Gelly:Graph Generators

完全图
环图
空图
网格图
超立方体图
路径图
RMat图
单边图
星图

Gelly 提供了一组可扩展的图生成器。每个生成器都是：

- 并行的, 用于创建大型数据集。
- 自由扩展的, 用于生成并行度无关的同样的图。
- 简洁的，使用了尽可能少的操作。

图生成器使用Builder模式进行配置，可以通过调用`setParallelism(parallelism)`设置并行度。减少 并行度可以降低内存和网络缓冲区的使用。

特定的图配置必须首先被调用，该配置对所有的图生成器都是通用的，最后才会调用`generate()`。 接下来的例子使用两个维度配置了网格图，配置了并行度并生成了图。

```
ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();

boolean wrapEndpoints = false;

int parallelism = 4;

Graph<LongValue,NullValue,NullValue> graph = new GridGraph(env)
    .addDimension(2, wrapEndpoints)
    .addDimension(4, wrapEndpoints)
    .setParallelism(parallelism)
    .generate();
```

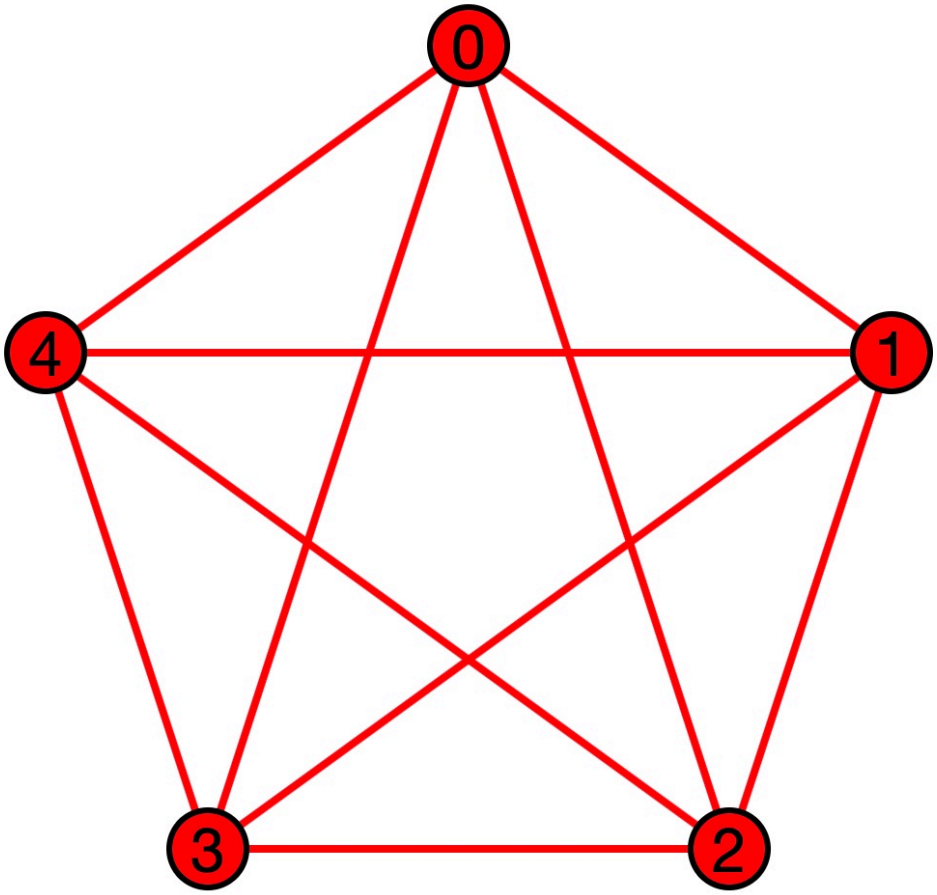
完全图

连接所有不同顶点对的无向图。

```
ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();
```

```
long vertexCount = 5;

Graph<LongValue,NullValue,NullValue> graph = new CompleteGraph(env, vertexCount)
    .generate();
```



