基于Flink Gelly的反欺诈场景的实现和调优

Technical Report

v0.1

许利杰，段世凯，赵伟

{ xulijie, duanshikai14, zhaowei15}@otcaix.iscas.ac.cn

2017-02

# 基于Flink Gelly的反欺诈场景的实现和调优

## 项目背景

如今，金融市场十分火热，特别是在互联金融发展之后，人们可以更加方便的进行金融交易。然而，金融欺诈也开始变得普遍，金融反欺诈也应运而生。鉴于金融欺诈的复杂性和实时性，传统的反欺诈技术很难令人满意。传统的风控和反欺诈依赖于专家系统的规则匹配，在团伙欺诈、淘宝刷钻和金融风控等利用关系网络的场景中，规则匹配很难发现这些特征。同时，这些欺诈场景一般是在线发生，传统技术通过事后离线计算，补救效果不大。所以，我们需要对设计新的反欺诈技术来满足现代金融市场的需求。鉴于图结构能够很好地表达数据之间的复杂依赖关系和聚集情况，我们可以将金融市场的人和交易抽象为图结构的点和边，将反欺诈抽象为图计算。本项目拟实现了基于Flink Gelly的反欺诈系统，通过设计并实现一套在动态流图上的图计算模型和社区发现的算法，从而提高反欺诈水平。

## 预期成果

1. Flink Gelly内核解析文档
2. 实时流式图计算引擎可行性研究
3. 基于Gelly的反欺诈场景的算法选型和算法实现
4. 在公开数据集和华为公司提供的数据集做验证

## 研究方案

构造针对Flink系统的可靠性基准测试框架：

1. 分析批处理图计算模型（包括BSP、GSA等）
2. 分析Flink Gelly内核实现
3. 分析反欺诈场景和社区发现的主流算法
4. 在批处理模型（基于Gelly）上实现这些算法
5. 研究如何在流式图上应用这些算法
6. 抽象出流式图计算模型

项目地址：https://github.com/HuaweiBigData/Flink-Graph.git

## 研发计划

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **时间段** | **类别** | **研究问题** | **系统实现** | **新发现或结论** |
| 2016.10-2016.11 | Flink框架学习 | 学习Flink的框架及API的使用 | 完成了《Learn Flink》手册，介绍Flink的架构和API使用方法 |  |
| 2016.11-2016.12 | Flink Gelly学习 | 学习Gelly的的架构及图相关的算法 | 完成了《Flink Gelly》手册，介绍了Gelly的框架和部分图算法的实现 | Flink Gelly是基于BSP模型实现的运行在批处理上的图计算框架。 |
| 2016.12-2017.01 | Flink Gelly测试 | 测试Flink Gelly中的TC算法 | 完成了论文的实验部分 | Flink Gelly是建立在批处理上的，图计算的相关性，导致当数据量增大时，节点之间的通信显著增加，吞吐率下降 |
| 2017.01-2017.02 | GraphFlow系统设计和实现 | 设计并实现了基于Flink Streaming的图计算框架 | 完成了原型系统GraphFlow，完成了DD和TC算法。 | 流处理充分考虑计算的局部性，采用分流的方式减少节点之间通信，因此当数据量增大时，相比批处理，吞吐率变化不太明显，更为稳定。 |
| 2017.02 | HzGraphFlow系统的设计与实现 | 设计并实现了基于Hazelcast的图计算框架 | 完成了原型系统HzGraphFlow,实现了DD,TC,SSSP和PR算法。 | 根据图算法的特点，利用分布式数据结构存储图状态信息。 |
|  |  |  |  |  |

[计划待增加]

