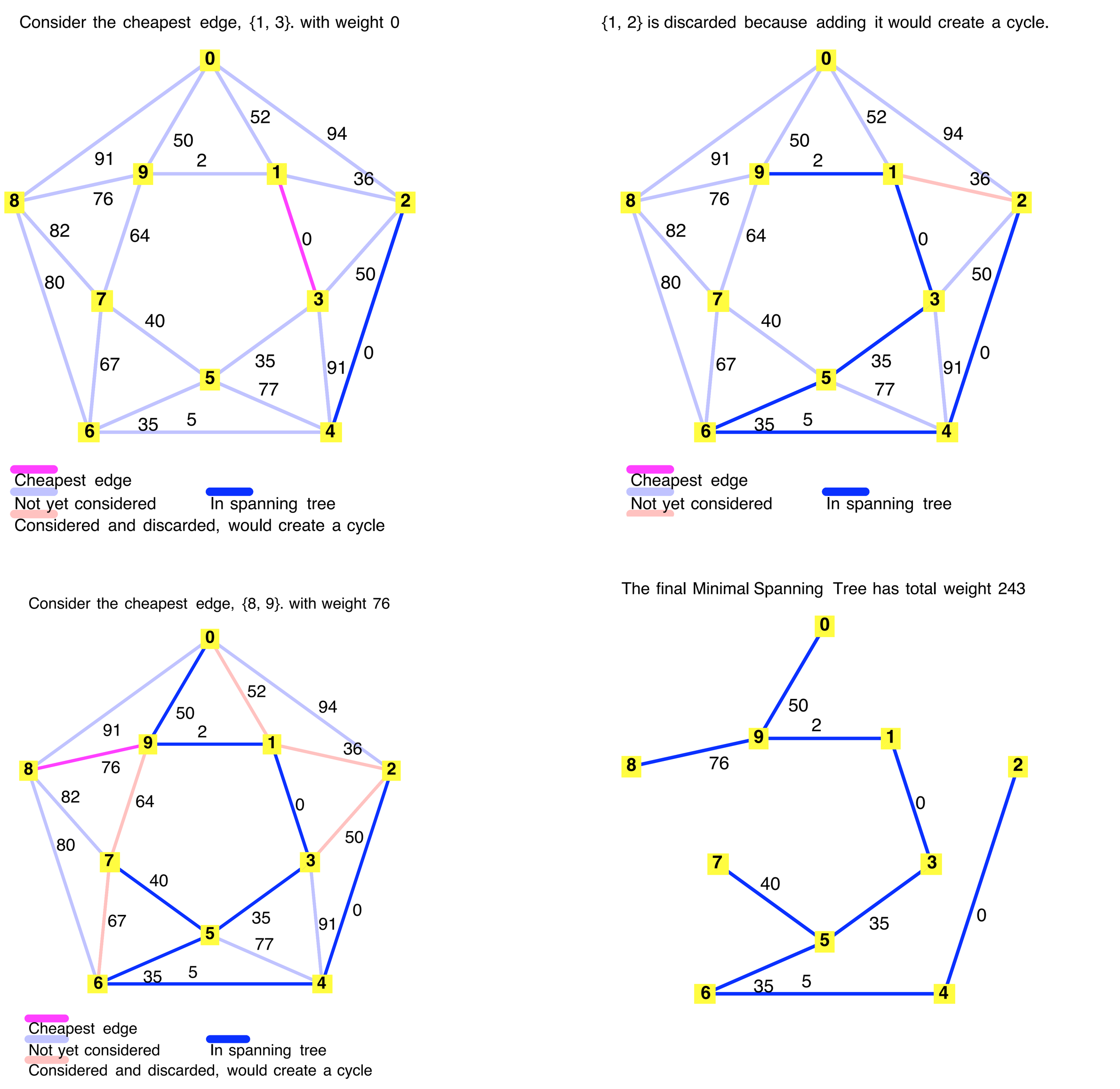
New Features of the GraphTheory Package

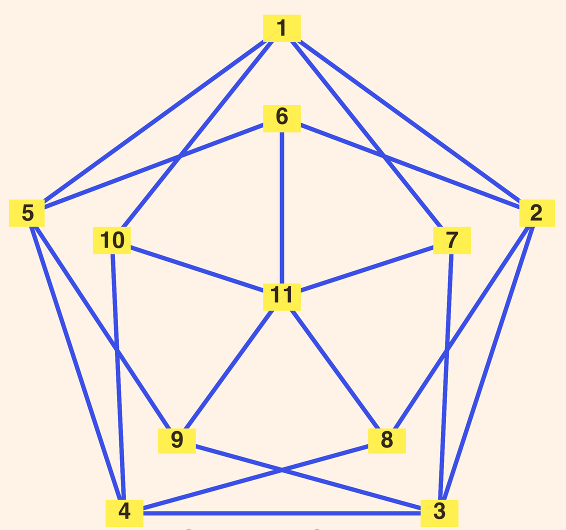
Contributors: Al Erickson, Mohammad Ghebleh, Simon Lo, Michael Monagan

Animations for Prim's and Kruskal's Algorithms

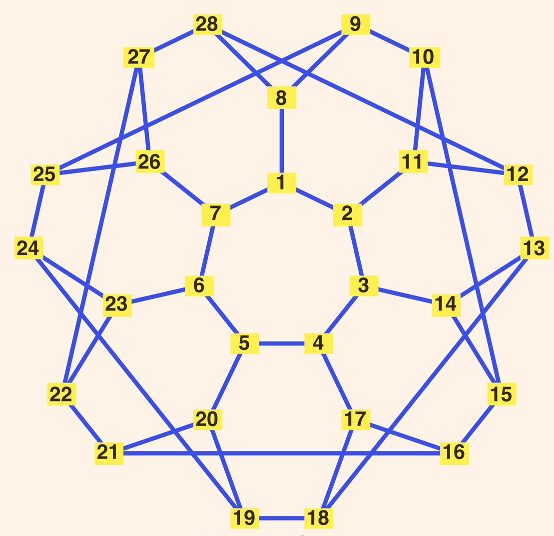
Below we show some key frames from the commands:
> G:=AntiPrismGraph(5,2):
> G:=AssignEdgeWeights(G,0..99):
> AnimateMinimalSpanningTree(G);

The code, which consists of a few `HighlightEdges` commands added to `KruskalsAlgorithm` and `PrimsAlgorithm`, generates a pleasing and instructive animation of these two fundamental greedy algorithms.

