Description of the `sage-drg` package

Janoš Vidali
Faculty of Mathematics and Physics
University of Ljubljana, 1000 Ljubljana, Slovenia
janos.vidali@fmf.uni-lj.si

Submitted: Mar 30, 2018; Accepted: Sep 27, 2018; Published: Oct 19, 2018

1 Installation

The `sage-drg` package [48] can be installed by cloning the `git` repository or extracting the ZIP file and making sure that Sage sees the `drg` directory (e.g., by starting it from the package's root directory, or by copying or linking the `drg` directory into the package library used by the copy of Python in Sage's installation directory). Once Sage is run, the package can be imported.

```
sage: import drg
```

The central class is `drg.DRGParameters`, which can be given an intersection array in the form of two lists or tuples of the same length.

```
sage: syl = drg.DRGParameters([5, 4, 2], [1, 1, 4])
sage: syl
Parameters of a distance-regular graph with intersection array
{5, 4, 2; 1, 1, 4}
```

Instead of an intersection array, parameters (k, λ, μ) for a strongly regular graph or classical parameters (d, b, α, β) (see [7, §6]) may also be specified.

```
sage: petersen = drg.DRGParameters(3, 0, 1)
sage: petersen
Parameters of a distance-regular graph with intersection array {3, 2; 1, 1}
sage: q7 = drg.DRGParameters(7, 1, 0, 1)
sage: q7
Parameters of a distance-regular graph with intersection array
{7, 6, 5, 4, 3, 2, 1; 1, 2, 3, 4, 5, 6, 7}
```

The intersection array (given in any of the forms above) may also contain variables. Substitution of variables is possible using the `subs` method. Note that the diameter must be constant.

```
sage: r = var("r")
sage: fam = drg.DRGParameters([2*r^2*(2*r+1), (2*r-1)*(2*r^2+r+1), 2*r^2],
                              [1, 2*r^2, r*(4*r^2-1)])
sage: fam1 = fam.subs(r == 1)
sage: fam1
Parameters of a distance-regular graph with intersection array
{6, 4, 2; 1, 2, 3}
```

2 Parameter computation

As a `drg.DRGParameters` object is being constructed, its intersection numbers are computed. If any of them is determined to be negative or nonintegral,[1] then an exception is thrown and the construction is aborted. Several other conditions are also checked, for example that the sequences $\{b_i\}_{i=0}^d$ and $\{c_i\}_{i=0}^d$ are non-ascending and non-descending, respectively, and that $b_j \geqslant c_i$ if $i + j \leqslant d$ [7, Prop. 4.1.6(ii)]. The handshake lemma is also checked for each subconstituent [7, Lem. 4.3.1]. If the graph is determined to be antipodal, then the covering index is also checked for integrality.

The number of vertices, their valency and the diameter of the graph can be obtained with the `order`, `valency`, and `diameter` methods.

```
sage: syl.order()
36
sage: syl.valency()
5
sage: syl.diameter()
3
```

The entire array of intersection numbers can be obtained with the `pTable` method, which returns a `drg.Array3D` object implementing a three-dimensional array.

```
sage: syl.pTable()
0: [ 1  0  0  0]
   [ 0  5  0  0]
   [ 0  0 20  0]
   [ 0  0  0 10]

1: [0 1 0 0]
```

[1] Non-constant expressions are also checked for nonnegativity – if there is no assignment of real numbers to the variables such that the value of the expression is nonnegative, then the expression is marked as negative and cannot appear as, e.g., an intersection number.

```
   [1 0 4 0]
   [0 4 8 8]
   [0 0 8 2]

2: [ 0  0  1  0]
   [ 0  1  2  2]
   [ 1  2 11  6]
   [ 0  2  6  2]

3: [ 0  0  0  1]
   [ 0  0  4  1]
   [ 0  4 12  4]
   [ 1  1  4  4]
```

The subsets of intersection numbers $\{a_i\}_{i=1}^d$, $\{b_i\}_{i=0}^{d-1}$, $\{c_i\}_{i=1}^d$, and $\{k_i\}_{i=0}^d$ (where $k_i = p_{ii}^0$ is the number of vertices at distance i from any vertex of the graph) can be obtained as tuples with the `aTable`, `bTable`, `cTable`, and `kTable` methods, respectively. There is also a method `intersectionArray` returning the entire intersection array as a pair of tuples.

```
sage: syl.aTable()
(0, 2, 1)
sage: syl.bTable()
(5, 4, 2)
sage: syl.cTable()
(1, 1, 4)
sage: syl.kTable()
(1, 5, 20, 10)
sage: syl.intersectionArray()
((5, 4, 2), (1, 1, 4))
```

Eigenvalues can be computed using the `eigenvalues` method.

```
sage: syl.eigenvalues()
(5, 2, -1, -3)
```

Eigenvalues are sorted in the decreasing order; if there is a variable in the intersection array, then the order is derived under the assumption that the variable takes a (large enough) positive value. If there is more than one variable, a warning is issued that the given ordering is not necessarily correct. This can be avoided by explicitly specifying an order of the variables using the `set_vars` method (see the jupyter/DRG-d3-2param.ipynb notebook for an example). The ordering of eigenvalues (and thus the ordering of corresponding parameters) can also be changed later using the `reorderEigenvalues` method, which accepts an ordering of the indices of the nontrivial eigenvalues (i.e., integers from 1 to d).

