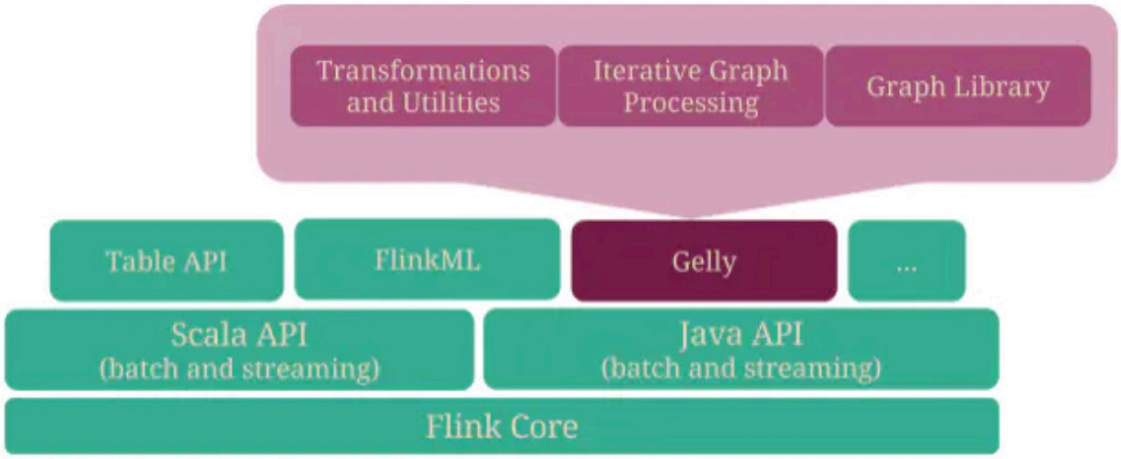
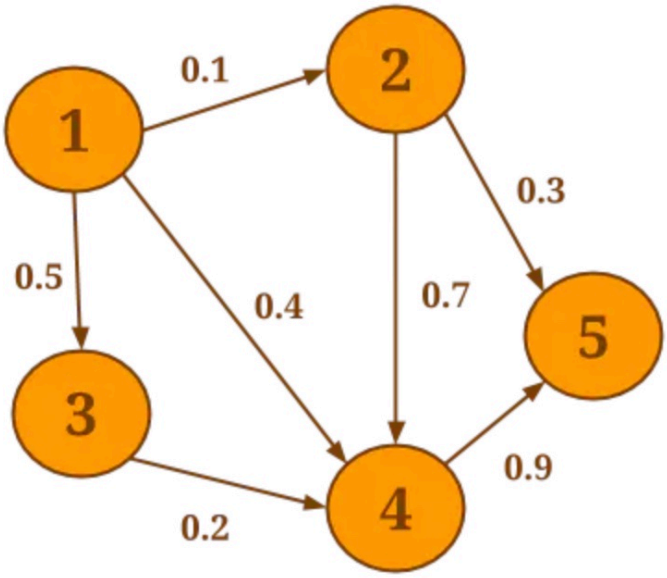
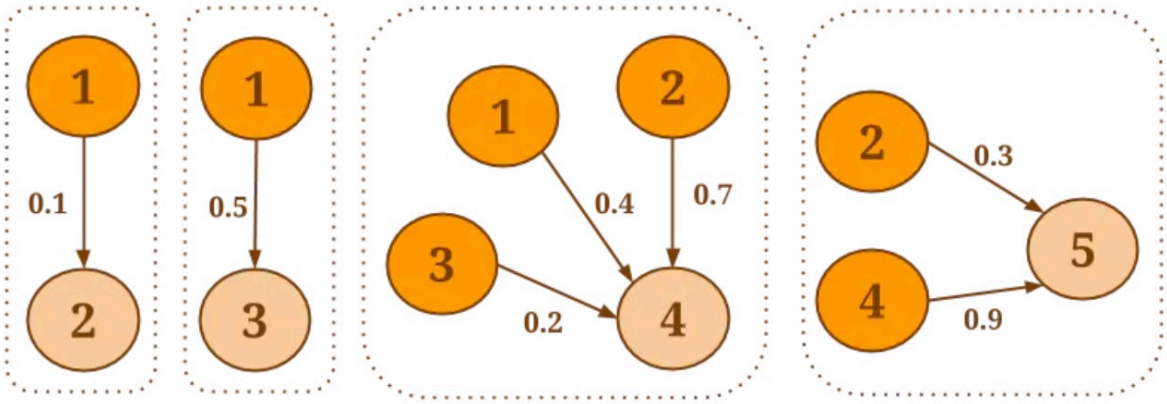
Flink gelly

