

## 第一章 经典图查询算法

本章将详细介绍几种经典的图查询算法。由于基本的图查询模式均为“过滤-验证”，所以要提高查询效率，只能从两方面优化。一是过滤阶段优化，二是验证阶段优化。众所周知，图是结构画的数据，目前并没有一个统一高效的索引机制可以实现较好的查询结果。因此大多数学者都会在过滤阶段选用不同的索引方式来得到更好的查询结果。而在验证阶段，由于图同构是个 NP-hard 问题，所以验证会消耗大量时间。如何快速检测同构，或将同构转为其他非复杂多项式的一半问题也是学者们关心的一个热点课题。本章介绍的几种查询方法均为过滤阶段的优化。

### 1.1 图精确查询算法

本节将详细介绍子图查询中的 *GraphGrep* 算法<sup>[1]</sup> 和 *gIndex* 算法<sup>[2]</sup>。

#### 1.1.1 GraphGrep 算法

2002 年 ShaSha 教授等人提出了 *GraphGrep* 算法<sup>[1]</sup>。*GraphGrep* 算法是基于特征索引方法中的第一个经典算法，它采取典型的“过滤-验证”框架，*GraphGrep* 算法中只讨论过滤阶段。由于基于结构的算法中对图的顺序扫描代价太高，*GraphGrep* 提出基于路径的过滤方法，来减少候选集大小。

其算法的主要流程是这样的：

1. 构造索引: 首先将节点和边利用哈希存成两个二维表作为未来筛选用特征之一，然后枚举图数据库中所有长度不大于  $l_p$  的路径  $v_0, v_1, v_2, \dots, v_k, (k \leq l_p, \forall i \in [1, k-1], (v_i, v_{i+1}) \in E)$ ，然后将这些路径与图的关系存为一个二维表作为索引。表的每一行代表每一个图，每一列为一个路径，利用简单哈希确定路径对于行号，每一个单元格代表在该图包含该路径几条，如表1.1所示。
2. 解析查询: 如同图数据库，首先将节点和边利用哈希存成两个二维表，然后枚举查询中的所有长度不大于  $l_p$  的路径，哈希存在一个列表中。

Key	$g_1$	$g_2$	$g_3$
h(CA)	1	0	1
...			
h(ABAB)	2	2	0

表 1.1 GraphGrep 索引

### 3. 数据库过滤:

- (a) 利用节点信息过滤: 如果查询图中的某一节点在数据库中一图里未出现, 则此图一定不会包含查询图, 因此可以删去。
- (b) 利用边信息过滤: 同节点过滤, 如果某条边未出现, 则一定不会包含查询图, 可删去。
- (c) 利用路径个数进行过滤: 如果查询图中某一路径个数大于数据库中一图的此路径个数, 则此图一定不会包含查询图, 可以从候选集中删去。

### 4. 子图查询: 利用标号进行路径合成, 从候选集中删去所有未能成功合成的候选图。剩下的就是 GraphGrep 算法返回的候选集。后续再加以图同构验证即可得到最终子图查询结果。

## 成过程

由于 GraphGrep 算法是基于路径的, 编写十分简单, 但是路径数目过多, 造成索引集很大, 建立十分费时, 子图查询过程中也有大量运算量, 因此效果有限。

### 1.1.2 gIndex 算法

2004 年, Yan 教授等人提出了 *gIndex* 算法<sup>[2]</sup>。*gIndex* 算法是以频繁子图结构作为索引特征的子图查询算法。它采用动态支持度 (*size-increasing support*) 和区分度片段两种方法来优化算法, 获得性能更好的索引。

## Index 详

其算法主要流程如此:

### 1.

## 1.2 图相似性搜索

本节将着重介绍两种相似性搜索方法，分别是 *G-Hash* 算法<sup>[2]</sup> 和 *C-tree* 算法<sup>[3]</sup>。

### 1.2.1 G-Hash 算法

*G-Hash* 算法是 Wang 教授等人于 2009 年提出的一种图相似性搜索方法。相比较其他近似搜索方法，这种方法更加稳定高效。*G-Hash* 开创性地采用了简化包表示 (*Reduced Bag Represent*) 来表示每个节点特征，并表示成字符串，Hash 存储作为索引。并利用小波匹配核函数 (*Wavelet Graph matching kernels*) 来计算节点间的相似度，从而得到和查询图最为相似的  $K$  个图。但是 *G-Hash* 算法的准确度和速度完全取决于相似度度量函数，因此一个好的相似性度量方法尤为重要。