Once the ordering of eigenvalues is determined, the cosine sequences and multiplicities of the eigenvalues can be computed using the `cosineSequences` and `multiplicities` methods. The multiplicities are checked to be integral.

```
sage: syl.cosineSequences()
[    1     1     1     1]
[    1   2/5 −1/20  −1/5]
[    1  −1/5  −1/5   2/5]
[    1  −3/5   1/5  −1/5]
sage: syl.multiplicities()
(1, 16, 10, 9)
```

The eigenmatrix and dual eigenmatrix can be computed using the `eigenmatrix` and `dualEigenmatrix` methods. The `is_formallySelfDual` method checks whether the two matrices are equal (using the current ordering of eigenvalues).

```
sage: syl.eigenmatrix()
[ 1  5 20 10]
[ 1  2 −1 −2]
[ 1 −1 −4  4]
[ 1 −3  4 −2]
sage: syl.dualEigenmatrix()
[    1    16    10     9]
[    1  32/5    −2 −27/5]
[    1  −4/5    −2   9/5]
[    1 −16/5     4  −9/5]
sage: syl.is_formallySelfDual()
False
```

The Krein parameters can be computed using the `kreinParameters` method, which returns a `drg.Array3D` object. The Krein parameters are checked to be nonnegative.

```
sage: syl.kreinParameters()
0: [ 1  0  0  0]
   [ 0 16  0  0]
   [ 0  0 10  0]
   [ 0  0  0  9]

1: [   0    1    0    0]
   [   1 44/5 22/5  9/5]
   [   0 22/5    2 18/5]
   [   0  9/5 18/5 18/5]

2: [     0      0      1      0]
   [     0 176/25   16/5 144/25]
   [     1   16/5      4    9/5]
   [     0 144/25    9/5  36/25]

3: [   0    0    0    1]
   [   0 16/5 32/5 32/5]
```

```
[   0 32/5    2  8/5]
[   1 32/5  8/5    0]
```

Classical parameters can be computed using the `is_classical` method, which returns a list of all tuples of classical parameters, or `False` if the graph is not classical.

```
sage: fam1.is_classical()
[(3, 1, 0, 2)]
```

The method `genPoly_parameters` returns the tuple (g, s, t) if the parameters correspond to those of a collinearity graph of a generalized g-gon of order (s, t), or `(False, None, None)` if there is no such generalized g-gon. See [46] and [7, §6.5] for definitions of generalized polygons and some results.

```
sage: drg.DRGParameters([6, 4, 4], [1, 1, 3]).genPoly_parameters()
(6, 2, 2)
```

Note that the existence of a strongly regular graph for which (g, s, t) are defined does not imply the existence of a corresponding generalized quadrangle. A distance-regular graph Γ of diameter at least 3 has these parameters defined precisely when Γ is isomorphic to the collinearity graph of a corresponding generalized g-gon.

All the methods mentioned above store their results, so subsequent calls will not redo the computations. Note that one does not need to call the methods in the order given here – if some required computation has not been done before, it will be performed when needed. Where applicable, the methods above also take three named boolean parameters `expand`, `factor`, and `simplify` (all set to `False` by default), which control how the returned expression(s) will be manipulated. In the case when there are no variables in use, setting these parameters has no effect.

3 Parameters of derived graphs

In some cases, the parameters of a distance-regular graph imply the existence of another distance-regular graph which can be derived from the original graph. This is true for imprimitive graphs (i.e., antipodal or bipartite), but sometimes, new distance-regular graphs can be obtained by taking subgraphs or by merging classes.

The antipodality of a graph can be checked with the `is_antipodal` method, which returns the covering index for antipodal graphs, and `False` otherwise. The parameters of the antipodal quotient of an antipodal graph can then be obtained with the `antipodalQuotient` method.

```
sage: q7.is_antipodal()
2
sage: q7.antipodalQuotient()
Parameters of a distance-regular graph with intersection array
{7, 6, 5; 1, 2, 3}
```

The bipartiteness of a graph can be checked with the `is_bipartite` method. The parameters of the bipartite half of a bipartite graph can then be obtained with the `bipartiteHalf` method.

```
sage: q7.is_bipartite()
True
sage: q7.bipartiteHalf()
Parameters of a distance-regular graph with intersection array
{21, 10, 3; 1, 6, 15}
```

In some cases, distance-regularity of the local graph can be established (for instance, for tight graphs, see [31]). In these cases, the parameters of the local graph can then be obtained with the `localGraph` method.

```
sage: drg.DRGParameters([27, 10, 1], [1, 10, 27]).localGraph()
Parameters of a distance-regular graph with intersection array
{16, 5; 1, 8}
```

Similarly, the distance-regularity of a subconstituent (i.e., a graph induced by vertices at a given distance from a vertex) can be established in certain cases. Their parameters can be obtained using the `subconstituent` method. Usually, distance-regularity is derived from triple intersection numbers (see Section 5), which are not computed by default. To force this computation, the parameter `compute` can be set to `True`.

```
sage: drg.DRGParameters([204, 175, 48, 1],
                        [1, 12, 175, 204]).subconstituent(2,compute = True)
Parameters of a distance-regular graph with intersection array
{144, 125, 32, 1; 1, 8, 125, 144}
```

Note that calling `localGraph()` is equivalent to calling `subconstituent(1)`. The `localGraph` method also accepts the `compute` parameter.

The complement of a strongly regular graph is also strongly regular. If the complement is connected, its parameters can be obtained with the `complementaryGraph` method.

```
sage: petersen.complementaryGraph()
Parameters of a distance-regular graph with intersection array {6, 2; 1, 4}
```

Sometimes, merging classes of the underlying association scheme yields a new distance-regular graph. Its parameters (or the parameters of a connected component if the resulting graph is disconnected) can be obtained with the `mergeClasses` method, which takes the indices of classes which will be merged into the first class of the new scheme (i.e., the distances in the original graph which will correspond to adjacency in the new graph).

```
sage: q7.mergeClasses(2, 3, 6)
Parameters of a distance-regular graph with intersection array
{63, 30, 1; 1, 30, 63}
```

Note that `mergeClasses(2)` gives the parameters of the bipartite half for bipartite graphs, and of the complement for non-antipodal strongly regular graphs.