What is Gelly?

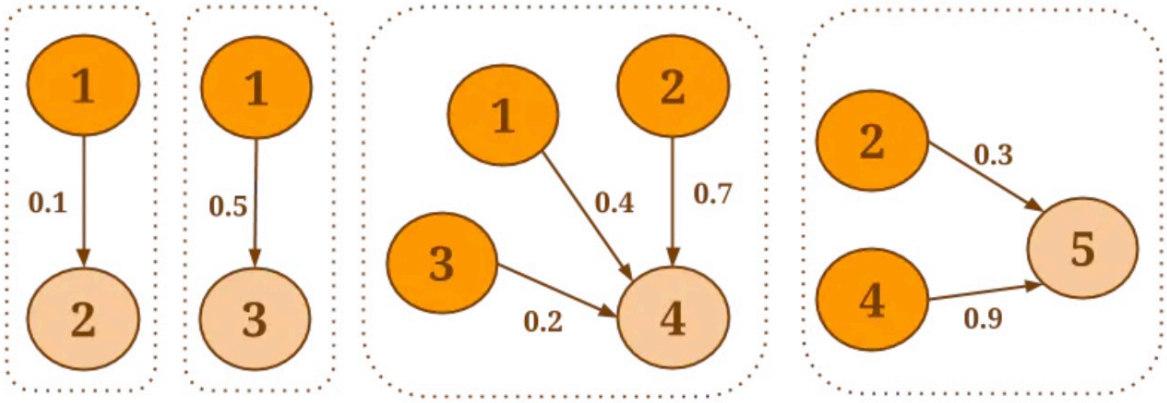


Neighborhood Aggregations(邻居聚合)





result
{ [2, 1], [3, 1], [4, 6], [5, 6] }



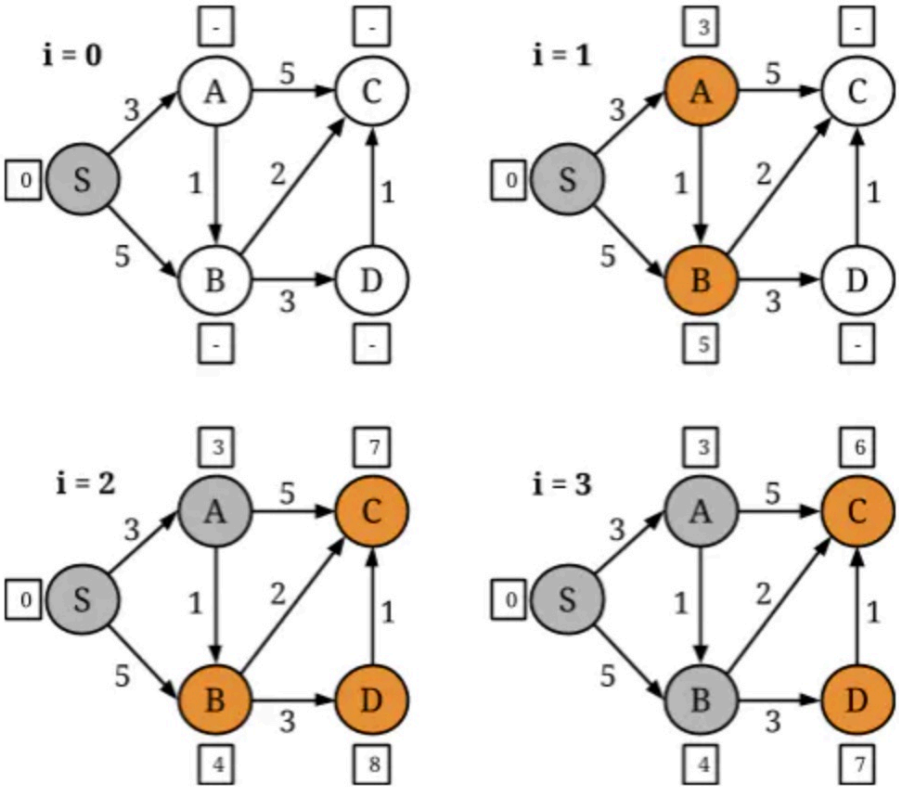
result
{ [2, 1], [3, 1], [4, 6], [5, 6] }

