Gelly:Library Methods

Gelly 拥有一组至今仍在不断增长的图算法来简易分析大规模的图。

社区探测 (Community Detection)
标签传播 (Label Propagation)
连接组件 (Connected Components)
GSA 连接组件 (GSA Connected Components)
单源最短路径 (Single Source Shortest Paths)
GSA 单源最短路径 (GSA Single Source Shortest Paths)
三角枚举器 (Triangle Enumerator)
总结算法 (Summarization)
集聚 (Clustering)
平均集聚系数 (Average Clustering Coefficient)
整体集聚系数 (Global Clustering Coefficient)
局部集聚系数 (Local Clustering Coefficient)
三元统计 (Triadic Census)
三角罗列 (Triangle Listing)
链接分析 (Link Analysis)
基于超链接的主题检索 (Hyperlink-Induced Topic Search)
佩奇排名 (PageRank)
指标 (Metric)
顶点指标 (Vertex Metrics)
边指标 (Edge Metrics)
相似度 (Similarity)
AA指数 (Adamic-Adar)
杰卡德指数 (Jaccard Index)

Gelly 库的方法能够通简单地过对输入图调用 `run()` 方法来使用：

Java Scala

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

Graph<Long, Long, NullValue> graph = ...

// 用 30 次迭代运行 Label Propagation 来探测输入图的社区 (communities)
```

```
DataSet<Vertex<Long, Long>> verticesWithCommunity = graph.run(new
LabelPropagation<Long>(30));