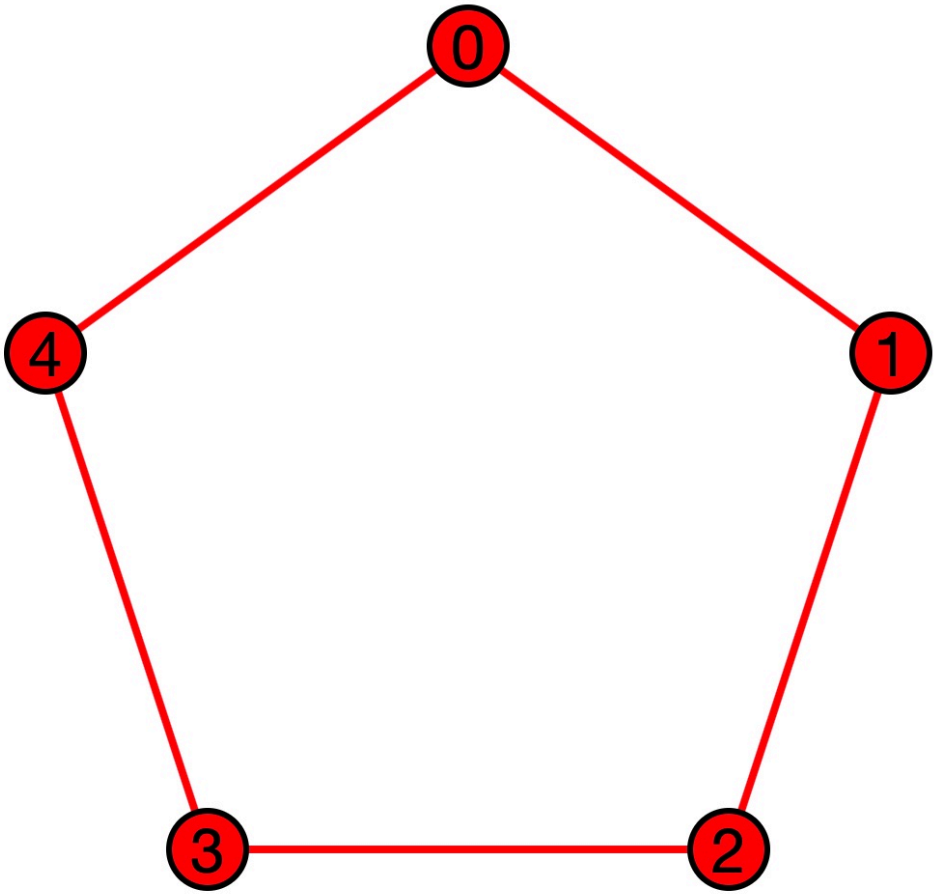
环图

所有的边形成一个环的无向图。

```
ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();

long vertexCount = 5;

Graph<LongValue,NullValue,NullValue> graph = new CycleGraph(env, vertexCount)
    .generate();
```



空图

不存在边的图。

```
ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();

long vertexCount = 5;

Graph<LongValue,NullValue,NullValue> graph = new EmptyGraph(env, vertexCount)
    .generate();
```

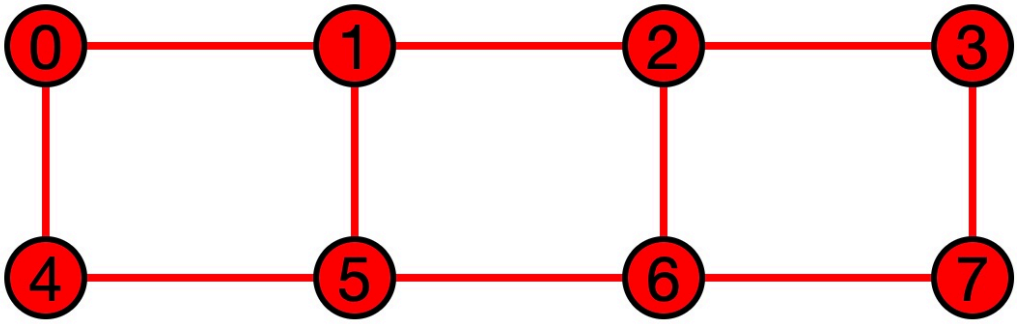
网格图

一种点在一到多个维度正常平铺的无向图。每个维度都是独立配置的。当维度大小多于3时，每个维度的端点 可以通过设置`wrapEndpoints`连接起来，那么下边例子的`addDimension(4, true)`将会连接`0`和`3` 以及`4`和`7`。

```
ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();

boolean wrapEndpoints = false;
```