A dictionary mapping the merged indices to parameters of a new graphs for all possibilities can be obtained using the `distanceGraphs` method.

```
sage: q7.distanceGraphs()
{(1, 2): Parameters of a distance-regular graph with intersection array
         {28, 15, 6, 1; 1, 6, 15, 28},
 (1, 2, 3, 4, 5, 6): Parameters of a distance-regular graph
                     with intersection array {126, 1; 1, 126},
 (1, 3, 5): Parameters of a distance-regular graph with intersection array
            {63, 62, 1; 1, 62, 63},
 (1, 3, 5, 7): Parameters of a distance-regular graph
               with intersection array {64, 63; 1, 64},
 (1, 4, 5): Parameters of a distance-regular graph with intersection array
            {63, 32, 1; 1, 32, 63},
 (1, 5): Parameters of a distance-regular graph with intersection array
        {28, 27, 16; 1, 12, 28},
 (1, 7): Parameters of a distance-regular graph with intersection array
         {8, 7, 6, 5; 1, 2, 3, 8},
 (2,): Parameters of a distance-regular graph with intersection array
       {21, 10, 3; 1, 6, 15},
 (2, 3, 6): Parameters of a distance-regular graph with intersection array
            {63, 30, 1; 1, 30, 63},
 (2, 4, 6): Parameters of a distance-regular graph with intersection array
            {63; 1},
 (2, 6): Parameters of a distance-regular graph with intersection array
         {28, 15; 1, 12},
 (3, 7): Parameters of a distance-regular graph with intersection array
         {36, 35, 16; 1, 20, 36},
 (4,): Parameters of a distance-regular graph with intersection array
       {35, 16; 1, 20},
 (6,): Parameters of a distance-regular graph with intersection array
       {7, 6, 5; 1, 2, 3},
 (7,): Parameters of a distance-regular graph with intersection array
       {1; 1}}
```

4 Feasibility checking

To check whether a given parameter set is feasible, the `check_feasible` method may be called. This method calls other `check_*` methods which perform the actual checks. Selected checks may also be skipped by providing a parameter `skip` with a list of strings identifying checks to be skipped.

sporadic. The `check_sporadic` method checks whether the intersection array matches one from a list of intersection arrays for which nonexistence of a corresponding graph

has been proven, but does not belong to any infinite family. If so, the parameter set is reported as infeasible. Currently, the list includes:

- $\{14, 12; 1, 4\}$, cf. Wilbrink and Brouwer [51],
- $\{16, 12; 1, 6\}$, cf. Bussemaker et al. [11],
- $\{21, 18; 1, 7\}$, cf. Haemers [26],
- $\{30, 21; 1, 14\}$, cf. Bondarenko, Prymak and Radchenko [5],
- $\{32, 21; 1, 16\}$, cf. Azarija and Marc [2],
- $\{38, 27; 1, 18\}$, cf. Degraer [17],
- $\{40, 27; 1, 20\}$, cf. Azarija and Marc [1],
- $\{57, 56; 1, 12\}$, cf. Gavrilyuk and Makhnev [21],
- $\{67, 56; 1, 2\}$, cf. Brouwer and Neumaier [9],
- $\{116, 115; 1, 20\}$, cf. Makhnev [39],
- $\{153, 120; 1, 60\}$, cf. Bondarenko et al. [4],
- $\{165, 128; 1, 66\}$, cf. Makhnev [38],
- $\{486, 320; 1, 243\}$, cf. Makhnev [38],
- $\{5, 4, 3; 1, 1, 2\}$, cf. Fon-Der-Flaass [19],
- $\{11, 10, 10; 1, 1, 11\}$ (projective plane of order 10), cf. Lam, Thiel and Swiercz [36],
- $\{13, 10, 7; 1, 2, 7\}$, cf. Coolsaet [12],
- $\{18, 12, 1; 1, 2, 18\}$ (generalized quadrangle of order $(6, 3)$ minus a spread), cf. [7, Prop. 12.5.2] and [43, 6.2.2],
- $\{20, 10, 10; 1, 1, 2\}$ (projective plane of order 10), cf. Lam, Thiel and Swiercz [36],
- $\{21, 16, 8; 1, 4, 14\}$, cf. Coolsaet [13],
- $\{22, 16, 5; 1, 2, 20\}$, cf. Sumalroj and Worawannotai [45],
- $\{27, 20, 10; 1, 2, 18\}$, cf. Brouwer, Sumalroj and Worawannotai [10],
- $\{36, 28, 4; 1, 2, 24\}$, cf. Brouwer, Sumalroj and Worawannotai [10],
- $\{39, 24, 1; 1, 4, 39\}$, cf. Bang, Gavrilyuk and Koolen [3],
- $\{45, 30, 7; 1, 2, 27\}$, cf. Gavrilyuk and Makhnev [23],
- $\{52, 35, 16; 1, 4, 28\}$, cf. Gavrilyuk and Makhnev [22],
- $\{55, 36, 11; 1, 4, 45\}$, cf. Gavrilyuk [20],
- $\{56, 36, 9; 1, 3, 48\}$, cf. Gavrilyuk [20],
- $\{69, 48, 24; 1, 4, 46\}$, cf. Gavrilyuk and Makhnev [22],
- $\{74, 54, 15; 1, 9, 60\}$, cf. Coolsaet and Jurišić[14],
- $\{105, 102, 99; 1, 2, 35\}$, cf. De Bruyn and Vanhove [16],

- $\{130, 96, 18; 1, 12, 117\}$, cf. Jurišić and Vidali [33],
- $\{135, 128, 16; 1, 16, 120\}$, see [49, Theorem 4],
- $\{234, 165, 12; 1, 30, 198\}$, see [49, Theorem 5],
- $\{4818, 4248, 192; 1, 72, 4672\}$, cf. Jurišić and Vidali [33],
- $\{5928, 5920, 5888; 1, 5, 741\}$, cf. De Bruyn and Vanhove [16],
- $\{120939612, 120939520, 120933632; 1, 65, 1314561\}$, cf. De Bruyn and Vanhove [16],
- $\{97571175, 97571080, 97569275; 1, 20, 1027065\}$, cf. De Bruyn and Vanhove [16],
- $\{290116365, 290116260, 290100825; 1, 148, 2763013\}$, cf. De Bruyn and Vanhove [16],
- $\{5, 4, 3, 3; 1, 1, 1, 2\}$, cf. Fon-Der-Flaass [18],
- $\{10, 9, 1, 1; 1, 1, 9, 10\}$, cf. [7, Prop. 11.4.5],
- $\{32, 27, 6, 1; 1, 6, 27, 32\}$, cf. Soicher [44],
- $\{32, 27, 9, 1; 1, 3, 27, 32\}$, cf. Soicher [44],
- $\{56, 45, 20, 1; 1, 4, 45, 56\}$, cf. [7, Prop. 11.4.5],
- $\{55, 54, 50, 35, 10; 1, 5, 20, 45, 55\}$, see [49, Theorem 7], and
- $\{15, 14, 12, 6, 1, 1; 1, 1, 3, 12, 14, 15\}$, cf. Ivanov and Shpectorov [28].

