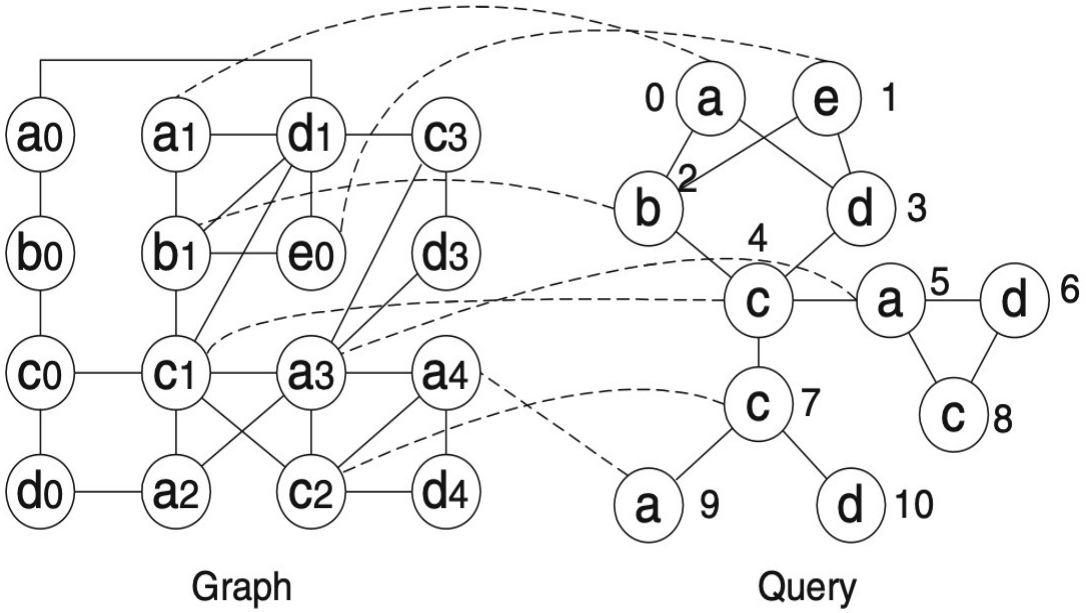
动态图匹配算法介绍

2020.12.07 杨荣键

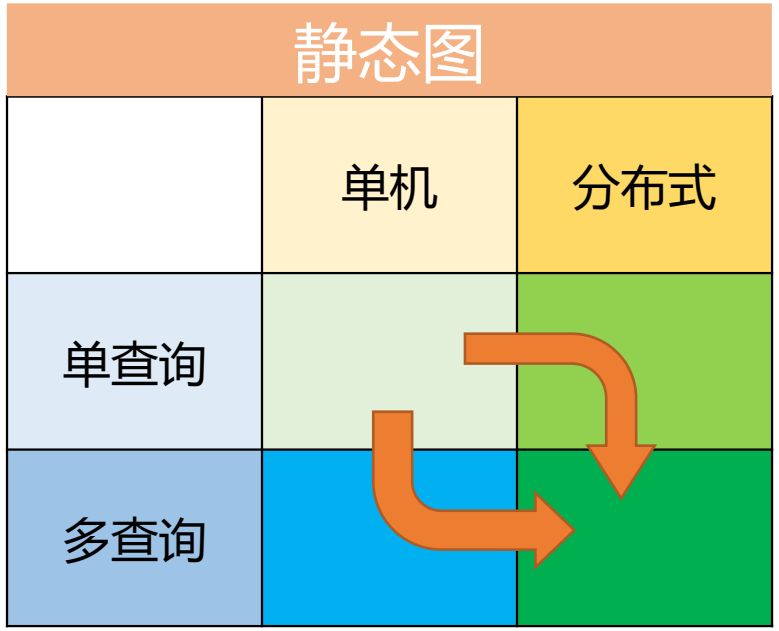
子图匹配

给定一个查询图 Q 和一个数据图 G，子图匹配被定义为“找到数据图 G 当中所有与查询图 Q 同构的子图。”

图示 [1]



研究领域



[1]: VLDBJ16. Jun Gao， Chang Zhou， Jeffrey Xu Yu. Toward continuous pattern detection over evolving large graph with snapshot isolation.