Iterative Graph Processing

Vertex-centric

MessagingFunction: 给下一个超步发送的消息。

VertexUpdateFunction:定义一个节点对收到的消息如何去处理

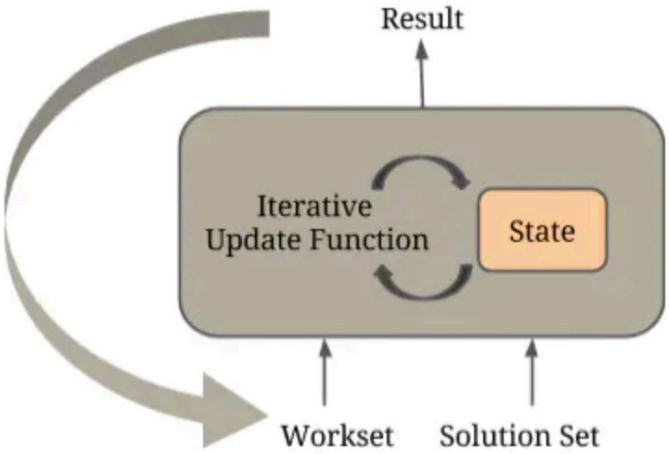


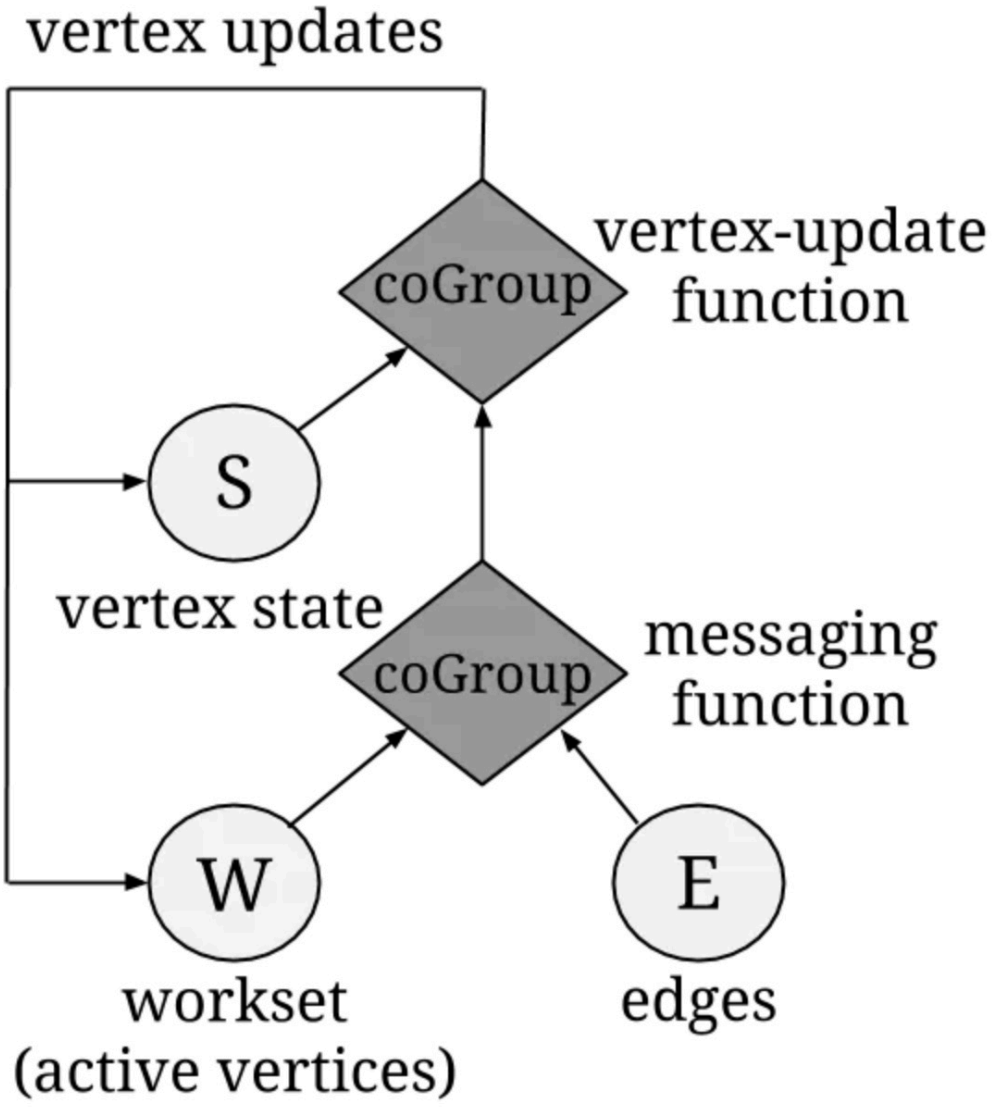
operator收到两个输入：

the Solution Set： 当前输入的状态

the Workset, 图的一部分在下次迭代中会被重新计算

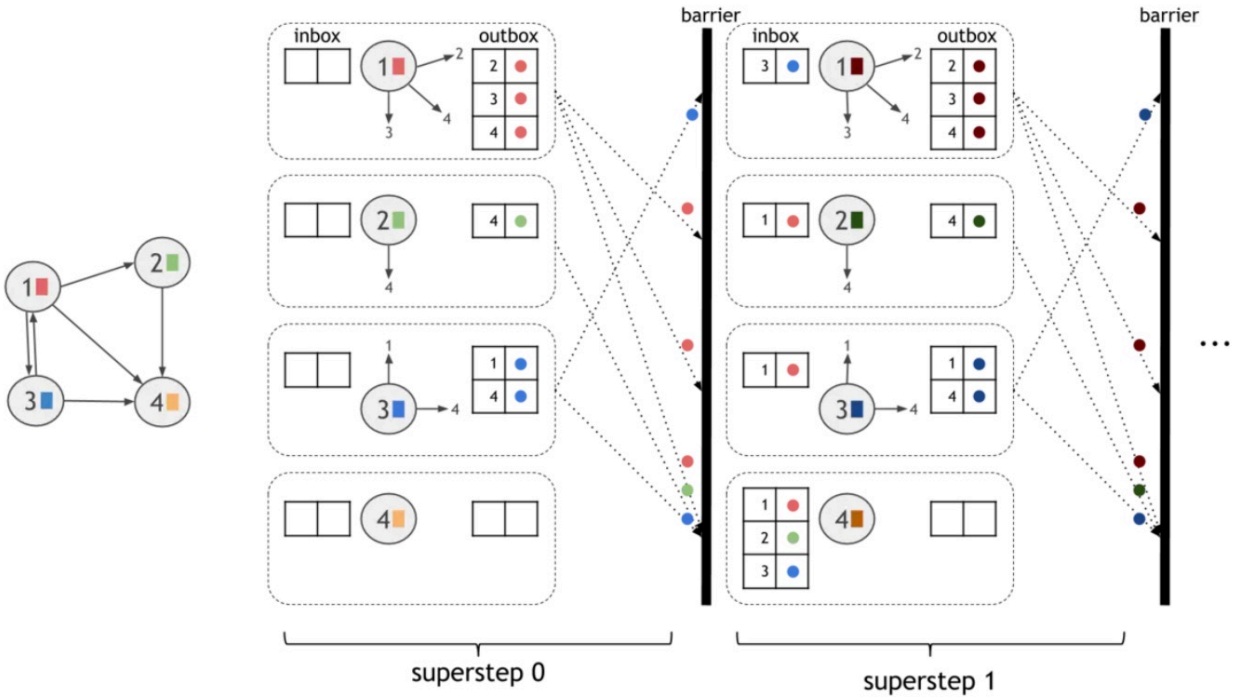
Vertex Centric迭代计算





Vertex Centric模型，也称为"像顶点一样思考"或"Pregel"，通过图顶点的角度表达计算。 该计算在迭代的每一步（称为超步）中同步地处理，在每个超步时，每个顶点执行一个UDF（User Defined Function）。 顶点之间通过消息进行通讯，任何一个顶点可以给图中任何其他的顶点发送消息，只要知道它的ID即可。

下面的图中展示该计算模型，虚线框和并行单元对应。在每个超步中，所有的活跃顶点并行执行相同的用户定义的计算。因为超步时同步执行的，因此每次超步中发送的消息都被保证发送了到下次超步的开始。



在Gelly中使用Vertex Centric迭代，用户只需要定义顶点的计算函数ComputeFunction即可。

该函数和最大迭代次数通过Gelly的runVertexCentricIteration函数参数指定。该方法在输入图上执行Vertex Centric迭代计算，并输出更新后顶点值的新图。可选的MessageCombiner函数可以被用于减少通信消耗。