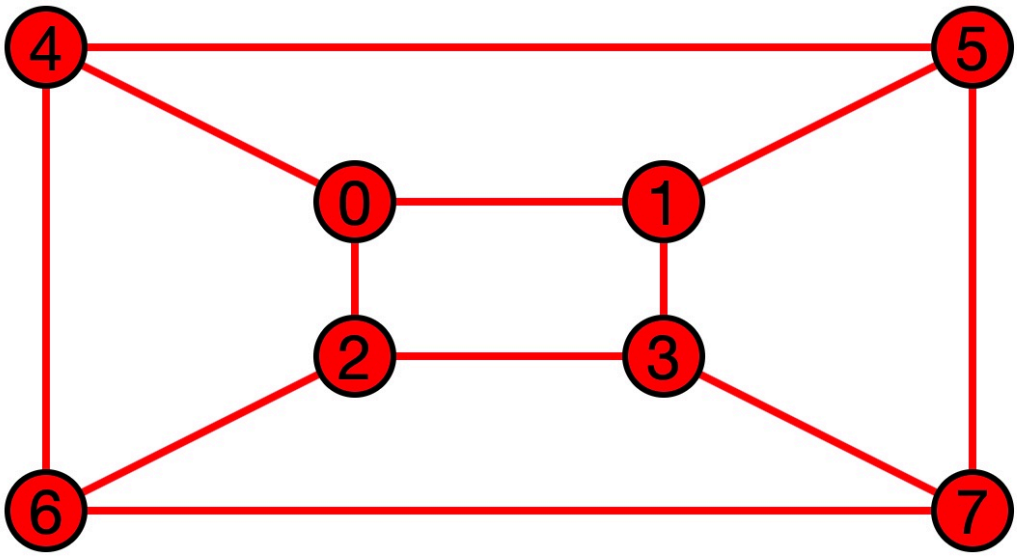
```
4
5 Graph<LongValue,NullValue,NullValue> graph = new GridGraph(env)
6     .addDimension(2, wrapEndpoints)
7     .addDimension(4, wrapEndpoints)
8     .generate();
```



超立方体图

所有的边形成N维超立方体的无向图。超立方体内的每个顶点和同维度的其他顶点连接。

```
1 ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();
2
3 long dimensions = 3;
4
5 Graph<LongValue,NullValue,NullValue> graph = new HypercubeGraph(env, dimensions)
6     .generate();
```



路径图

所有的边形成了一条路径的无向图。

```
1 ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();
2
```

```
long vertexCount = 5

Graph<LongValue,NullValue,NullValue> graph = new PathGraph(env, vertexCount)
    .generate();
```

RMat图

使用Recursive Matrix (R-Mat)模型 生成的有向或者无向幂图。

RMat是一个使用实现RandomGenerableFactory接口的随机源配置的随机生成器，JDKRandomGeneratorFactory 和MersenneTwisterFactory实现了该接口。它产生了一个用于生成边的随机种子的随机初始序列。

```
ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();

RandomGenerableFactory<JDKRandomGenerator> rnd = new JDKRandomGeneratorFactory();

int vertexCount = 1 << scale;
int edgeCount = edgeFactor * vertexCount;

Graph<LongValue,NullValue,NullValue> graph = new RMatGraph<>(env, rnd, vertexCount,
edgeCount)
    .generate();
```

The default RMat contants can be overridden as shown in the following example. The contants define the interdependence of bits from each generated edge's source and target labels. The RMat noise can be enabled and progressively perturbs the contants while generating each edge.

The RMat generator can be configured to produce a simple graph by removing self-loops and duplicate edges. Symmetrization is performed either by a "clip-and-flip" throwing away the half matrix above the diagonal or a full "flip" preserving and mirroring all edges.

```
ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();

RandomGenerableFactory<JDKRandomGenerator> rnd = new JDKRandomGeneratorFactory();

int vertexCount = 1 << scale;
int edgeCount = edgeFactor * vertexCount;

boolean clipAndFlip = false;

Graph<LongValue,NullValue,NullValue> graph = new RMatGraph<>(env, rnd, vertexCount,
edgeCount)
    .setConstants(0.57f, 0.19f, 0.19f)
    .setNoise(true, 0.10f)
    .generate();
```