`family`. The `check_family` method checks whether the intersection array matches one from a list of infinite families of intersection arrays for which nonexistence of corresponding graphs has been proven. If so, the parameter set is reported as infeasible. Currently, the list includes:

- $\{r^2(r+3), (r+1)(r^2+2r-2); 1, r(r+1)\}$ with $r \geqslant 3$, $r \neq 4$, cf. Bondarenko and Radchenko [6],
- $\{(2r^2-1)(2r+1), 4r(r^2-1), 2r^2; 1, 2(r^2-1), r(4r^2-2)\}$ with $r \geqslant 2$, cf. Jurišić and Vidali [32],
- $\{2r^2(2r+1), (2r-1)(2r^2+r+1), 2r^2; 1, 2r^2, r(4r^2-1)\}$ with $r \geqslant 2$, cf. Jurišić and Vidali [32],
- $\{4r^3+8r^2+6r+1, 2r(r+1)(2r+1), 2r^2+2r+1; 1, 2r(r+1), (2r+1)(2r^2+2r+1)\}$ with $r \geqslant 1$, cf. Coolsaet and Jurišić [14],
- $\{(2r+1)(4r+1)(4t-1), 8r(4rt-r+2t), (r+t)(4r+1); 1, (r+t)(4r+1), 4r(2r+1)(4t-1)\}$ with $r, t \geqslant 1$, see [49, Theorem 3],
- $\{(r+1)(r^3-1), r(r-1)(r^2+r-1), r^2-1; 1, r(r+1), (r^2-1)(r^2+r-1)\}$ with $r \geqslant 3$, cf. Urlep [47],
- $\{r^2(rt+t+1), (r^2-1)(rt+1), r(r-1)(t+1), 1; 1, r(t+1), (r^2-1)(rt+1), r^2(rt+t+1)\}$ with $r \geqslant 3$ and $(r, t) \neq (3, 1), (3, 3), (4, 2)$, cf. Jurišić and Koolen [30], and
- $\{2r^2+r, 2r^2+r-1, r^2, r, 1; 1, r, r^2, 2r^2+r-1, 2r^2+r\}$ with $r \geqslant 2$, cf. Coolsaet, Jurišić and Koolen [15].

2graph. The check_2graph method checks conditions related to two-graphs, cf. [7, Thm. 1.5.6]. For strongly regular graphs with parameters (v, k, λ, μ), for which $v = 2(2k - \lambda - \mu)$ holds, it records the parameters $(v - 1, 2(k - \mu), k + \lambda - 2\mu, k - \mu)$ of a strongly regular graph to be checked for feasibility later. For Taylor graphs (i.e., antipodal double covers of diameter 3) for which $a_1 > 0$ holds, it checks whether a_1 is even and whether the number of vertices n is a multiple of 4, and then records the parameters $(k, a_1, (3a_1 - k - 1)/2, a_1/2)$ of a strongly regular graph as those af the local graph to be checked for feasibility later, cf. [7, Thm. 1.5.3].

classical. The check_classical method checks whether any of the classical parameters for the parameter set match some from a list of infinite families of classical paramters for which nonexistence of corresponding graphs has been proven. If so, the parameter set is reported as infeasible. Currently, the list only includes two sets of classical parameters:

- $(d, b, \alpha, \beta) = (d, -2, -2, ((-2)^{d+1} - 1)/3)$ with $d \geqslant 4$, cf. Huang, Pan and Weng [27], and
- $(d, b, \alpha, \beta) = (d, -r, -r/(r-1), r + r^2((-r)^{d-1} - 1)/(r^2 - 1))$ with $d \geqslant 4$, $r \geqslant 2$, cf. De Bruyn and Vanhove [16].

Additionally, the method checks whether nonexistence can be derived from one of the following characterizations:

- a characterization of Grassmann graphs by Metsch [40, Thm. 2.3],
- a characterization of bilinear forms graphs by Metsch [41, Prop. 2.2],
- a characterization of graphs with classical parameters (d, b, α, β) and $d \geqslant 4$, $b < 0$ by Weng [50, Thm. 10.3], and
- a characterization of graphs with classical parameters (d, b, α, β) and $d \geqslant 3$, $a_1 = 0$, $a_2 > 0$ by Pan and Weng [42, Thm. 2.1].

combinatorial. The check_combinatorial method checks various combinatorial conditions:

- a graph with $b_1 = 1$ must be a cycle or a cocktail party graph,
- Godsil's diameter bound [7, Lem. 5.3.1],
- a lower bound for c_3 [7, Thm. 5.4.1],
- a condition for $b_1 = b_i$ [7, Prop. 5.4.4],
- a lower bound for a_1 [7, Prop. 5.5.1],
- a handshake lemma for Pappus subgraphs, cf. Koolen [34]
- TurÃ¡n's theorem [7, Lem. 5.6.4],
- a counting argument by Lambeck [37],
- two lower bounds for the size of the last subconstituent [7, Props. 5.6.1, 5.6.3],
- handshake lemmas for the numbers of edges and triangles [7, Lem. 4.3.1], and

- a condition for $p_{dd}^2 = 0$ [7, Prop. 5.7.1].

Additionally, the method checks whether a condition for the existence of cliques of size $a_1 + 2$ is satisified [7, Prop. 4.3.2] and performs additional checks:

- a divisibility check if the last subconstituent is a union of cliques [7, Prop. 4.3.2(ii)],
- a handshake lemma for maximal cliques [7, Prop. 4.3.3], and
- two inequalities from counting arguments [7, Prop. 4.3.3].

`conference`. For a strongly regular graph with parameters (v, k, λ, μ), where $k = 2\mu$ and $\mu = k - \lambda - 1$, the `check_conference` method checks whether $n \not\equiv 1 \pmod 4$ and n is a sum of two squares, cf. [7, §1.3].

`geodeticEmbedding`. For a distance-regular graph with intersection array $\{2b, b, 1; 1, 1, 2b\}$, the `check_geodeticEmbedding` method checks whether $b \leqslant 4$, cf. [7, Prop. 1.17.3].