// 打印结果
verticesWithCommunity.print();
```

社区探测 (Community Detection)

概览

在图论中，社区 (communities) 指的是一组对内紧密连接，但对外与其它组连接稀疏的节点。 该库方法是一个社区探测算法的实现，该算法的具体描述请参阅 Towards real-time community detection in large networks 这篇论文。

细节

该算法通过使用 scatter-gather iterations 来实现。 最开始，每个顶点被分配一个 `Tuple2` 它包含了其初始值和一个分数， 该分数等于 1.0. 在每一次迭代中，所有顶点将自身的标签和分数发送给它们的邻居。当接受到来自邻居的信息时，顶点选择分数最高的标签并用边值，一个用户指定的跳衰减 (hop attenuation) 参数 `delta`，和超步 (superstep) 数对标签重新算分。 该算法在顶点不再更新它们的值或到达最大迭代次数时收敛。

用法

该算法接收一个任何顶点类型的 `Graph` ， `Long` 类型顶点值和 `Double` 类型的边值作为输入，返回一个和输入同类型的 `Graph`， 其中顶点值与社区标签 (community labels) 对应，也就是说如果两个顶点有相同的顶点值，则这两个顶点属于同一个社区。 构造函数接收两个参数：

- `maxIterations`: 要运行的最大迭代数.
- `delta`: 跳衰减参数，默认值为 0.5.

标签传播 (Label Propagation)

概览

这是一个有名的标签传播算法的实现，该算法在这篇论文中有所描述。该算法对一个图通过迭代地在邻居间传播标签来发现该图中的社区。与社区探测不同，该算法的实现并不使用与顶点相关的分数。

细节

该算法通过使用 scatter-gather iterations 来实现。 标签需要是 `Comparable` 类型，并且通过使用输入 `Graph` 的顶点值来初始化。 该算法通过传播标签来迭代地精细化社区。在每次迭代中，顶点使用在其邻居的标签中频数最高的标签。为了避免有两个或多个标签频数相同的情况，该算法会选取比较大的标签。该算法在没有顶点改变它们的值或者到达最大迭代次数时收敛。需要注意的是不同的初始化可能会导致不同的结果。

用法

该算法接收一个 `Comparable` 顶点类型的 `Graph`，一个 `Comparable` 顶点值类型和一个任意边值类型。 它返回一个顶点组成的 `DataSet`，其中顶点值与该算法收敛后该点所属的社区对应。 构造器接受一个参数：

- `maxIterations`: 要运行的最大迭代数.

连接组件 (Connected Components)

概览

这是一个弱连接组件 (Weakly Connected Components) 算法的实现。当该算法收敛时，如果两个点间是相连的，不管是从哪个点连向哪个点，这两点都属于同一个组件。

细节

该算法通过使用 scatter-gather iterations 来实现。 该实现使用一个可比较的顶点值作为初始的组件ID (component identifier)。定点在每次迭代中广播它们当前的值。当顶点从邻居点接收到组件ID时，如果顶点的值低于它当前的组件ID，该顶点会采用新的组件ID。该算法在顶点不再更新它们的组件ID值或到达最大迭代次数时收敛。

用法

该算法的结果是一个顶点组成的 `DataSet`，其中顶点值与分配给该顶点的组件相对应。 构造器接收一个参数：

- `maxIterations`: 要运行的最大迭代数.

GSA 连接组件 (GSA Connected Components)

概览

这是一个弱连接组件 (Weakly Connected Components) 算法的实现。当该算法收敛时，如果两个点间是相连的，不管是从哪个点连向哪个点，这两点都属于同一个组件。

细节

该算法通过使用 gather-sum-apply iterations 来实现。 该实现使用一个可比较的顶点值作为初始的组件ID (component identifier)。在收集阶段 (gather phase)，每一个顶点收集它们的邻接点的顶点值。在求总阶段 (sum phase) 选择这些值中的最小值。在应用阶段 (apply phase)，如果最小值小于当前值，该算法把最小值设为新的顶点值。该算法在顶点不再更新它们的组件ID值或到达最大迭代次数时收敛。

用法

该算法的结果是一个顶点组成的 `DataSet`，其中顶点值与分配给该顶点的组件相对应。 构造器接收一个参数：

- `maxIterations`: 要运行的最大迭代数.

单源最短路径 (Single Source Shortest Paths)

概览

该算法是一个单源最短路径算法的实现，该实现是为了计算加权图。给定一个源点，该算法计算图中从该源点到所

有其它点的最短路径。

细节