算法主要流程如下：

1. 索引构建: 将图数据库中每幅图用简化包表示，然后将每个节点特征变为字符串，利用 Hash 存成索引。如例1.1所示, 就是一个图建成索引过程。需要注意的是在例子中，我们只有三个标号，所以特征矩阵只有四列。但是在实际中当我们选取节点标号数目作为特征时，需要先统计出图数据库中有的所有标号，这样才好做归一化存储。

**例 1.1.** 图1.1就是一个 *G-Hash* 构建索引的实例。图1.1(a)是需要存储的一幅图。我们选取节点标签和邻接节点各个标号的个数作为度量特征，所以一共有四个特征。首先统计各个节点周围各个标号的数目，结果如图1.1(b)所示。然后用小波函数提取出实际特征值。举例而言，对于  $v_3$  在用  $h = 0$  的小波函数提取后，局部特征是  $[B, 2, 0, 1]$ 。将这些表示特征值的字符串利用 Hash 找到对应位置，然后将对应节点标号填到此位置，最后生成的哈希表如图1.1(c)所示。而整个数据库生成的哈希表就如图1.1(d)所示。

2. 查询过程: 将查询图也同数据库中图一样用简化包表示。然后计算其中每个节点字符串和图数据库中每个字符串的相似性，乘以每个图中此字符串出现的次数，得到这个节点和每幅图的相似度，求和即可得到总的相似度。排序得到最相近的  $K$  个。

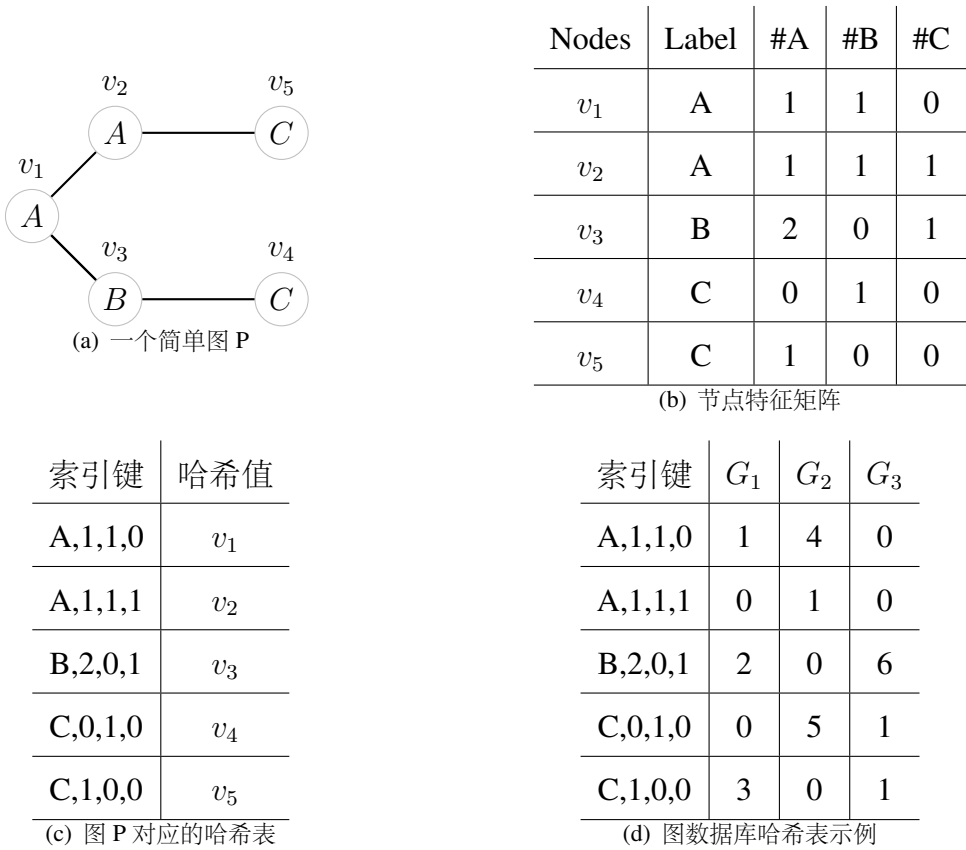


图 1.1 一幅简单示例图

图是 G—Hash 查询的一个完整例子。从 XXXX

此方法还支持动态增删图数据，如例1.1因为其用字符串作为索引键，所以如果增删的图不改变原有的特征情况，在例1.1中就是标号的个数，那么在添加时需要添加数据库中的一列，在删除时也只需要删除一列即可。不需重建整个索引。

添加 G-Hash+  
中那副图

1.2.2 Closure tree 算法

C-tree

## 参考文献

- [1] Rosalba Guigno and Dennis Shasha. Graphgrep : A fast and universal method for querying graphs. 2002.
- [2] Xifeng Yan, Philip S.Yu, and Jiawei Han. Graph indexing : A frequent structure-based approach. *ACM*, 2004.