## 目前进展

### 金融反欺诈场景和社区发现的主流算法

1. 金融反欺诈场景有：

|  |  |
| --- | --- |
| 欺诈类型 | 描述 |
| 保险欺诈  insurance fraud | 保险欺诈是指保险合同的买方或卖方的非法行为。 发行人（卖方）的保险欺诈包括来自不存在的公司的销售政策，未能提交保费和改变政策以创建更多的佣金。 买家欺诈包括夸大的索赔，伪造的病史，后期政策，假死亡或绑架，谋杀等 |
| 信用卡欺诈  credit card fraud | 信用卡欺诈是一种广泛的盗窃和诈骗术语，通过使用或涉及支付卡（如信用卡或借记卡）作为交易中欺诈性资金的来源，目的可能是在没有付款的情况下获得货物，或从帐户获得未经授权的资金。  信用卡是欺诈的一个不错的目标，因为在很短的时间内，很可以赚取多钱而不采取许多风险。一些成功的信用卡欺诈技术如下：   * 复制信用卡并以某种方式获得用户的秘密PIN码（如果需要的话） * 供应商收取比答应客户更多的钱，而客户却不知情 |
| 财务报表欺诈  Financial statement fraud | 财务报表欺诈是故意虚假陈述，错误陈述或省略财务报表数据，目的是误导读者并对组织的财务实力造成虚假印象。 公共和私人企业进行财务报表欺诈以确保投资者利益或获得银行批准融资，作为奖金或增加工资的证明或满足股东的期望。 上层管理层通常是财务报表欺诈的中心，因为财务报表是在管理层创建的。 |
| 电信欺诈telecommunications fraud | 电信欺诈是指使用电信产品或服务，意图非法从电信公司或其客户收取金钱，或者逃避付款。电信欺诈类型包括：   * 电话克隆 - 通过模拟另一个正品手机的识别码访问网络。 这里，使用真实识别码来操作未授权手机，包含在移动电话芯片上的数据从一个移动电话复制并嵌入到另一个移动电话中。 * 订阅欺诈 - 使用虚假身份信息订阅移动电话网络。 这种类型的欺诈往往直到账单未付才会被发现。 该问题中的移动电话也可以被克隆，导致使用相同帐户的电话的多于一个实例。 |
| 证券欺诈  Securities Fraud | 证券欺诈，也称为股票欺诈和投资欺诈，是股票或商品市场中的欺骗性做法，诱使投资者基于虚假信息做出购买或销售决定，经常导致损失，违反证券法。对无法评估风险并且无法承受资本损失的不复杂投资者提供风险投资机会是一个核心问题。 |

1. 相关社区发现算法和图算法：

|  |  |
| --- | --- |
| 算法 | 描述 |
| KCore | KCore是图算法中的一个经典算法，其计算结果是判断节点重要性最常用的参考值之一。  对于一个给定的图G和K值，K-core算法主要分为以下两个步骤：  1. 遍历图G所有节点，删除其中度小于K的，并且删除掉和这些被删除的节点相连的边。  2. 更新相应的节点信息，重复步骤1，直到剩下节点的度均不小于K，即得到该图的K-core子图。 |
| LPA | 标签传播算法基本思想是通过标记节点的标签信息预测未标记节点的标签情况。节点之间的标签传播主要依照标签相似度进行，在传播过程中，未标记的节点根据邻接点的标签情况来迭代更新自身的标签信息，如果其邻接点与其相似度越相近，则表示对其所标注的影响权值就越大，邻接点的标签就更容易进行传播。  LPA主要思想是起初每个节点拥有独立的标签，那么网络中有n不同标签，每次迭代中对于每个节点将其标签更改为其邻接点中出现次数最多的标签，如果这样的标签有多个，则随机选择一个。通过迭代，直到每个节点的标签与其邻接点中出现次数最多的标签相同，则达到稳定状态，算法结束。此时具有相同标签的节点即属于同一个社区。 |
| GN | 边介数GN算法是一种分裂型的社区结构发现算法。该算法根据网络中社区内部高内聚、社区之间低内聚的特点，逐步去除社区之间的边，取得相对内聚的社区结构。算法用边介数的概念来探测边的位置，某边的边介数定义为网络上所有顶点之间的最短路径通过该边的次数。 |
| CPM | 派系过滤CPM方法(clique percolation method)用于发现重叠社区，派系(clique)是任意两点都相连的顶点的集合，即完全子图。  在社区内部节点之间连接密切，边密度高，容易形成派系(clique)。因此，社区内部的边有较大可能形成大的完全子图，而社区之间的边却几乎不可能形成较大的完全子图，从而可以通过找出网络中的派系来发现社区。 |
| BP | 置信度传播BP算法利用结点与结点之间相互传递信息而更新当前整个MRF的标记状态，是基于MRF的一种近似计算。该算法是一种迭代的方法，可以解决概率图模型概率推断问题，而且所有信息的传播可以并行实现。经过多次迭代后，所有结点的信度不再发生变化，就称此时每一个结点的标记即为最优标记，MRF也达到了收敛状态。对于无环环路的MRF，BP算法可以收敛到其最优解。 |