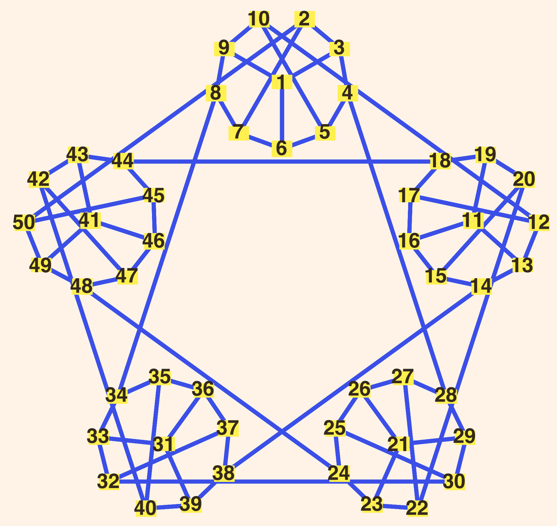




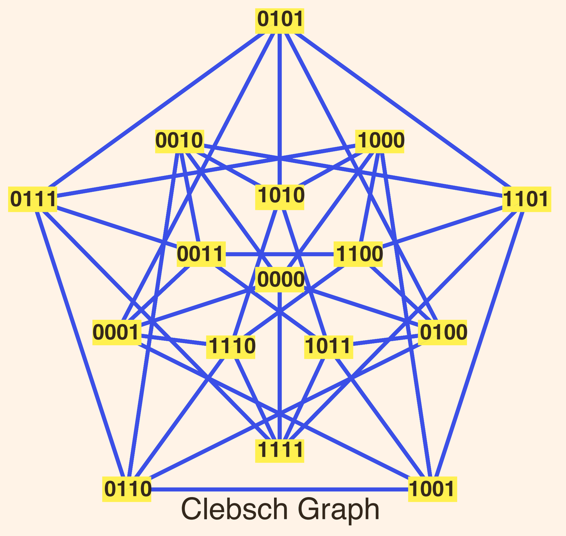
Grotzsch Graph



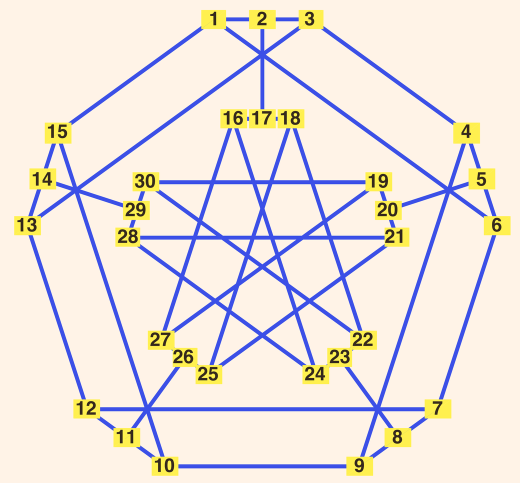
Flower Snark



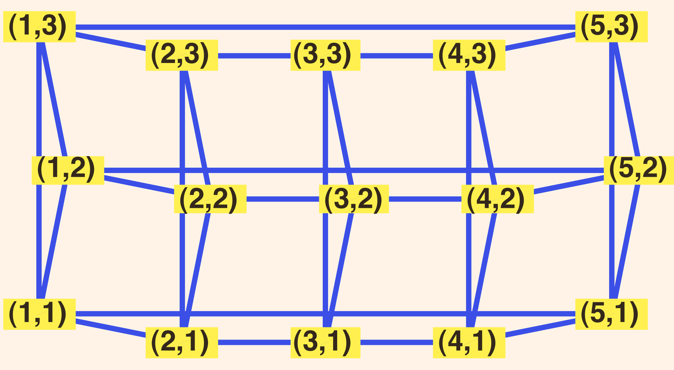
Szekeres Snark



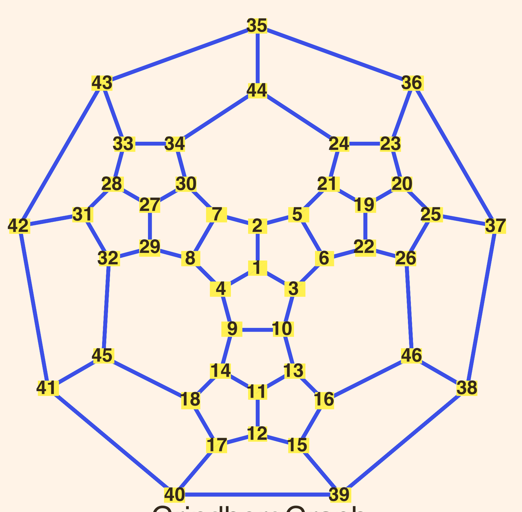
Clebsch Graph



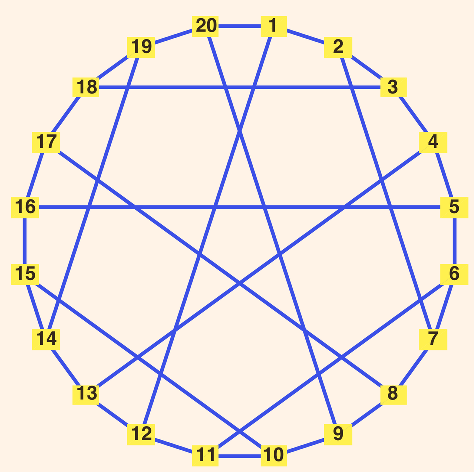
Double Star Snark



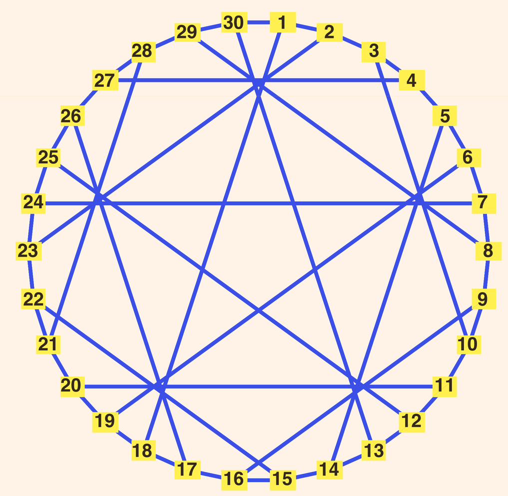
Torus Grid Graph