`2design`. For a distance-regular graph with intersection array $\{r\mu + 1, (r-1)\mu, 1; 1, \mu, r\mu + 1\}$, the `check_2design` method checks that a corresponding 2-design exists, cf. [7, Prop. 1.10.5].

`hadamard`. For a distance-regular graph with intersection array $\{2\mu, 2\mu - 1, \mu, 1; 1, \mu, 2\mu - 1, 2\mu\}$ with $\mu > 1$, the `check_hadamard` method checks whether μ is even, i.e., whether a Hadamard matrix of order 2μ can exist, cf. [7, Cor. 1.8.2].

`antipodal`. For an antipodal cover of even diameter at least 4, the `check_antipodal` method checks whether its quotient satisfies necessary conditions for the existence of a cover, cf. [7, Prop. 4.2.7].

`genPoly`. The `check_genPoly` method checks conditions related to generalized polygons. First, it checks whether the conditions for the existence of cliques of size $a_1 + 2$ have been checked, and calls `check_combinatorial` otherwise to obtain this information. If the existence of cliques has been established and the intersection array matches that of a collinearity graph of a generalized g-gon with parameter (s, t), then we know that we should indeed have such a graph, and the following conditions are checked:

- Feit-Higman theorem [7, Thm. 6.5.1],
- $(s + t) | st(s + 1)(t + 1)$ for generalized quadrangles [43, 1.2.2], and
- Bruck-Ryser theorem for thin generalized hexagons [7, Thm. 1.10.4].

An antipodal distance-regular r-cover of diameter 3 with $k = (r - 1)(c_2 + 1)$ such that the existence of cliques of size $a_1 + 2$ has been established corresponds to the collinearity graph of a generalized quadrangle of order $(s, t) = (r - 1, c_2 + 1)$ with a spread removed. Therefore, in this case, `check_genPoly` checks for feasibility of such a generalized quadrangle (first two conditions above), and also checks whether it can contain a spread, see [7, Prop. 12.5.2] and [43, 1.8.3].

`clawBound`. For a strongly regular graph, the `check_clawBound` method checks the claw bound, i.e., whether the graph must be the point graph of an infeasible partial geometry, cf. Brouwer and Van Lint [8].

terwilliger. The check_terwilliger method checks conditions related to Terwilliger graphs and induced quadrangles, cf. [7, §1.16]. First, it checks the coclique bound by Koolen and Park [35, Thm. 3]; if it is met with equality, it checks wheter the graph can be a Terwilliger graph [7, Cor. 1.16.6]. If the parameters imply that the graph contains an induced quadrangle, then Terwilliger's diameter bound is checked [7, Thm. 5.2.1].

secondEigenvalue. For a distance-regular graph with an eigenvalue equal to $b_1 - 1$, the check_secondEigenvalue method checks whether the graph belongs to the classification given in [7, Thm. 4.4.11].

localEigenvalue. The check_localEigenvalue method checks conditions related to the eigenvalues of the local graph. For a distance-regular graph of diameter 3 that is not the dodecahedron, the following conditions are checked:

- general bounds for the second largest and smallest eigenvalue of the local graph [7, Thm. 4.4.3],
- a bound on eigenvalues of the local graph of a non-bipartite graph by Jurišić and Koolen [29],
- the fundamental bound by Jurišić, Koolen and Terwilliger [31], and
- bounds on multiplicities of eigenvalues, see [7, Thm. 4.4.4], [24] and [25].

If equality is met in the bounds of the second or third point above, then the local graph is determined to be strongly regular and is thus stored as such to be checked for feasibility later.

absoluteBound. The check_absoluteBound method checks the absolute bound on the multiplicities of eigenvalues [7, Thm. 2.3.3].

After running all the checks described above, the check_feasbile method calls itself on all already derived graphs (antipodal quotient, bipartite half, complement, 2-graph derivation, subconstituents where applicable), and then also on each parameter set for distance-regular graphs obtained by merging classes. To avoid repetitions, a list of checked intersection arrays is maintained. This step can be skipped by setting the derived parameter to False.

If the parameter set is feasible (i.e., it passes all checks), then check_feasible returns without error. Otherwise, a drg.InfeasibleError exception is thrown indicating the reason for nonexistence and providing a reference.

```
sage: drg.DRGParameters(266, 220, 210).check_feasible()
...
InfeasibleError: complement: nonexistence by GavrilyukMakhnev05
```

Details on the given references are available in the drg.references submodule.

```
sage: import drg.references
sage: drg.references.refs["GavrilyukMakhnev05"]
{'authors': [('Gavrilyuk', ('Alexander', 'L.')),
```

```
            ('Makhnev', ('Alexander', 'Alexeevich'))],
 'fjournal': 'Doklady Akademii Nauk',
 'journal': 'Dokl. Akad. Nauk',
 'number': 6,
 'pages': (727, 730),
 'title': 'Krein graphs without triangles',
 'type': 'article',
 'volume': 403,
 'year': 2005}
```

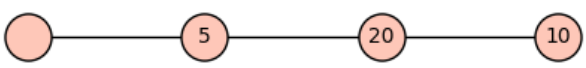
Details on the nonexistence may also be extracted from the exception.

```
sage: try:
....:     drg.DRGParameters([65, 44, 11], [1, 4, 55]).check_feasible()
....: except drg.InfeasibleError as ex:
....:     print("Part: %s" % (ex.part, ))
....:     print("Reason: %s" % ex.reason)
....:     for r, thm in ex.refs:
....:         ref = drg.references.refs[r]
....:         print("Authors: %s" % ref["authors"])
....:         print("Title: %s" % ref["title"])
....:         print("Theorem: %s" % thm)
....:
Part: ()
Reason: coclique bound exceeded
Authors: [('Koolen', ('Jack', 'H.')), ('Park', ('Jongyook',))]
Title: Shilla distance–regular graphs
Theorem: Thm. 3.
```

5 Partitions and triple intersection numbers

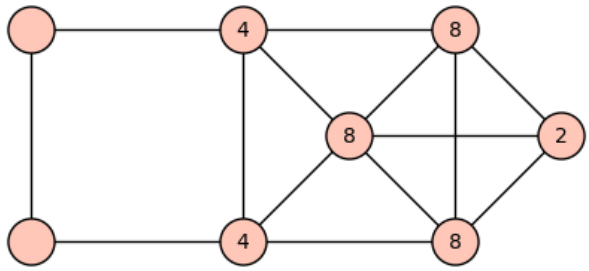
For a given parameter set, the distance partition corresponding to a vertex can be obtained with the `distancePartition` method, which returns a graph representing the distance partition.

```
sage: dp = syl.distancePartition()
sage: dp
Distance partition of {5, 4, 2; 1, 1, 4}
sage: dp.show()
```



A distance partition corresponding to two vertices can be obtained by passing the distance between them as an argument to `distancePartition`.

```
sage: syl.distancePartition(1).show()
```



Note that edges are shown between any pair of cells such that their distances from either of the initial vertices differ by at most 1. To show all the distance partitions corresponding to at most two vertices, the `show_distancePartitions` method may be used (see below).