该算法通过使用 scatter-gather iterations 来实现。 在每次迭代中，一个顶点发送一个消息到该顶点的邻居，该消息包含该顶点当前距离和连接该点与邻居的边的权重。一旦接收到这个消息，顶点计算最小距离，如果发现更短的路径，该算法会更新改顶点的值。如果一个顶点在一个超步 (superstep) 间没有改变自身的值，则不会在下一个超步给邻居生成消息。计算在超过指定的最大超步数或没有值可以更新的时候终止。

用法

该接收一个任何顶点类型的 `Graph` 和 `Double` 类型的边值。顶点值可以是任何类型，且不会被该算法使用。顶点类型必须实现 `equals()`。 输出是一个顶点组成的 `DataSet`，其中顶点值对应从给定源点到该点的最小距离。 构造器接收两个参数：

- `srcVertexId` 源点的点ID.
- `maxIterations`: 要运行的最大迭代数.

GSA 单源最短路径 (GSA Single Source Shortest Paths)

该算法通过使用 gather-sum-apply iterations 来实现。

参阅单源最短路径库方法获取实现细节和使用信息。

三角枚举器 (Triangle Enumerator)

概览

这个库方法枚举出现在输入图中的唯一三角 (unique triangles)。一个三角由连接相互三个点的三条边组成。 该实现忽略了边的方向。

细节

这个基本的三角枚举算法对所有共享一个共有顶点的边进行分组，并构建被两个边相连接的三点组合 (triad)。 然后过滤所有不存在关闭三角的第三条边的三点组合。 对于一组共享一个共有顶点的 *n* 边，构建的三点组合的数量是 *((n*(n-1))/2)*。 因此，该算法的一个优化是用较小的出边自由度 (output degree) 对顶点上的边进行分组来减小三角组合的数量。 该实现通过计算边顶点 (edge vertices) 的出边自由度和用较小的自由度来对点上的边进行分组来继承基本的算法。

用法

该算法接收一个有向图作为输入，并输入一个 `Tuple3` 组成的 `DataSet`。顶点ID类型必须是 `Comparable`。 每一个 `Tuple3` 对应一个三角，其中的字段包含了组成三角形的顶点的ID。

总结算法 (Summarization)

概览

总结算法通过基于点和边的值对点和边分组来计算一个浓缩版的输入图。通过这种做法，该算法能帮助洞察获悉图中的模式和分布。 该算法的一个用途是社区的可视化，其中社区所在图极大且该图需要基于存于顶点的社区ID来总结。

细节

在结果图中，每个顶点标识了一组分享同个值的顶点。连接顶点的边表示所有拥有相同边值的边，这些边从相同的顶点群连接顶点。输出图中的两个顶点之间的边表示所有输入图中两个不同顶点组内的顶点之间的拥有相同值的边。

这个算法拥 Flink 数据算子实现。首先，通过顶点的值将顶点分组，并从每一组中选出一个代表点。对于任意边，源点和目标点ID用对应的代表点替换，并拥源点，目标点和边值进行分组。输出顶点和边从他们对应的组中创建。

用法

该算法接收一个有向，顶点（和边)带属性的图作为输入，并输出一个新的图，其中该新图的每一个顶点表示一组来自输入图的顶点，每一条边表示一组来自输入图的边。不仅如此，输出图的每一个顶点和每一条边储存了共有的组值和代表元素的数量。

集聚 (Clustering)

平均集聚系数 (Average Clustering Coefficient)

概览

平均集聚系数衡量一个图的平均正确度。得分从 0.0 (邻居之间没有边) 到 1.0 (完全图)。

细节

想了解集聚系数的详细解释，参阅局部集聚系数 库方法。平均集聚系数是所有拥有至少两个邻居的顶点上的本地集聚系数得分的平均值。每一个顶点，无论自由度如何，对于该得分有相等的权重。

用法

有向和无向的变体均有提供。该分析接收一个简单图作为输入，并输出一个 `AnalyticResult`，该结果包含了顶点的总数和图的平均集聚系数。图 ID 类型必须是 `Comparable` 和 `Copyable`。

- `setLittleParallelism`: 覆盖处理少量数据的算子的平行度

整体集聚系数 (Global Clustering Coefficient)

概览

整体集聚系数衡量了一个图的正确度。得分从 0.0 (邻居之间没有边) 到 1.0 (完全图)。

细节

想了解集聚系数的详细解释，参阅本地集聚系数 库方法。整体集聚系数是整个图上的相连邻居的比率。拥有较高自由度的顶点对于该得分有比较大的权重，因为邻居对 (neighbor pairs) 数是自由数的二次方。

用法

有向和无向的变体均有提供。该分析接收一个简单图作为输入，并输出一个 `AnalyticResult`，该结果包含了图中三点组 (triplet) 和三角 (triangle) 的总数。该结果类提供了一个方法来计算整体集聚系数得分。图 ID 类型必须是 `Comparable` 和 `Copyable`。

- `setLittleParallelism`: 覆盖处理少量数据的算子的平行度

局部集聚系数 (Local Clustering Coefficient)

概览

局部聚类系数衡量每个顶点的邻居的正确度。得分从 0.0 (邻居之间没有边) 到 1.0 (邻居是一个團)。

细节

一个顶点的邻居之间的边是一个三角。对邻居间的边计数相当于计算包含了顶点的三角形的数量。集聚系数是邻居间的边的数目与邻居间潜在边的数目的商。