### Flink Gelly的图迭代计算模型分析

Flink Gelly目前提供以下几种图迭代计算模型：vertex-centric（以顶点为中心）、scatter-gather（分散-聚集）、gather-sum-apply 。

1. Vertex-Centric迭代模型

Vertex-Centric 迭代模型又被称为pregel迭代模型，以图中的结点为中心执行计算。该迭代过程分一个个超步superstep，每个superstep之间是同步的。在一个superstep之中，每个结点只需执行一个自定义计算函数，并将计算结果当做消息发送给任意结点。由于superstep之间是同步的，所以每个结点在执行计算时，能收到上一个superstep 中发送给自己的消息。

**Single-Source-Shortest-Paths（SSSP）**：

1. 初始化，每个顶点有一个初始距离（源顶点的初始值为0）。
2. 第一个superstep中，每个顶点检查他所接收到的消息，并选择他们中间最小的距离。如果选择的这个距离小于顶点当前值，它会更新它的状态并向邻居节点传送消息。如果不改变自身状态，则不会向下一个superstep的顶点传递消息。

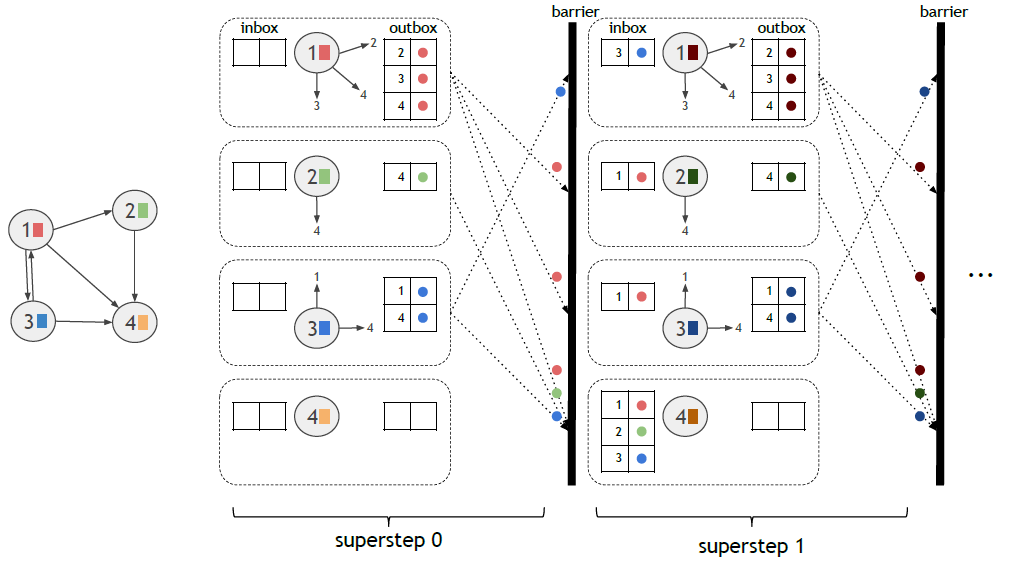


图1. VC模型中的执行超步和信息传递

1. Scatter-Gather 迭代模型

Scatter-Gather迭代过程分为多个superstep，superstep之间是同步的，在一个superstep期间，图中的结点要执行下面两个过程：

1. Scatter: 产生发送给邻居结点的消息；