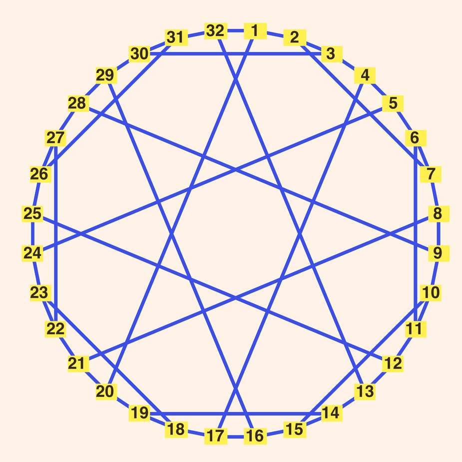
GrindbergGraph



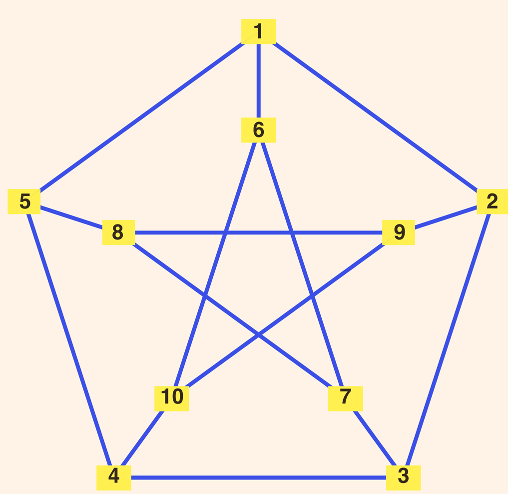
Desargues Graph



Levi Graph



Dyck Graph



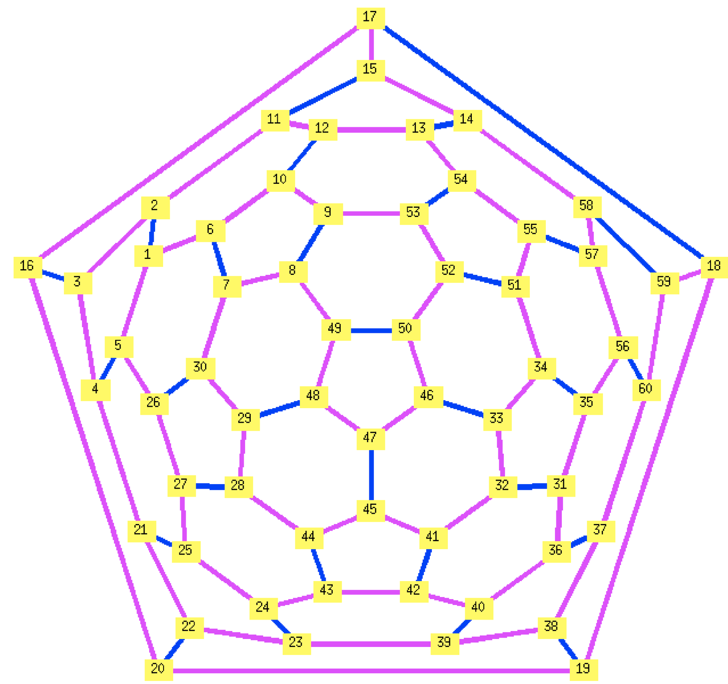
Petersen Graph

Special Graphs with Special Properties

Some of the larger graphs with properties that are difficult to detect as well as convenient drawings have them hardcoded into the graph.
Here we use the `SoccerBallGraph` as an example.