动态图匹配

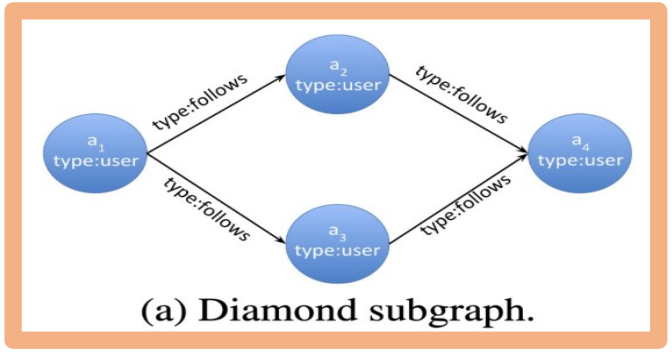
给定一张查询图，在**动态变化的数据图**当中实时地查询和监控匹配结果的变化，这就是动态图匹配。

动态图

动态图被定义为一张**初始的图**和一系列的**图更新流**，这些更新可能是边的增加也可能是删减。

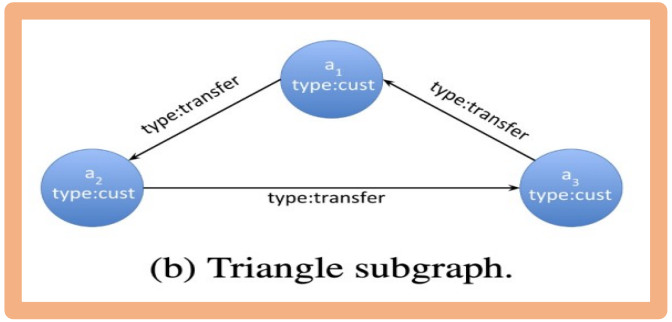
两个例子 [1]

推特的 MagicRecs 推荐软件 [1] 可以在推特的 who-follows-whom 数据图中连续地探测一种“钻石”图，如图 1a 所示。“钻石”图有一个用户 a1，他关注了另外两个用户 a2 和 a3，同时这两个用户同时关注了 a4 。那么，一旦这样的“钻石”图被检测到了， MagicRecs 就会因为 a1 关注的两个用户都对 a4 感兴趣，进而向 a1 推荐用户 a4 。



(a) Diamond subgraph.

考虑一个金融欺诈检测的引用。在如图 1b 所示的查询图当中，个人消费者是作为点，金钱的交易活动用边来表示。这个应用能够探测出一个循环交易，这可能表示某种洗钱活动，并把这个案例报告给欺诈探测的专家。该查询对应的是一个连续的 cycle 查询。



(b) Triangle subgraph.

[1]: SIGMOD17. Kankanamge C , Sahu S , Mhedbhi A , et al. Graphflow: An Active Graph Database.

本文所说的动态图（或者说图流）上的子图查询（子图匹配）问题，由于其主要应用在于“检测”（detection），因此涉及到的算法主要分为两类：一种仅仅考虑数据图中是否存在感兴趣的子图模式（Subgraph Detection）；另一种是将感兴趣的子图模式实时地提供其匹配结果（Subgraph Matching）。

Graph Model

给定查询图 Q = (V, Eq)，数据图 G = (U, Ed)，其中 V 表示查询图上的结点，U 表示数据图上的结点，Eq 表示查询图上的边，Ed 表示数据图上的边。

数据图 G 的图流 S = (u1,u2,...,ut)，这是一个有序的更新系列，其中 ui 表示新增的边或删减的边。

Subgraph Detection

定义 m 为查询图 Q 向数据图 G 的映射关系。subgraph detection（或 pattern detection）将返回 m 存在与否。

Subgraph Matching

子图匹配会在数据图应用图流的过程中，不断更新匹配结果（增加新的匹配结果或删减原有的匹配结果），并返回给用户（或指令发出端）。

D-GJ

D-BJ

SSD

SJ-T

TRIC

TF

Delta-GenericJoin 、 Delta-BiGJoin

- SIGMOD17. Kankanamge C , Sahu S , Mhedbhi A , et al. **Graphflow: An Active Graph Database**.
- PVLDB18. Ammar K, et al. **Distributed evaluation of subgraph queries using worst-case optimal low-memory dataflows**.

背景信息

Setting

两个算法都是运用于带标签的有向图当中，但是：（1） Delta-GenericJoin：应用于单机版查询任务中；（2） Delta-BiGJoin：应用于分布式查询任务中。

Based on

本文的算法是基于 GenericJoin 的基础上，也就是说，这是 vertex-at-a-time （或者说基于 Exploration ）的查询方法。

Main Idea

- 根据 GenericJoin 的想法，算法按照某个顺序，一个点一个点地进行匹配。下一个匹配的点将从已匹配点的邻居当中确定。
- Delta-GJ 的思想在于，更新的边若对应为查询图当中的某条边 E ，则重新进行 GenericJoin 匹配时，当匹配到 E 时，不从数据图当中找候选集，而以更新边代替。

$$dQ_1 := \Delta R_1, R_2, R_3, \ldots, R_n$$

- Delta-BJ 的思想在于，不按照原始的顺序进行匹配，而从更新的位置开始进行匹配。

D-GJ

D-BJ

SSD

SJ-T

TRIC

TF

single-sink DAG （SSD）

VLDBJ16. Gao J , Zhou C , Yu J X. **Toward continuous pattern detection over evolving large graph with snapshot isolation**.

