regular with parameters $(\binom{n}{2}, 2(n-2), n-2, 4)$. For $n \neq 8$, $T(n)$ is the unique strongly regular graph with its parameters. There are four pairwise non-isomomorphic strongly regular graphs that have the same parameters as $T(8)$, which are the triangular graph $T(8)$ and the *Chang graphs* (see [Con58] and [Cha59]).

Example

```
gap> TriangularGraph(7);
rec( adjacencies := [ [ 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 ] ],
group := Group([ (1,7,12,16,19,21,6)(2,8,13,17,20,5,11)(3,9,14,18,4,10,15),
  (2,7)(3,8)(4,9)(5,10)(6,11) ]), isGraph := true, isSimple := true,
names := [ [ 1, 2 ], [ 1, 3 ], [ 1, 4 ], [ 1, 5 ], [ 1, 6 ], [ 1, 7 ],
    [ 2, 3 ], [ 2, 4 ], [ 2, 5 ], [ 2, 6 ], [ 2, 7 ], [ 3, 4 ], [ 3, 5 ],
    [ 3, 6 ], [ 3, 7 ], [ 4, 5 ], [ 4, 6 ], [ 4, 7 ], [ 5, 6 ], [ 5, 7 ],
    [ 6, 7 ] ], order := 21, representatives := [ 1 ],
schreierVector := [ -1, 2, 2, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1 ] )
```

5.3.4 SquareLatticeGraph

▷ `SquareLatticeGraph(n)` (function)

Returns: A record.

Given an integer `n`, where `n` ≥ 2, this function returns the square lattice graph on n^2 points.

Let n be an integer, where $n \geq 2$. The *square lattice graph*, $L_2(n)$, has vertex set $\{1,2,...,n\} \times \{1,2,...,n\}$, and two distinct vertices are joined by an edge precisely when they have the same value at one coordinate.

The graph $L_2(n)$ is strongly regular with parameters $(n^2, 2(n-1), n-2, 2)$. For $n \neq 4$, $L_2(n)$ is the unique strongly regular graph with its parameters. There are two pairwise non-isomomorphic strongly regular graphs that have the same parameters as $L_2(4)$, which are the square lattice graph graph $L_2(4)$ and the *Shrikhande graph* (see [Shr59]).

Example

```
gap> SquareLatticeGraph(6);
rec( adjacencies := [ [ 2, 3, 4, 5, 6, 7, 13, 19, 25, 31 ] ],
group := <permutation group with 5 generators>, isGraph := true,
names := [ [ 1, 1 ], [ 1, 2 ], [ 1, 3 ], [ 1, 4 ], [ 1, 5 ], [ 1, 6 ],
    [ 2, 1 ], [ 2, 2 ], [ 2, 3 ], [ 2, 4 ], [ 2, 5 ], [ 2, 6 ], [ 3, 1 ],
    [ 3, 2 ], [ 3, 3 ], [ 3, 4 ], [ 3, 5 ], [ 3, 6 ], [ 4, 1 ], [ 4, 2 ],
    [ 4, 3 ], [ 4, 4 ], [ 4, 5 ], [ 4, 6 ], [ 5, 1 ], [ 5, 2 ], [ 5, 3 ],
    [ 5, 4 ], [ 5, 5 ], [ 5, 6 ], [ 6, 1 ], [ 6, 2 ], [ 6, 3 ], [ 6, 4 ],
    [ 6, 5 ], [ 6, 6 ] ], order := 36, representatives := [ 1 ],
schreierVector := [ -1, 3, 3, 3, 3, 3, 1, 3, 3, 3, 3, 3, 1, 3, 3, 3, 3, 3,
    1, 3, 3, 3, 3, 3, 1, 3, 3, 3, 3, 3, 1, 3, 3, 3, 3, 3 ] )
```

5.3.5 HoffmanSingletonGraph

▷ `HoffmanSingletonGraph()` (function)

Returns: A record.

This function returns the Hoffman-Singleton graph.

The *Hoffman-Singleton graph* is the unique strongly regular graph with parameters $(50,7,0,1)$. For more information on this graph, see [Bro18b].

Example

```
gap> gamma:=HoffmanSingletonGraph();;
```

5.3.6 HigmanSimsGraph

▷ `HigmanSimsGraph()` (function)

Returns: A record.

This function returns the Higman-Sims graph.

The *Higman-Sims graph* is the unique strongly regular graph with parameters $(100, 22, 0, 6)$. For more information on this graph, see [Bro18b].

Example

```
gap> gamma:=HigmanSimsGraph();;
```

5.3.7 SimsGerwitzGraph

▷ `SimsGerwitzGraph()` (function)

Returns: A record.

This function returns the Sims-Gerwitz graph.

The *Sims-Gerwitz graph* is the unique strongly regular graph with parameters $(56, 10, 0, 2)$. For more information on this graph, see [Bro18b].

Example

```
gap> gamma:=SimsGerwitzGraph();;
```

References

[BCN89] Andries E. Brouwer, Arjeh M. Cohen, and Arnold Neumaier. *Distance-Regular Graphs*. Springer-Verlag, Berlin, 1989. 7, 8, 10, 24, 25, 26, 27, 28

[BH11] Andries E. Brouwer and Willem H. Haemers. *Spectra of Graphs*. Springer Science & Business Media, 2011. 10, 11, 15, 16

[Bro18a] Andries E. Brouwer. Database of strongly regular graphs, 2018. https://www.win.tue.nl/ aeb/graphs/srg/srgtab.html. 28, 32, 33

[Bro18b] Andries E. Brouwer. Graph descriptions, 2018. https://www.win.tue.nl/ aeb/graphs/index.html. 35, 36

[Cha59] Li-Chien Chang. The Uniqueness and Non-Uniqueness of the Triangular Association Scheme. In *Science Record*, volume 3, pages 604–613. Peking Mathematical Society, 1959. 34

[Con58] William S. Connor. The Uniqueness of the Triangular Association Scheme. *The Annals of Mathematical Statistics*, 29(1):262–266, 1958. 34

[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/. 4

[Del73] Phillipe Delsarte. An algebraic approach to the association schemes of coding theory. *Philips Research Reports. Supplements*, 10:143–161, 1973. 16

[Eva20] Rhys J. Evans. *On regular induced subgraphs of edge-regular graphs*. PhD thesis, School of Mathematical Sciences, Queen Mary University of London, 2020. 5, 17, 18, 19, 21, 22

[GS16] Gary R. W. Greaves and Leonard H. Soicher. On the clique number of a strongly regular graph. *Electronic Journal of Combinatorics*, 25(4)(P4.15), 2016. 17

[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. 4

[Shr59] Shartchandra S. Shrikhande. The Uniqueness of the L_2 Association Scheme. *The Annals of Mathematical Statistics*, 230(3):781–798, 1959. 35

[Soi10] Leonard H. Soicher. More on block intersection polynomials and new applications to graphs and block designs. *Journal of Combinatorial Theory, Series A*, 117(7):799–809, 2010. 17, 18

[Soi15] Leonard H. Soicher. On cliques in edge-regular graphs. *Journal of Algebra*, 421:260–267, 2015. 17, 18

[Soi18] L. H. Soicher. GRAPE, graph algorithms using permutation groups, Version 4.8, 2018. Refereed GAP package, available at https://gap-packages.github.io/grape/. 4, 24, 25

[Soi19] L. H. Soicher. The DESIGN package for GAP, Version 1.7, 2019. Refereed GAP package, available at https://gap-packages.github.io/design/. 4

Index