2. Gather: 根据收到的消息更新结点值。

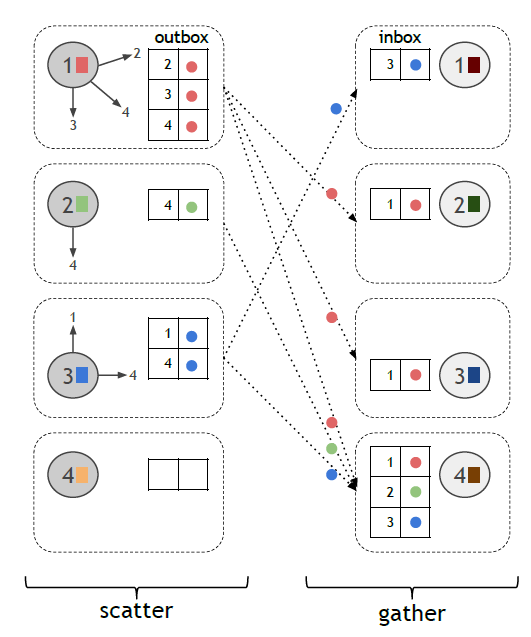


图2. SG模型的一次迭代过程

1. Gather-Sum-Apply迭代模型

Gather-Sum-Apply迭代过程分为多个superstep，superstep之间是同步的，在一个superstep期间，Gather-Sum-Apply分为三部分：

1. Gather: 结点收集每个邻居结点和对应邻边产生一个值，即有多少个邻居结点就会收集多少个值。
2. Sum: 根据一定规则，将Gather阶段产生的多个值聚合为一个值
3. Apply: 根据Sum阶段产生值更新结点值。
4. Scatter: 这一步是可选的，用于更新边的状态。 Flink Gelly未支持Scatter。

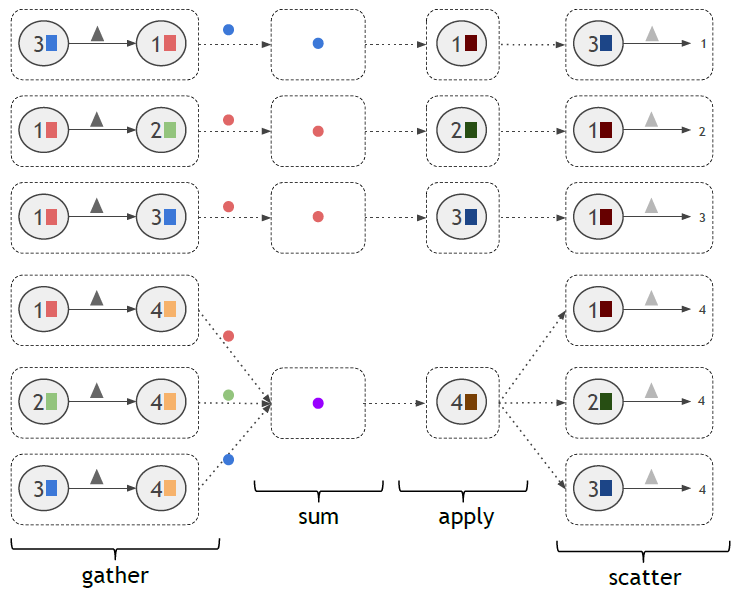


图3. GSA模型的一次迭代过程

1. 三种迭代模型对比

Vertex-centric迭代模型是最通用和普遍被实现的模型，它允许任意的结点更新函数和通信逻辑。Scatter-Gather迭代模型将产生数据和更新数据逻辑分离，它比Vertex-centric迭代模型严格，结点更新逻辑和通信逻辑都受到限制。Gather-Sum-Apply迭代模型和Scatter-Gather迭代模型相近，区别是GSA的Gather阶段作用于边，而SG作用于顶点，同时GSA只和neighborhood通信，而Scatter-Gather可以和任意结点通信，总的来说，GSA是限制最多的迭代模型。