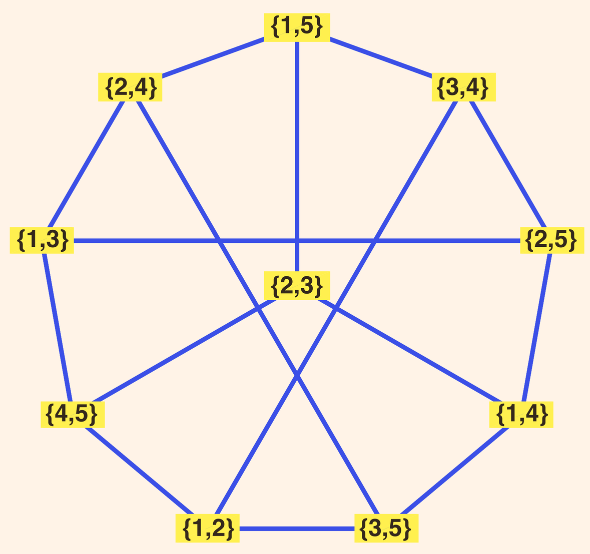
A Hamilton Cycle

Finding a hamilton cycle is NP-hard. `IsHamiltonian` is a simple back tracking algorithm. It comes to a vertex along an edge and then looks for an unvisited neighbour. If it cannot find one, it backtracks. The SoccerBallGraph has 60 vertices and is 3 regular, this means that `IsHamiltonian` is O(2^60) which is not doable unless there are lots of cycles and we get lucky.
The way we found the hamilton cycle shown in the figure is as follows. We generated a 3 edge colouring of the graph at random, then tested to see if deleting edges of one colour left one cycle. After a few tries we got lucky and found one.

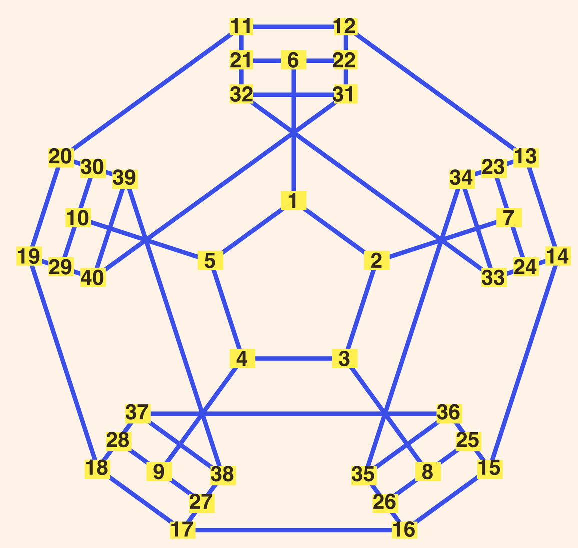


A Planar Drawing

Having found a Hamilton cycle, the need to display it properly motivated us to find a planar drawing. To avoid some extra labour we used the existing particle-spring drawing model to generate the vertex positions. By deleting the edges around one of the pentagonal faces of the graph (the outer face in the figure) the vertices of that face repelled each other toward the outside of the drawing. Projecting the result into the plane gives a planar drawing. This is now the default drawing for the `SoccerBallGraph`.



Kneser Graph



Goldberg Snark

Graph Isomorphism

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two undirected, unweighted graphs, where V_i is the set of vertices and E_i is the set of edges. The problem of Graph Isomorphism is to determine if there exists a bijection $\phi : V_1 \rightarrow V_2$ such that for every $u, v \in V_1$, $\{u, v\} \in E_1$ iff $\{\phi(u), \phi(v)\} \in E_2$. If G_1, G_2 are isomorphic, then we should output the mapping ϕ.