想要了解三角枚举的详细解释，参阅三角罗列 库方法。

用法

有向和无向的变体均有提供。该分析接收一个简单图作为输入，并输出一个 `UnaryResult` 组成的 `DataSet`，其中包含了顶点ID 顶点自由度，和包含顶点的三角的数量。该结果类提供一个方法来计算局部集聚系数得分。图 ID 类型必须是 `Comparable` 和 `Copyable`。

- `setIncludeZeroDegreeVertices`: 包含了自由度为 0 的顶点的结果
- `setLittleParallelism`: 覆盖处理少量数据的算子的平行度

三元统计 (Triadic Census)

概览

一个三点组合 (triad) 是由一个图内的三个顶点组成。每个三点组合包含了三队可能相连或不相连的顶点。三元统计计算图中每种类型的三点组合的出现次数。

细节

该分析统计四种无向三点组合类型 (由0, 1, 2， 或 3 相连边组成) 或 16 种有向三点组合类型来获得三点组 (triplet) 和边的数量，该统计是通过对来自三角罗列的三角形计数和运行顶点指标来进行的。从三点组的数目中推断出三角的数目，再把三角数和三点组数从边数中移除。

用法

有向和无向的变体均有提供。该分析接收一个简单图作为输入，并为查询每个三点组合类型的数目输出一个带有 accessor 方法的 `AnalyticResult`。图 ID 类型必须是 `Comparable` 和 `Copyable`。

- `setLittleParallelism`: 覆盖处理少量数据的算子的平行度

三角罗列 (Triangle Listing)

概览

枚举图中所有的三角。一个三角由三条把三个点连接成一个尺寸为3的團 (clique) 组成。

细节

通过对三点组终端点 (endpoint) 合并开三点组 (open triplets) (有一个公共的邻居的两个边) 来列出三角。 该实现使用来自 Schank's algorithm 的优化来提高高自由度顶点的表现。三点组从低自由度的顶点中产生，因为每个三角只有需要被列出来一次。这会显著减少产生的三点组的数量，三点组是顶点自由度的二次方。

用法

有向和无向的变体均有提供。该算法接收一个简单图作为输入，并输出一个 `TertiaryResult` 组成的 `DataSet`，其中包含了三个三角定点。对于有向算法，还包含一个位掩码，该位掩码标记六个潜在的连接三角的点的边。图 ID 类型必须是 `Comparable` 和 `Copyable`。

- `setLittleParallelism`: 覆盖处理少量数据的算子的平行度
- `setSortTriangleVertices`: 归范化三角罗列，这样对于每一个结果 (K0, K1, K2)，顶点 ID 会被排序成 K0 < K1 < K2

链接分析 (Link Analysis)

基于超链接的主题检索 (Hyperlink-Induced Topic Search)

概览

基于超链接的主题检索 (HITS, or "枢纽和权威" ("Hubs and Authorities")) 为一个有向图的每个顶点计算两个互相独立的分数。优质的枢纽是那些指向许多优质的权威的枢纽，而优质的权威是那些被优质的枢纽指向的权威。

细节

每个点被分配相同的初始枢纽得分和权威得分。该算法迭代地更新得分直到终结。在每一次迭代中新的枢纽得分从权威得分计算而得，然后新的权威分数从新的枢纽得分计算得出。这些分数接着为收敛被归一化和测试。 HITS 算法和佩奇排名相似，只是顶点得分全部发送给每一个邻居，而在佩奇排名中定点得分先除以邻居的分数。

用法

该算法接收一个简单的有向图作为输入，输出一个 `UnaryResult` 组成的 `DataSet`，其中包含了顶点的 ID，枢纽得分，和权威得分。可以通过配置迭代的次数和/或在所有顶点上的得分变化的迭代总和上的收敛阈值来决定何时终结算法。

- `setParallelism`: 覆盖算子的平行度

佩奇排名 (PageRank)

概览

佩奇排名是一个最早被使用来对网页搜索引擎进行排名的算法。如今，该算法及其许多变体在各种图应用领域都被使用。排气算法的思想是重要或相关的顶点总是倾向与其它重要的顶点链接。

细节

该算法迭代地进行，其中页面将它们的分数分布给它们的邻居 (它们链接的页面) 并根据它们接收的分数更新其分数。为了考虑一个页面到另一个页面的链接的重要性，分数会除以源页面的外向链接的总数。因此，一个有 10 个链接的页面会分配它的得分的 1/10 到每一个邻居，而一个有 100 个链接的页面会分配它的得分的 1/100 到每一个邻居.

用法

该算法接收一个有向图作为输入，并输出一个 `DataSet`，其中每一个 `Result` 包含顶点 ID 和佩奇排名得分。可以通过配置迭代的次数和/或在迭代之间的每一个顶点的分数变化的总和上的收敛阈值来决定何时终结算法。

- `setParallelism`: 覆盖算子的平行度

指标 (Metric)

顶点指标 (Vertex Metrics)

概览