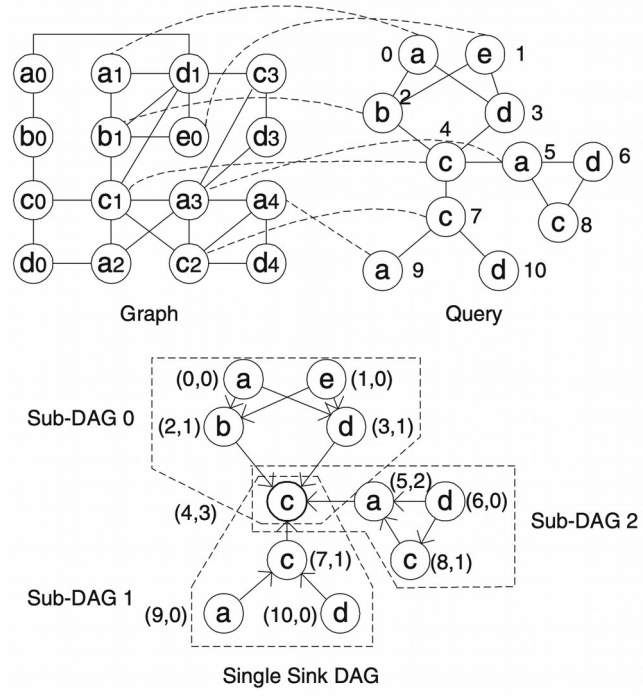
背景信息

Setting：本文研究的是带标签的无向图，查询过程是在分布式环境下进行的。

Based on：本文选择在 vertex-centric 的分布式框架当中，使用的是基于 Exploration 的方法。

Main Idea

- 首先将查询图转化为一个带有唯一 sink vertex 的 DAG （SSD）。
- 数据图当中，与查询图相符的结点将对其邻居依次进行传输消息，当最后一个结点 (sink vertex) 收到消息时则可以确定：在数据图当中存在满足查询图的子图。
- 由于数据图是处于动态变化当中的，因此消息传输也是一个持续的工作， sink vertex 会不断进行报告。



D-GJ

D-BJ

SSD

SJ-T

TRIC

TF

Subgraph Join Tree (SJ-Tree)

EDBT15. Choudhury S, Holder L, Chin G. **A Selectivity based approach to Continuous Pattern Detection in Streaming Graphs.**

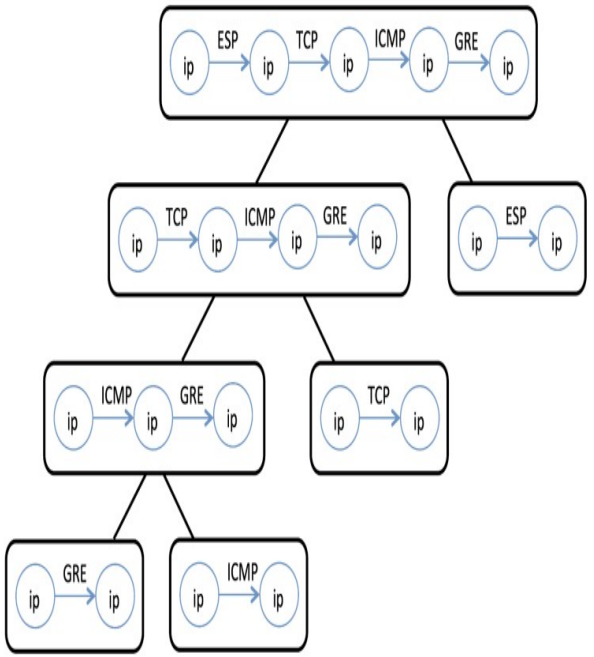
背景信息

Setting：本文研究的是带标签的有向图，并且只考虑边增加的情况。

Based on：基于 join 的查询方法。

Main Idea

- 将查询图拆为多个子图，并对每个子图分别进行搜索，得到每个子图的匹配结果。
- 然后将子查询图的 join 过程建成一棵二分树，其中叶子结点即为子查询图。该树的根节点对应的就是完整的查询图。
- 动态图中，每次更新的叶子结点会逐层网上进行 join 操作，并不断更新其祖先结点。若根节点发生了更新，则查询图的匹配结果会进行更新。



TRIe-based Clustering (TRIC)

EDBT20. Zervakis L, Setty V, Tryfonopoulos C, et al. **Efficient Continuous Multi-Query Processing over Graph Streams**.

