# 摘要

第一次提出把Q分解成多个模式子图。

## 介绍

1.1子图匹配：

1.2挑战：图的存储特性，没有locality。导致不能直接访问，不能在线分析，现有挑战如何在线分析，大图。

1.3贡献：

1.适用于大规模图数据，不需要index，节省了空间。

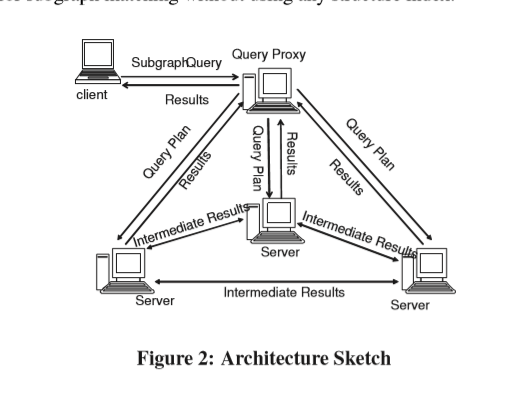
2.对于query的 partition，提出了查询优化策略 来避免额外的代价。

3.内存消耗小

## 2.背景

1. 子图匹配定义
2. Trinity memory cloud

Trinity是一个分布式图引擎。



## 3.exploration 和join

Exploration（先匹配a，再匹配b，不断的往外拓张）的最大优点，1）没有结构化的index 2）没有join 3）中间结果变少了

join:先分解成多个子图，返回这些子图的查询结果，再join。

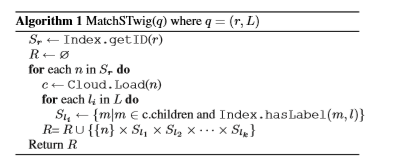
Costly:1.join代价昂贵2.大量中间无用结果

exploration：1.如果图用关系表的方式存储，仍然会有join，本文这种存储方式避免了2.naïve的exploration有问题，需要一个个的做3..有环图仍然要join。4.有些case无论join还是exploration代价都很大

1. 本文提出来的则是考虑了join和exploration特点。使用join的方法去做查询计划，使用exploration的方法去避免 join过程中产生无用的候选子集（利用了之前的的信息过滤）。

## 4.framework of approach

**4.1 基础单元 STwig**



如何匹配STwig:先找到和r匹配的所有节点。针对S内的每个节点c，找到其邻居满足定义L的所有节点，找到所有结果R。

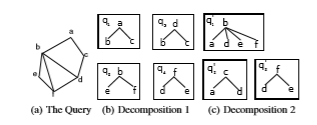
**4.2子图匹配**

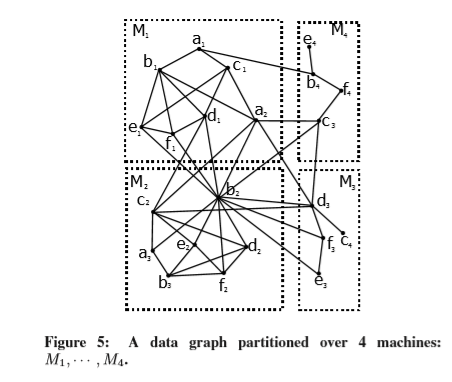
1. Query Decomposition 和STwig Ordering

分解策略和排序策略第五部分讲

2．Exploration

基于上一步的binding信息，探索新的stwig的过程中做了过滤。





3.join

做了2个优化。join order selection和pipeline join

**1. 获得了所有的exploration后，再做最后的join。**

**那么问题来了？为什么不直接在探索了之后就做join操作，而是推迟到所有的exploration都做完了再join。 可以选择join顺序减少代价。**

**2. block-based join pipeline ()**

**4.3 分布式 并行子图匹配**

Query和stwig order没办法并行。只有exploration并行。

1. 获得别的机器的G(qi)





获得哪些机器？不同机器R不相交，F尽可能小。 后文的优化部分讨论F如何确定。

第一次执行的时候，F=空，就满足了。

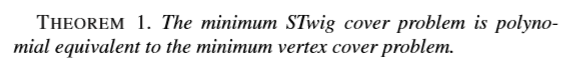
以后执行的时候，F后文判断。

## 5.优化部分

**5.1查询优化：**

STwigs的数量决定了join的数量，Stwigs的数量越多，交流的代价越大，想办法变少。

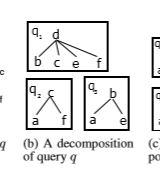
PROBLEM 1. Let G be a query graph. Let S={T1,···, Tn} be a set of STwigs such that any edge of G belongs to one and only one Ti ∈ S. We call S an STwig cover of G. The problem is to ﬁnd the minimum STwig cover of G.