该图分析对有向图和无向图计算下列统计： - 顶点数量 (number of vertices) - 边数量 (number of edges) - 平均自由度 (average degree) - 三点组数 (number of triplets) - 最大自由度 (maximum degree) - 三点组最大数 (maximum number of triplets)

对有向图，可以额外计算下列统计： - 无向边数量 (number of unidirectional edges) - 双向边数量 (number of bidirectional edges) - 最大出边自由度 (maximum out degree) - 最大入边自由度 (maximum in degree)

细节

该统计被计算在
从 `degree.annotate.directed.VertexDegrees` 或 `degree.annotate.undirected.VertexDegree` 产生的点自由度 (vertex degree) 上。

用法

有向和无向的变体均有提供。该分析接收一个简单图作为输入，并为计算过的统计输出一个带有 accessor 方法的 `AnalyticResult`。图 ID 类型必须是 `Comparable`。

- `setIncludeZeroDegreeVertices`: 包含了自由度为 0 的顶点的结果
- `setParallelism`: 覆盖算子的平行度
- `setReduceOnTargetId` (仅对无向图): 自由度可以从边的源点 ID 或 目标点 ID 计算。默认情况下用源头 ID

计算。如果输入边列表通过目标点 ID 排序，在目标点上的归约可能优化该算法

边指标 (Edge Metrics)

概览

该图分析对有向图和无向图计算下列统计： - 三角三点组数 (number of triangle triplets) - 长方三点组数 (number of rectangle triplets) - 三角三点组的最大数量 (maximum number of triangle triplets) - 长方三点组的最大数量 (maximum number of rectangle triplets)

细节

该统计被计算在
从 `degree.annotate.directed.EdgeDegreesPair` 或 `degree.annotate.undirected.EdgeDegreePair` 产生的边自由度 (vertex degree) 上并通过顶点分组。

用法

有向和无向的变体均有提供。该分析接收一个简单图作为输入，并为计算过的统计输出一个带有 accessor 方法的 `AnalyticResult`。图 ID 类型必须是 `Comparable`。

- `setParallelism`: 覆盖算子的平行度
- `setReduceOnTargetId` (仅对无向图): 自由度可以从边的源点 ID 或 目标点 ID 计算。默认情况下用源头 ID 计算。如果输入边列表通过目标点 ID 排序，在目标点上的归约可能优化该算法

相似度 (Similarity)

AA指数 (Adamic-Adar)

概览

AA指数衡量顶点对之间的相似度，通过共享邻居上的自由度的逆对数的和来计算。得分是非负且无界的。拥有较高自由度的顶点总体上有较大的影响，但每对邻居则没那么有影响。

细节

该算法首先用顶点自由度的逆对数标注每个顶点，然后用源点把该分数合并到边上。在源点上分组，发送每一对邻居与其顶点得分。在顶点组上分组，然后计算AA指数的得分。

查阅杰卡德指数库方法来了解相似的算法。

用法

该算法接收一个简单的无向图作为输入，并输出一个 `BinaryResult` 组成的 `DataSet`，其中包含了两个顶点 ID 和 AA 相似度分数。图 ID 类型必须是 `Copyable`。

- `setLittleParallelism`: 覆盖处理少量数据的算子的平行度
- `setMinimumRatio`: 过滤小于给定指数乘以平均分数的结果的得分
- `setMinimumScore`: 过滤小于给定最小值的得分

杰卡德指数 (Jaccard Index)

概览

杰卡德指数衡量顶点邻居间的相似度，通过共享的邻居数除以各自不同的邻居数来计算。分数范围从 0.0 (没有共享邻居) 到 1.0 (所有邻居均共享)。

细节

计算顶点对的共享邻居相当于计算长度为 2 的相连通路。各自不同的邻居通过存储顶点对的自由度的总和并减去共享邻居的数目来计算，这在自由度的和上是双倍计算。

该算法先用目标顶点的自由度标注每一条边。在源点上分组，发送每一对邻居与其自由度和。在顶点组上分组，然后计算共享邻居。

用法

该算法接收一个简单的无向图作为输入，并输出一个元组 (tuple) 组成的 `DataSet`，其中包含了两个顶点 ID，共享邻居的数量，和不同邻居的数量。该结果类提供一个方法来计算杰卡德指数得分。图 ID 类型必须是 `Copyable`。

- `setLittleParallelism`: 覆盖处理少量数据的算子的平行度
- `setMaximumScore`: 过滤大于等于给定最大值的得分
- `setMinimumScore`: 过滤小于给定最小值的得分