For a given triple of distances (U, V, W) such that $p_{UV}^W > 0$, the method `tripleEquations` gives the solution to the system of equations [49, (1)] augmented by equations derived from [49, Theorem 1] for each triple (i, j, h) $(1 \leqslant i, j, h \leqslant d)$ such that $q_{ij}^h = 0$. The solution is returned as a `drg.Array3D` object.

```
sage: syl.tripleEquations(1, 1, 2)
0: [0 0 0 0]
   [0 1 0 0]
   [0 0 0 0]
   [0 0 0 0]

1: [0 0 1 0]
   [0 0 0 0]
   [1 0 3 0]
   [0 0 0 0]

2: [0 0 0 0]
   [0 0 2 2]
   [0 2 2 4]
   [0 2 4 2]

3: [0 0 0 0]
   [0 0 0 0]
   [0 0 6 2]
   [0 0 2 0]
```

If the solution is not unique, one or more parameters will be present in the solution.

```
sage: syl.tripleEquations(1, 2, 3)
0: [0 0 0 0]
   [0 0 1 0]
   [0 0 0 0]
   [0 0 0 0]
```

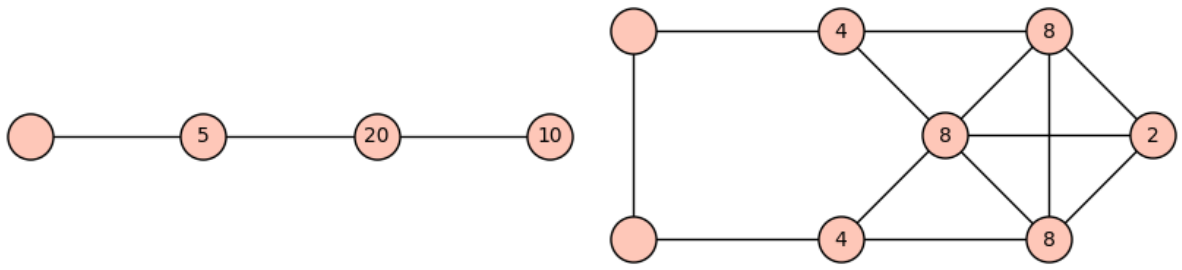
```
1: [0 0 0 1]
   [0 0 0 0]
   [0 1 2 1]
   [0 0 0 0]

2: [         0          0          0          0]
   [         0          0          3          1]
   [         0         r2     r2 + 4 -2*r2 + 4]
   [         1   -r2 + 2   -r2 + 4  2*r2 + 1]

3: [         0          0          0          0]
   [         0          0          0          0]
   [         0   -r2 + 3   -r2 + 6  2*r2 - 1]
   [         0     r2 - 1         r2 -2*r2 + 3]
```

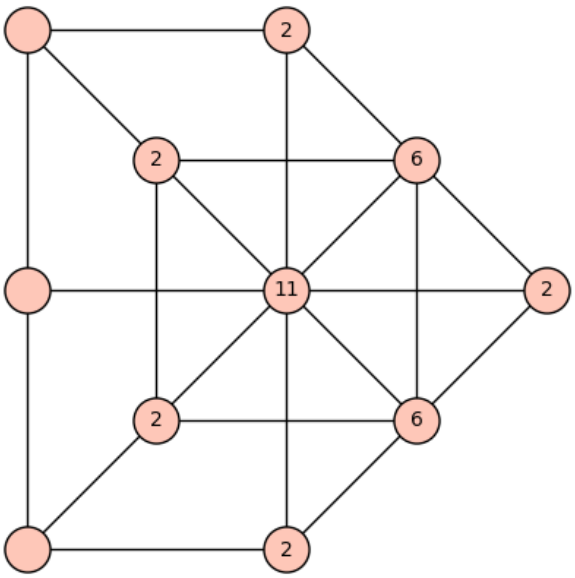
Parameters may also be set explicitly by passing a `params` argument with a dictionary mapping the name of the parameter to the triple of distances it represents.

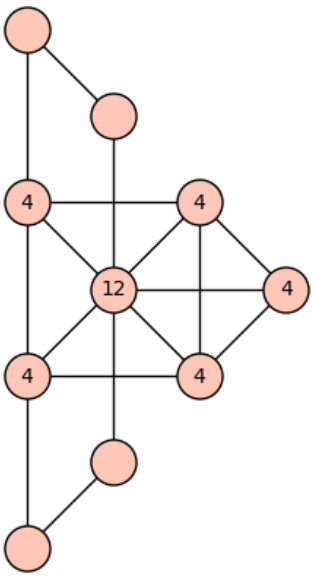
```
sage: syl.tripleEquations(1, 3, 3, params = {"a": (3, 3, 3)})
0: [0 0 0 0]
   [0 0 0 1]
   [0 0 0 0]
   [0 0 0 0]

1: [0 0 0 1]
   [0 0 0 0]
   [0 0 4 0]
   [0 0 0 0]

2: [          0           0           0           0]
   [          0           0           4           0]
   [          0 -1/2*a + 4 -1/2*a + 4           a]
   [          0       1/2*a  1/2*a + 4     -a + 4]

3: [          0           0           0           0]
   [          0           0           0           0]
   [          0       1/2*a  1/2*a + 4     -a + 4]
   [          1 -1/2*a + 1      -1/2*a           a]
```

If the appropriate triple intersection number is computed to be zero, an edge will not be shown in the distance partition.

```
sage: syl.show_distancePartitions()
```







References

[1] J. Azarija and T. Marc. There is no (95,40,12,20) strongly regular graph, 2016. arXiv:1603.02032.