背景信息

Setting

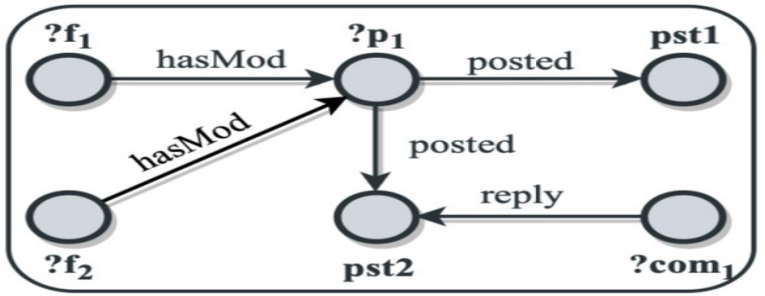
本文研究的是带标签的有向图，并且是多查询问题。

Based on

本文在动态图方面的基本思路与 SJ-Tree 本质上无异，其主要内容着重于呈现如何解决多查询的问题。

Main Idea

- 将每一张原始查询图 Qi 都转换为一组路径集合，并且需要保证这些路径可以覆盖 Qi 的所有点和边。
- 利用路径之间的共性来进行聚类，形成多个树结构。
- 在面对更新时，我们能更快地找出受影响的路径，然后进行 join 操作形成新的匹配结果。



$$P_1 = \{?var \xrightarrow{hasMod} ?var \xrightarrow{posted} \text{“}pst1\text{”}\}$$
$$P_2 = \{?var \xrightarrow{hasMod} ?var \xrightarrow{posted} \text{“}pst2\text{”}\}$$
$$P_3 = \{?var \xrightarrow{reply} \text{“}pst2\text{”}\}$$

D-GJ

D-BJ

SSD

SJ-T

TRIC

TF

TurboFlux

SIGMOD18. Kim K, Seo I, Han W S. **TurboFlux: A Fast Continuous Subgraph Matching System for Streaming Graph Data**.

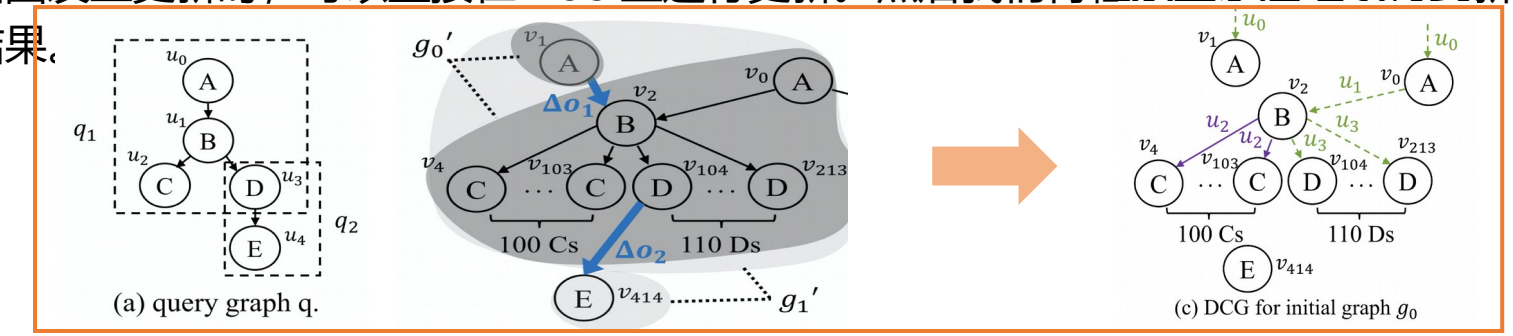
背景信息

Setting：本文研究的是带标签的有向图，并且是单机上的动态查询。

Based on：本文的查询方式可以看作是一种连接操作，但是比连接操作要更加简单。

Main Idea

- 首先我们需要将查询图制作为一棵查询树。
- 对于每个数据顶点 v，我们将相应的候选查询顶点存储为 v 的输入边，这样我们就形成了一张图。我们将这张图保存，并称其为 data-centric graph (DCG)。
- DCG 的边分为**显示边**和**隐式边**，**显示边**意味着数据点 v 所对应的查询点 u 的所有子孙结点都能在数据图中找到匹配，否则其为**隐式边**。
- 当数据图发生更新时，可以直接在 DCG 上进行更新。然后我们再检索**显示边**继续得到新的匹配结果。



算法比较

算法	论文年份	查询图数量	单机/分布式	查询方式	是否存储额外信息
Delta-GenericJoin	2017	单查询	单机	vertex-at-a-time	否
Delta-BiGJoin	2018	单查询	分布式	vertex-at-a-time	否
SSD	2016	单查询	分布式	vertex-at-a-time	部分结点保存传输信息
SJ-Tree	2015	单查询	单机	edge-at-a-time	保存部分匹配结果
TRIC	2020	多查询	单机	edge-at-a-time	保存部分匹配结果
TurboFlux	2018	单查询	单机	edge-at-a-time	保存 DCG