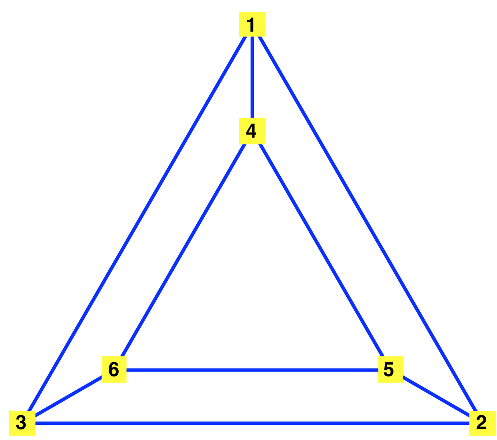
The Graph Isomorphism problem is believed to be neither NP-complete nor in P. There is no known polynomial time algorithm that solves the general problem, therefore, we use certain heuristics such as degree sequences, neighbourhood information, and distances between vertices to prune the search.

All Pairs Distance Matrix

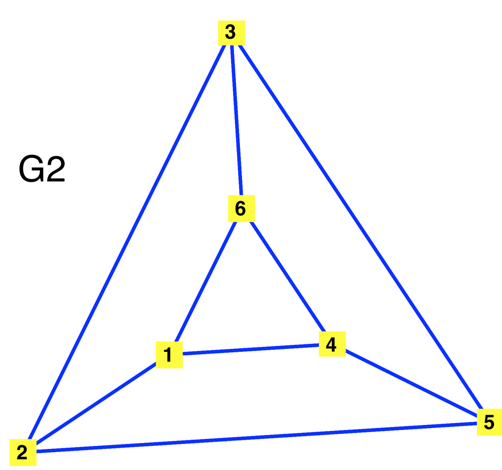
Obviously, if the two graphs have different numbers of vertices or edges, or different degree sequences, they can't be isomorphic. Next, we compute the All Pair Distance (APD) matrix, whose entry m_{ij} is the distance between vertices i, j. **Observation:** if we sort the integers in each row of the APD matrix, then the rows corresponding to matching vertices must be the same. To test if two rows have the same entries (distances) efficiently, we compute a hash value for them. In the backtracking algorithm we match vertices with the same hash value, using neighbour degrees and distances to prune the search. If we can't match any vertices, then the graphs must be non-isomorphic.

Shown is a prism graph (G1) with 6 vertices, and G2, a random permutation of the vertices of G1, and their adjacency matrices and their APD matrices.

We have implemented the computation of the APD matrix in C. It costs O(n^3). We compute the characteristic polynomial $c(\lambda)$ modulo a prime p at two random points α and β in $\mathbb{Z}_p$ which also is O(n^3). As an example, if G is the prism graph on 200 vertices (it has 300 edges) we can find an random isomorphism in just over 1 second. Of this, about 20% is spent computing $c(\alpha)$ and $c(\beta)$ and 5% is computing the APD matrix and over 80% is backtracking.



G1



G2

$$\begin{bmatrix} 0&1&1&1&0&0\\ 1&0&1&0&1&0\\ 1&1&0&0&0&1\\ 1&0&0&0&1&1\\ 0&1&0&1&0&1\\ 0&0&1&1&1&0 \end{bmatrix}, \begin{bmatrix} 0&1&0&1&0&1\\ 1&0&1&0&1&0\\ 0&1&0&0&1&1\\ 1&0&0&0&1&1\\ 0&1&1&1&0&0\\ 1&0&1&1&0&0 \end{bmatrix}$$

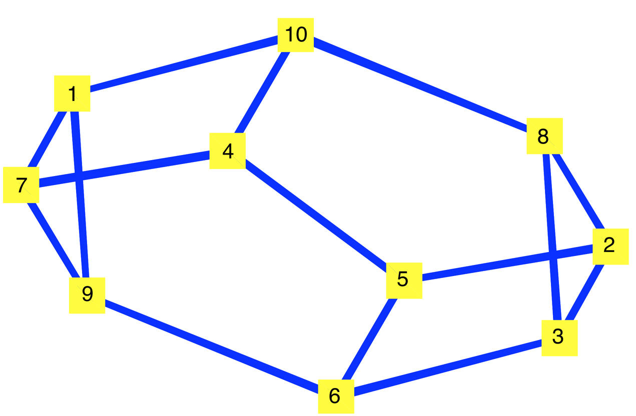
$$\begin{bmatrix} 0&1&1&1&2&2\\ 1&0&1&2&1&2\\ 1&1&0&2&2&1\\ 1&2&2&0&1&1\\ 2&1&2&1&0&1\\ 2&2&1&1&1&0 \end{bmatrix}, \begin{bmatrix} 0&1&2&1&2&1\\ 1&0&1&2&1&2\\ 2&1&0&2&1&1\\ 1&2&2&0&1&1\\ 2&1&1&1&0&2\\ 1&2&1&1&2&0 \end{bmatrix}$$