让我们考虑基于Vertex Centric的单源点最短路径算法（SSSP）。算法开始时，除了源顶点初始值为0，每个顶点初始值为正无穷。第一次超步计算时，源顶点将距离传播给它的邻居。在接下来的超步中，每个顶点检查接收的消息并选择其中最小的距离值。如果该距离值比顶点当前值小，则更新顶点的值，并产生消息发送给其邻居。如果顶点在超步中没有更新它的值，则在下次超步时不会发送任何消息给它的邻居。当没有顶点的值发生更新或者达到了最大的超步迭代次数，算法将会收敛。在这个算法中，Message Combiner可以被用来减少发送给目标顶点的消息个数。

```
// 构建图
Graph<Long, Double, Double> graph = ...

// 最大迭代次数
```

```
int maxIterations = 10;

// 执行vertex-centric iteration
Graph<Long, Double, Double> result = graph.runVertexCentricIteration(
        new SSSPComputeFunction(), new SSSPCombiner(), maxIterations);

// 抽取结果
DataSet<Vertex<Long, Double>> singleSourceShortestPaths = result.getVertices();
//用户定义函数
public static final class SSSPComputeFunction extends ComputeFunction<Long, Double, Double,
Double> {

public void compute(Vertex<Long, Double> vertex, MessageIterator<Double> messages) {

    double minDistance = (vertex.getId().equals(srcId)) ? 0d : Double.POSITIVE_INFINITY;

    for (Double msg : messages) {
        minDistance = Math.min(minDistance, msg);
    }

    if (minDistance < vertex.getValue()) {
        setNewVertexValue(minDistance);
        for (Edge<Long, Double> e: getEdges()) {
            sendMessageTo(e.getTarget(), minDistance + e.getValue());
        }
    }
}

// 消息合并
public static final class SSSPCombiner extends MessageCombiner<Long, Double> {

    public void combineMessages(MessageIterator<Double> messages) {

        double minMessage = Double.POSITIVE_INFINITY;
        for (Double msg: messages) {
           minMessage = Math.min(minMessage, msg);
        }
        sendCombinedMessage(minMessage);
    }
}
```