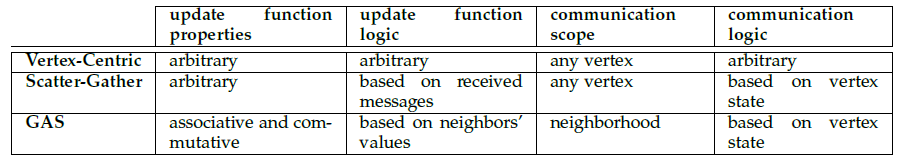


表1. 三种迭代模型的对比

### Flink Gelly已有算法分析测试

|  |  |  |
| --- | --- | --- |
| **现有算法** | **算法简介** | **是否测试** |
| Connected Components | 基于SG实现的弱连通分量算法，不考虑边方向 | Y |
| Strongly connected components | 基于SG实现的强连通分量算法，考虑边方向 | Y |
| GSA Connected Components | 基于GSA实现的弱连通分量算法 | Y |
| PageRank | 基于SG实现的PageRank算法 | **Y** |
| GSA PageRank | 基于GSA实现的PageRank算法 | Y |
| Single Source Shortest Paths | 基于SG实现的单源最短路径算法 | Y |
| GSA Single Source Shortest Paths | 基于GSA实现的单源最短路径算法 | Y |
| Triangle Count | 统计一个图中，三个结点能构成三角形的个数，忽略边的方向 | Y |
| Triangle Listing | 统计一个图中，所有能构成三角形的顶点组合，可以选择是否考虑边的方向 | Y |
| Triangle Enumerator | 统计一个图中，所有能构成三角形的顶点组合，忽略边的方向 | Y |
| Community Detection | 基于《Towards real-time community detection in large networks》实现的社区发现算法 | Y |
| Label propagation | 基于《Near linear time algorithm to detect community structures in large-scale networks》实现的标签传播算法 | Y |

### 流式图计算模型

###### 5.4.1基于状态更新的动态图计算模型

传统的图计算模型（例如BSP模型）中，图数据是静态的，即在计算的过程中图数据不会发生变化；本文提出的基于状态更新的动态图计算模型，能够很好的解决动态图计算问题，它将动态图在每个时刻抽象成一个状态（State），将流动的图数据抽象成一系列事件流（Event Stream），事件（Event）触发了图由一个状态转换（Transform）成另一个状态。



图 4. 基于状态更新的动态图计算模型

基于状态更新的动态图计算模型有如下几个定义：

（1）状态（State）：反应了图当前的特征信息，这些特征信息可以以顶点为单位进行体现，也可以由用户自定义的特征信息来体现，状态是由因子（Factor）组成，因子是指组成状态的基本单位，如状态可以以顶点的方式组织，那么这里的因子就是顶点。需要注意的是，状态反应了用户的关注点，虽然是根据流动的图数据而动态计算生成的，但并不等价于图数据本身，即状态不直接存储原始的图数据，而只存储用户关心的图的某些特征信息。这使得系统无需存储庞大的原始图数据，只需要存储设计精巧的状态信息即可反应图的特征信息。例如当统计图的边数，此时State可以设计为一个计数器，该计数器反应了当前时刻流入系统中的图的边数，每次新增或者删除边时，增加或减少这个计数器的值，即可实时反应当前图的边数信息。

在这里，本文将状态抽象成一个接口，该接口的基本方法表见表2，用户可以扩展该接口来实现更加复杂的状态信息。

表2. State接口方法表

|  |  |
| --- | --- |
| **方法签名** | **方法作用** |
| State GET-STATE(Factor) | 获取指定因子的状态 |
| SET-STATE(Factor, State) | 设置指定因子的状态 |
| SET-STATE(State) | 向图中添加一个状态 |
| Map GET-STATE() | 获取整个图的状态 |

（2）事件（Event）：触发图由T1时刻的State1转换为T2时刻的State2的事件，例如在T2时刻新增加了一条边，将使得图由State1经过某种运算得到State2。事件是由事件值（Event Value）和事件类型(Event Type)组成。如“增加一条边e(v1,v2)”这个事件中，e(v1,v2)是事件的值，“增加”是事件的类型。一般来说，事件的值分为两种：（顶点编号，顶点的值）和（边起点，边终点，边值）；而事件的类型分为三种：新增（ADD），删除（DELETE），更新（UPDATE）。这样总共可以组成6种事件：新增边，删除边，更新边；新增顶点，删除顶点，更新顶点。这6种事件基本涵盖了所有的图变化的情形。事件的接口方法表见表3.