[2] J. Azarija and T. Marc. There is no (75,32,10,16) strongly regular graph. *Linear Algebra Appl.*, 557:62–83, 2018. doi:10.1016/j.laa.2018.07.019.

[3] S. Bang, A. L. Gavrilyuk, and J. H. Koolen. Distance-regular graphs without 4-claws. *European J. Combin.*, 2018. doi:10.1016/j.ejc.2018.02.022.

[4] A. V. Bondarenko, A. Mellit, A. V. Prymak, D. V. Radchenko, and M. S. Viazovska. There is no strongly regular graph with parameters (460, 153, 32, 60). In *Contemporary computational mathematics–a celebration of the 80th birthday of Ian Sloan*, volume 1–2, pages 131–134. Springer, Cham, 2018. doi:10.1007/978-3-319-72456-0_7.

[5] A. V. Bondarenko, A. V. Prymak, and D. V. Radchenko. Non-existence of $(76, 30, 8, 14)$ strongly regular graph, 2017. doi:10.1016/j.laa.2017.03.033.

[6] A. V. Bondarenko and D. V. Radchenko. On a family of strongly regular graphs with $\lambda = 1$. *J. Combin. Theory Ser. B*, 103(4):521–531, 2013. doi:10.1016/j.jctb.2013.05.005.

[7] A. E. Brouwer, A. M. Cohen, and A. Neumaier. *Distance-regular graphs*, volume 18 of *Ergebnisse der Mathematik und ihrer Grenzgebiete (3) [Results in Mathematics and Related Areas (3)]*. Springer-Verlag, Berlin, 1989. doi:10.1007/978-3-642-74341-2.

[8] A. E. Brouwer and J. H. van Lint. Strongly regular graphs and partial geometries. In *Enumeration and design (Waterloo, Ont., 1982)*, pages 85–122. Academic Press, Toronto, 1984. https://ir.cwi.nl/pub/1817/1817A.pdf.

[9] A. E. Brouwer and A. Neumaier. *Strongly regular graphs where mu equals two and lambda is large.* Afdeling Zuivere Wiskunde [Department of Pure Mathematics]. Stichting Mathematisch Centrum, Amsterdam, 1981. https://ir.cwi.nl/pub/6792/6792A.pdf.

[10] A. E. Brouwer, S. Sumalroj, and C. Worawannotai. The nonexistence of distance-regular graphs with intersection arrays $\{27, 20, 10; 1, 2, 18\}$ and $\{36, 28, 4; 1, 2, 24\}$. *Australas. J. Combin.*, 66:330–332, 2016. http://ajc.maths.uq.edu.au/pdf/66/ajc_v66_p330.pdf.

[11] F. C. Bussemaker, W. H. Haemers, R. A. Mathon, and H. A. Wilbrink. A $(49, 16, 3, 6)$ strongly regular graph does not exist. *European J. Combin.*, 10(5):413–418, 1989. doi:10.1016/S0195-6698(89)80014-9.

[12] K. Coolsaet. Local structure of graphs with $\lambda = \mu = 2$, $a_2 = 4$. *Combinatorica*, 15(4):481–487, 1995. doi:10.1007/BF01192521.

[13] K. Coolsaet. A distance regular graph with intersection array $(21, 16, 8; 1, 4, 14)$ does not exist. *European J. Combin.*, 26(5):709–716, 2005. doi:10.1016/j.ejc.2004.04.005.

[14] K. Coolsaet and A. Jurišić. Using equality in the Krein conditions to prove nonexistence of certain distance-regular graphs. *J. Combin. Theory Ser. A*, 115(6):1086–1095, 2008. doi:10.1016/j.jcta.2007.12.001.

[15] K. Coolsaet, A. Jurišić, and J. Koolen. On triangle-free distance-regular graphs with an eigenvalue multiplicity equal to the valency. *European J. Combin.*, 29(5):1186–1199, 2008. doi:10.1016/j.ejc.2007.06.010.

[16] B. De Bruyn and F. Vanhove. On Q-polynomial regular near $2d$-gons. *Combinatorica*, 35(2):181–208, 2015. doi:10.1007/s00493-014-3039-x.

[17] J. Degraer. *Isomorph-free exhaustive generation algorithms for association schemes.* PhD thesis, Ghent University, 2007. https://cage.ugent.be/geometry/Theses/52/degraer-phd.pdf.

[18] D. G. Fon-Der-Flaass. A distance-regular graph with intersection array $(5, 4, 3, 3; 1, 1, 1, 2)$ does not exist. *J. Algebraic Combin.*, 2(1):49–56, 1993. doi:10.1023/A:1022476614402.

[19] D. G. Fon-Der-Flaass. There exists no distance-regular graph with intersection array $(5, 4, 3; 1, 1, 2)$. *European J. Combin.*, 14(5):409–412, 1993. doi:10.1006/eujc.1993.1045.

[20] A. L. Gavrilyuk. Distance-regular graphs with intersection arrays $\{55, 36, 11; 1, 4, 45\}$ and $\{56, 36, 9; 1, 3, 48\}$ do not exist. *Dokl. Akad. Nauk*, 439(1):14–17, 2011. doi:10.1134/S1064562411040028.

[21] A. L. Gavrilyuk and A. A. Makhnev. Krein graphs without triangles. *Dokl. Akad. Nauk*, 403(6):727–730, 2005.

[22] A. L. Gavrilyuk and A. A. Makhnev. Distance-regular graphs with intersection arrays $\{52, 35, 16; 1, 4, 28\}$ and $\{69, 48, 24; 1, 4, 46\}$ do not exist. *Des. Codes Cryptogr.*, 65(1–2):49–54, 2012. doi:10.1007/s10623-012-9695-1.

[23] A. L. Gavrilyuk and A. A. Makhnev. A distance-regular graph with intersection array $\{45, 30, 7; 1, 2, 27\}$ does not exist. *Diskret. Mat*, 25(2):13–30, 2013.