(top)
Adjacency Matrices for G1, G2, respectively
(bottom)
All Pairs Distance Matrices for G1, G2, respectively

RandomRegularGraph

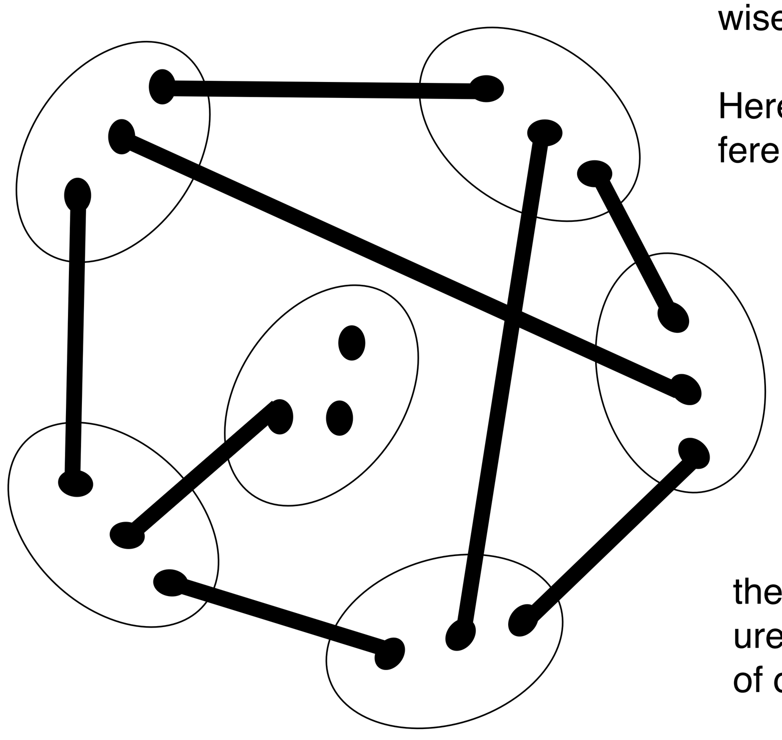
The RandomGraphs package contains routines for generating random graphs, trees, digraphs, networks, tournaments and regular graphs. For regular graphs on n vertices with each vertex of degree d, we have implemented Steger and Wormald's [1] algorithm. Quoted from their paper, the algorithm is as follows:

1. Start with nd points {1,2,...,nd} (nd even) in n groups. Set U={1,2,...,nd}. (U denotes the set of unpaired points.)
2. Repeat the following until no suitable pair can be found: Choose two random points i and j in U, and if they are suitable, pair i with j and delete i and j from U.
3. Create a graph G with edge from vertex r to vertex s if and only if there is a pair containing points in the r'th and s'th groups. If G is d-regular, output it, otherwise return to Step 1.



A 3-regular graph on 10 vertices

Here, suitable means the points "lie in different groups and no currently existing pair contains points in the same two groups".
The running time for the algorithm is O(n d^2 + d^4) which makes it efficient for 3 and 4 regular graphs. The authors prove randomness for small d. We get randomness for large d by taking the complement.
The algorithm can get stuck inserting the last few edges as illustrated in the figure. This is the reason for the d^2 instead of d in the running time.



6 groups and 3 points. The algorithm is 'stuck'.

References:
1. Steger, A. and Wormald, N. C.
Generating random regular graphs quickly.
Combin. Probab. Comput. 8 (1999), no. 4, 377--396.