表3 Event接口方法表

|  |  |
| --- | --- |
| **方法签名** | **方法作用** |
| Value GET-VALUE(Event) | 获取指定事件的值 |
| Type GET-TYPE(Event) | 获取指定事件的类型 |

（3）转换（Transform）：由事件触发的图的更新过程，即图是如何根据相应的事件来由State1转换成State2。如图6所示，在T1时刻图的状态为State1，在T2时刻，图接收了一条边e，这个事件会触发图状态转换函数（Transform），将图的状态转换为State2。转换函数是动态图计算模型中的计算逻辑，详细定义了图如何根据到达的事件，从一个状态转变成另外一个状态，可以称之为状态更新的图计算模型的驱动程序，驱动图从一系列的事件流转换成一系列对应的状态流。

###### 5.4.2 HzGraphFlow：基于Hazelcast的流式图计算框架

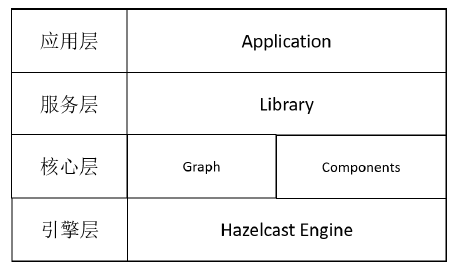
设计背景

基于Flink Gellly的GraphFlow框架实现SSSP和PR算法时遇到困难，因为Flink没有提供灵活的分布式数据结构来存储图的状态。

之所以DD算法和TC算法能够在Flink上实现，是因为这两种算法都是局部算法,而且不需要进行迭代计算，因此很容易通过分流的方式存储局部计算结果，并且进行局部计算即可。而SSSP和PR算法是要在整个连通子图内多次迭代的进行计算，如果没有分布式数据结构来存储每个节点的状态，很难在整个连通子图内共享信息。而Hazelcast提供了这样的数据结构，因此希望通过在Hazelcast上构建整套模型，并且实现该四种算法。

架构实现

如图4所示，定义了hzGraphFlow的动态图计算模型的运行过程：系统每次从EventStream中读取一个Event(如增加一条边这样的事件),利用该事件和图的原始状态State1，在用户自定义的Transform函数的驱动下，转变为另一个状态State2。因此需要实现该模型的这三个基本组件，并且在该组件之上定义图的算法。整个系统的架构如下图所示：



* **Application**：面向用户的上层运用，这些运用涵盖了典型的使用场景，例如链接分析、欺诈检测、社区发现等，是针对某个具体问题的具体应用；
* **Library**: 框架提供给用户使用的丰富的库函数和图算法，诸如Degree Distribution, Triangle Count, Single Source Shortest Path, PageRank等算法包都会在该层中实现；
* **Graph& Components:** 提供了图的基本定义和组件的基本定义。该层是系统的核心层，也是模型的实现层，用户可以实现该层定义的接口来实现自定义的图算法。
* **Engine**: 最底层的具体的引擎，本文使用Hazelcast这样的分布式数据结构框架作为整个系统的底层存储引擎。

算法实现

1. Degree Distribution

节点的度分布算法，是用来统计无向图中各个节点的度。如图2.所示，图中数字表示各个节点的度，当增加一条新边时，将这条边的两个顶点的度各加1即可。

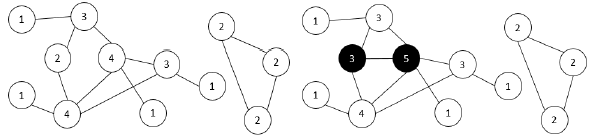


图5. DD Algorithm

1. Triangle Count

TC算法是用来统计无向图中不同三角形的数目。如图3所示，图中节点编号表示节点拥有三角形的数目。当增加一条边时，找出这条边的两个顶点的公共邻接点，即为新增的三角形的数目。