找到minimum vertex cover 是np-hard。

二阶近似：随机找边，移除顶点和相邻边，直至完成。

**5.2 顺序优化**



q1,q2,q3i is a better order than hq2,q1,q3i,

尽可能让当前的stwig受到之前的约束。因此在上面的二阶近似算法上：

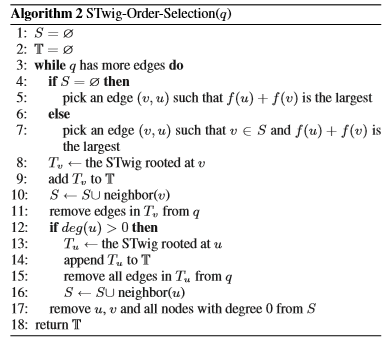
1. 选择已经和之前选择的边相连的 2.选择和具有high selectivity的顶点 很近的顶点。

1. Select edges that already connect to previously selected edges.

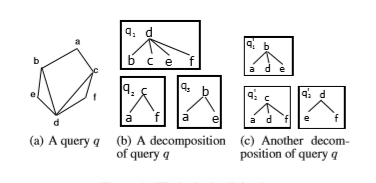
2. Select edges incident to nodes with high selectivity.

想法是下一次选择尽可能和上一次选择受到限制。

第一条规则保证了受到前面的bound。第二条规则保证了减少中间结果，(和高selectity相连的顶点，，那么选择了这个顶点后，这个顶点和high selectity匹配的可能性就很大了 。)



order selection和query一起的过程。



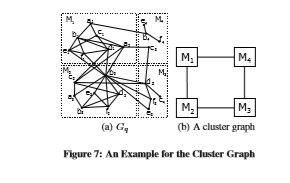
假设每个label出现次数都是10.f(d)=0.4,f(c)=0.3，第一条边选择d-c。也就是形成了（d，（b,e,c,f））（c，（a,f）） .这个时候S=(a,b,e)

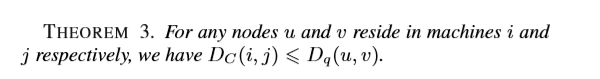
计算f-values and sorting nodes by f-values have O(nlogn)。N轮的话O(n2 logn).

**5.3 Head STwig and Load Set Selection**

The Cluster Graph

G和Gq的结果一样。移除肯定不需要的边，形成Gq。

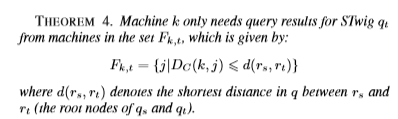




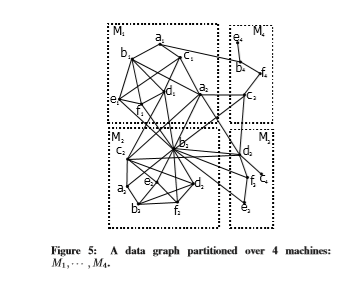
u和v所在机器之间的距离，肯定是小于等于u和v之间的距离。

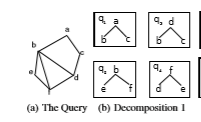
**Load Set：**

需要访问的机器。



2个子query的根之间的距离d(rs,rt)。如果2个机器之间的距离比2个根之间的距离还要大，那么肯定是不可能有任何关系的。因此只要获得 机器距离小于根距离的机器。





假定q1被选择当作head swig，那么a和b之间的距离是1.那么M3的q3结果就不需要被load到机器M1上了。

**Head STwig Selection：**

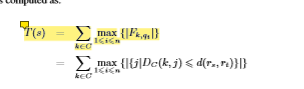
优化问题。

一次交流往往可以传输多个stwig。因此我们考虑的是最小化总的交换的轮数。

每个机器k的交换轮数 是 。



也就是最小化



怎么计算呢?

计算查询q的2点间的距离。矩阵的形式。代价很小的。





d（i）是节点i到别的顶点的最大距离。

选择所有d(i)里最小的 d(s)。

## 总结：

第一个分布式的图匹配算法。实验很low。主要是提出了exploration和join的结合。

Query的分解计划。

从哪个子query开始执行query计划。能够最小化交流的机器数目代价。