[24] C. D. Godsil and A. D. Hensel. Distance regular covers of the complete graph. *J. Combin. Theory Ser. B*, 56(2):205–238, 1992. doi:10.1016/0095-8956(92)90019-T.

[25] C. D. Godsil and J. H. Koolen. On the multiplicity of eigenvalues of distance-regular graphs. *Linear Algebra Appl.*, 226–228:273–275, 1995. doi:10.1016/0024-3795(95)00152-H.

[26] W. H. Haemers. There exists no $(76, 21, 2, 7)$ strongly regular graph. In *Finite geometry and combinatorics*, London Mathematical Society Lecture Note Series, pages 175–176. Cambridge Univ. Press, Cambridge, 1993. doi:10.1017/CBO9780511526336.018.

[27] Y. Huang, Y. Pan, and C. Weng. Nonexistence of a class of distance-regular graphs. *Electron. J. Combin.*, 22(2):2.37, 2015. http://www.combinatorics.org/ojs/index.php/eljc/article/view/v22i2p37.

[28] A. A. Ivanov and S. V. Shpectorov. The P-geometry for M_{23} has no nontrivial 2-coverings. *European J. Combin.*, 11(4):373–379, 1990. doi:10.1016/S0195-6698(13)80139-4.

[29] A. Jurišić and J. Koolen. Nonexistence of some antipodal distance-regular graphs of diameter four. *European J. Combin.*, 21(8):1039–1046, 2000.

[30] A. Jurišićand J. Koolen. Classification of the family AT4(qs, q, q) of antipodal tight graphs. *J. Combin. Theory Ser. A*, 118(3):842–852, 2011.

[31] A. Jurišić, J. Koolen, and P. Terwilliger. Tight distance-regular graphs. *J. Algebraic Combin.*, 12(2):163–197, 2000. doi:10.1023/A:1026544111089.

[32] A. Jurišić and J. Vidali. Extremal 1-codes in distance-regular graphs of diameter 3. *Des. Codes Cryptogr.*, 65(1–2):29–47, 2012. doi:10.1007/s10623-012-9651-0.

[33] A. Jurišić and J. Vidali. Restrictions on classical distance-regular graphs. *J. Algebraic Combin.*, 46(3–4):571–588, 2017. doi:10.1007/s10801-017-0765-3.

[34] J. H. Koolen. A new condition for distance-regular graphs. *European J. Combin.*, 13(1):63–64, 1992. doi:10.1016/0195-6698(92)90068-B.

[35] J. H. Koolen and J. Park. Shilla distance-regular graphs. *European J. Combin.*, 31(8):2064–2073, 2010. doi:10.1016/j.ejc.2010.05.012.

[36] C. W. H. Lam, L. Thiel, and S. Swiercz. The nonexistence of finite projective planes of order 10. *Canad. J. Math.*, 41(6):1117–1123, 1989. doi:10.4153/CJM-1989-049-4.

[37] E. Lambeck. Some elementary inequalities for distance-regular graphs. *European J. Combin.*, 14(1):53–54, 1993. doi:10.1006/eujc.1993.1008.

[38] A. A. Makhnev. On the nonexistence of strongly regular graphs with the parameters $(486, 165, 36, 66)$. *Ukrain. Mat. Zh.*, 54(7):941–949, 2002. doi:10.1023/A:1022066425998.

[39] A. A. Makhnev. The graph Kre(4) does not exist. *Dokl. Math.*, 96(1):348–350, 2017. doi:10.1134/S1064562417040123.

[40] K. Metsch. A characterization of Grassmann graphs. *European J. Combin.*, 16(6):639–644, 1995. doi:10.1016/0195-6698(95)90045-4.

[41] K. Metsch. On a characterization of bilinear forms graphs. *European J. Combin.*, 20(4):293–306, 1999. doi:10.1006/eujc.1998.0280.

[42] Y. Pan and C. Weng. A note on triangle-free distance-regular graphs with $a_2 \neq 0$. *J. Combin. Theory Ser. B*, 99(1):266–270, 2009. doi:10.1016/j.jctb.2008.07.005.

[43] S. E. Payne and J. A. Thas. *Finite generalized quadrangles*. EMS Series of Lectures in Mathematics. European Mathematical Society (EMS), Zürich, second edition, 2009. doi:10.4171/066.

[44] L. H. Soicher. The uniqueness of a distance-regular graph with intersection array $\{32, 27, 8, 1; 1, 4, 27, 32\}$ and related results. *Des. Codes Cryptogr.*, 84(1–2):101–108, 2017. doi:10.1007/s10623-016-0223-6.

[45] S. Sumalroj and C. Worawannotai. The nonexistence of a distance-regular graph with intersection array $\{22, 16, 5; 1, 2, 20\}$. *Electron. J. Combin*, 23(1):#P1.32, 2016. http://www.combinatorics.org/ojs/index.php/eljc/article/view/v23i1p32.

[46] J. Tits. Sur la trialité et certains groupes qui s'en déduisent. *Inst. Hautes Études Sci. Publ. Math.*, (2):13–60, 1959. http://www.numdam.org/item?id=PMIHES_1959__2__13_0.

[47] M. Urlep. Triple intersection numbers of Q-polynomial distance-regular graphs. *European J. Combin.*, 33(6):1246–1252, 2012. doi:10.1016/j.ejc.2012.02.005.

[48] J. Vidali. `jaanos/sage-drg`: `sage-drg` Sage package v0.8, 2018. https://github.com/jaanos/sage-drg/, doi:10.5281/zenodo.1418410.

[49] J. Vidali. Using symbolic computation to prove nonexistence of distance-regular graphs. *Electron. J. Combin.*, 25(4)#P4.21, 2018. http://www.combinatorics.org/ojs/index.php/eljc/article/view/v25i4p21

[50] C. Weng. Classical distance-regular graphs of negative type. *J. Combin. Theory Ser. B*, 76(1):93–116, 1999. doi:10.1006/jctb.1998.1892.

[51] H. A. Wilbrink and A. E. Brouwer. A $(57, 14, 1)$ strongly regular graph does not exist. *Nederl. Akad. Wetensch. Indag. Math.*, 45(1):117–121, 1983.