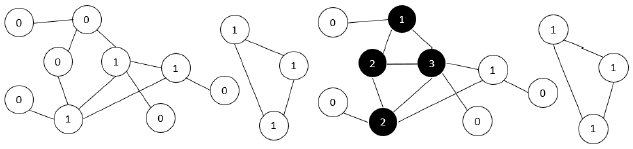


图6. TC Algorithm

1. Single Source Shortest Path

单源点最短路径算法，是在有向图中，给定一个源点，就该源点到图中其他各点的最短路径。下图中黄色顶点为给定的源点，白色顶点为再处理新增边之前已经存在而且处理好的顶点，黑色顶点和边为新增的顶点。当增加一条边时，这条边有三种可能：

1. 这条边的两个顶点都是最新出现的

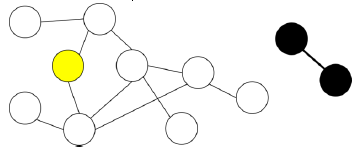


图7. SSSP：这条边的两个顶点都是最新出现的

因为新增的这条边的两个顶点都是最新出现的，因此原图中的任何顶点都无法与之建立连接，即这两个顶点是不可达的。所以他们的SSSP值为无穷大。

1. 这条边的两个顶点有一个是最新出现的

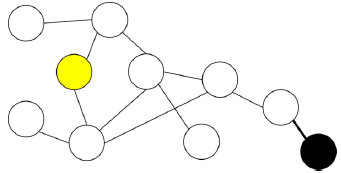


图8. SSSP：这条边的两个顶点有一个是最新出现的

假设新增的这条边为（v1,v2,distance），如果v1是原图中已经有的，v2是新增加的节点，v1指向v2,则v2的SSSP值为v1+distance；反过来，如果v1是新增加的，v2是已经有的，则图中没有节点指向v2，即v2是不可达的，SSSP值为无穷大。

1. 这条边的两个顶点都是原图已经存在的顶点

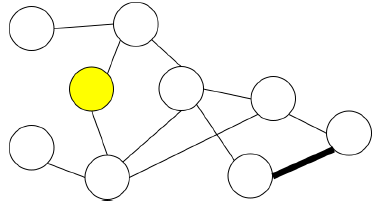


图9. SSSP：这条边的两个顶点都是原图已经存在的顶点

1. PageRank

DD,TC和SSSP算法能够用流式的增量计算模型是显而易见的。但PageRank是否可以呢？我们假设原图为𝐺0,原图的初始状态为𝑥0，新增一条边之后，现在的图为𝐺1，新增节点的初始状态为𝑥1。PageRank算法为𝑓。

则有

𝑅0 = 𝑓(𝑥0, 𝐺0)

𝑅1 = 𝑓(𝑅0 + 𝑥1, 𝐺1)

𝑅∗ = 𝑓(𝑥0 + 𝑥1, 𝐺1)

𝑅0表示在原始状态为𝑥0的图𝐺0中进行若干次迭代之后的计算结果；𝑅1表示以原始图的计算结果𝑅0和新增节点的初始状态𝑥1作为新图𝐺1的初始状态，经过若干次迭代之后的运行结果；𝑅∗表示直接在最初的𝑥0和𝑥1上进行若干次迭代之后的运行结果，如果有𝑅1=𝑅∗，则证明PageRank算法可以用流式的增量计算模型进行计算，而且能够得到准确结果。

Page和Sergey Brin，他们从理论上证明了不论初始值如何选取，PageRank算法都保证了计算结果能够收敛到他们的真实值。因此PageRank算法可以使用流式的增量计算模型。如果初始值越接近真实值，那么算法的收敛就越快，而利用上一次的计算结果作为下一次的初始值，显然要比从头开始计算收敛的要快。

算法测试

上述四种算法，均完成了功能性的测试，测试结果无误。后续会展开进行性能测试。测试过程带后续补充。

###### 5.4.3 GraphFlow：基于Flink的流式图计算框架