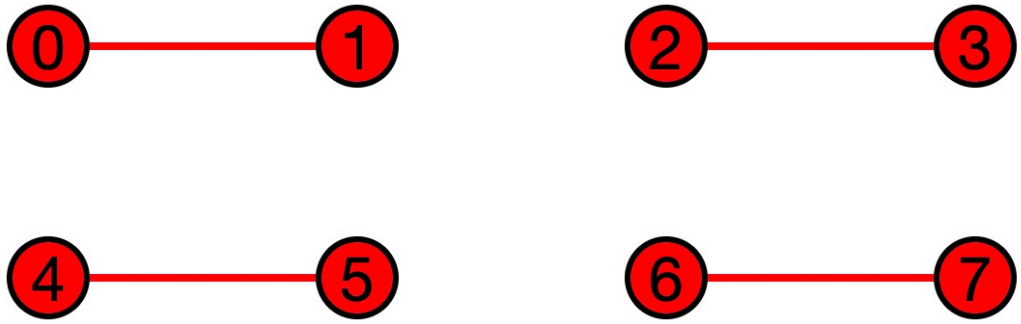
单边图

包含独立的双路径的无向图。

```
ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();

long vertexPairCount = 4

// note: configured with the number of vertex pairs
Graph<LongValue,NullValue,NullValue> graph = new SingletonEdgeGraph(env, vertexPairCount)
    .generate();
```



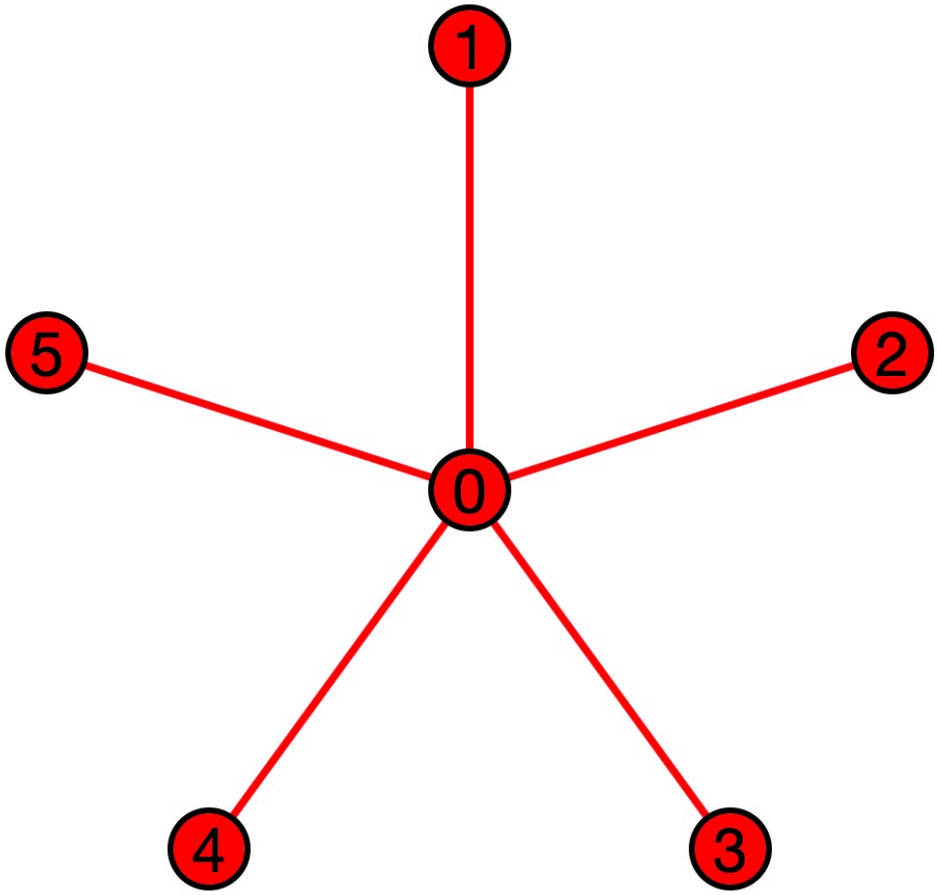
星图

包含一个连接到所有其他叶子顶点的中心顶点的无向图。

```
ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();

long vertexCount = 6;

Graph<LongValue,NullValue,NullValue> graph = new StarGraph(env, vertexCount)
    .generate();
```


1
5
2
0
4
3