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[?] 和 *gIndex* 算法[?]。

1.1.1　GraphGrep 算法

2002 年 ShaSha 教授等人提出了 *GraphGrep* 算法[?] 。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 $v_0, v_1, v_2, ..., v_k, (k \leq l_p, \forall i \in [1, k-1], (v_i, v_i+1) \in E)$，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。

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，尤其在内存有限的情况下哈希常发生冲突现象，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[?]。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。

其算法主要流程如此:

1. 利用数据挖掘中的相关算法挖掘出频繁子结构，并用最小支持阀值 min_sup 筛选出频繁子结构作为特征片段

2. 利用 DFS 序列化编码将特征片段序列化，然后存成一颗前缀树称为 $gIndex$ 树作为索引，如图1.1所示。并记录每个图包含哪些特征片段。

3. 对于给定查询图 q，找出其所有特征片段，然后对包含此片段的图取交集即为最终结果。

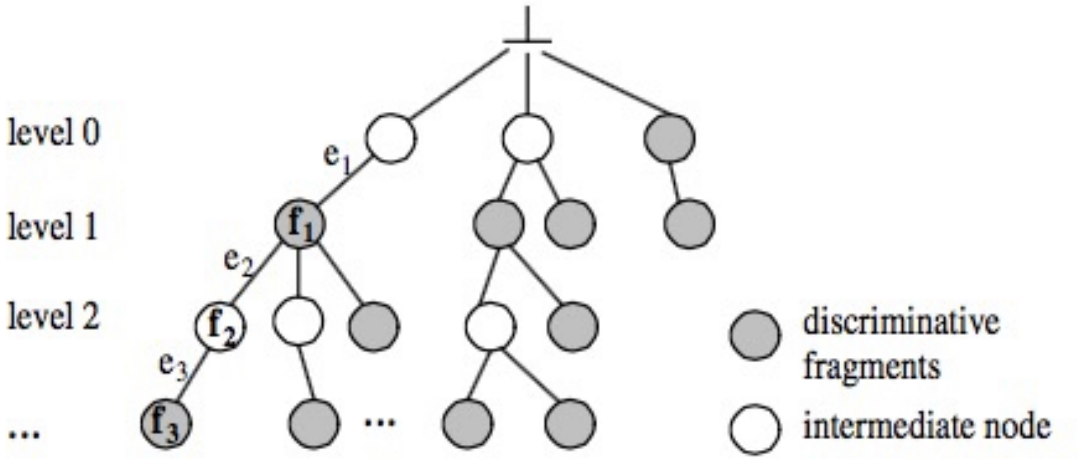


图 1.1 gIndex 树

gIndex 算法是首个以频繁子结构为索引特征的经典算法，其利用了图的结构化信息，使得效率大大提高。但是由于记录的为结构化的子图信息，对空间要求较大，同时特征片段的筛选使得其只携带了自身的结构特点，丢失掉的结构信息也会对算法效率造成一定影响。

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[?] 和 *C-tree* 算法[?]。

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

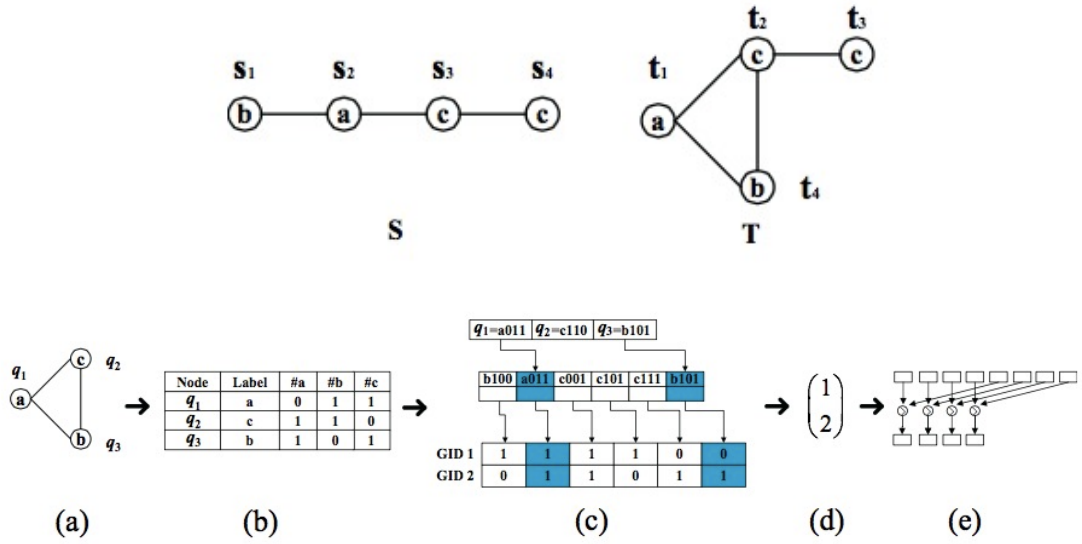
的相似度，从而得到和查询图最为相似的 K 个图。但是 G-Hash 算法的准确度和速度完全取决于相似度度量函数，因此一个好的相似性度量方法尤为重要。

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。

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。

如图1.2, 便是 G—Hash 查询的一个完整例子。

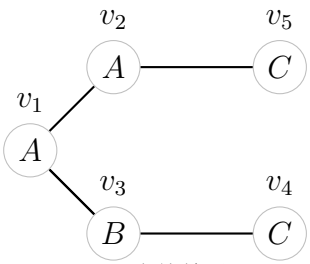


Node	Label	#a	#b	#c
q_1	a	0	1	1
q_2	c	1	1	0
q_3	b	1	0	1

图 1.2 G-Hash 查询示例:(a) 图数据库 S,T 和查询 Q,(b) 查询 Q 的简化包表示,(c) 与哈希存储的图数据库匹对,(d) 计算匹对结果,(e) 排序得出最相似的 k 个图

例 1.1. 图1.3就是一个 G-Hash 构建索引的实例。图1.3(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1.3(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1.3(c) 所示。而整个数据库生成的哈希表就如图1.3(d) 所示。



(a) 一个简单图 P

Nodes	Label	#A	#B	#C
v_1	A	1	1	0
v_2	A	1	1	1
v_3	B	2	0	1
v_4	C	0	1	0
v_5	C	1	0	0

(b) 节点特征矩阵

索引键	哈希值
A,1,1,0	v_1
A,1,1,1	v_2
B,2,0,1	v_3
C,0,1,0	v_4
C,1,0,0	v_5

(c) 图 P 对应的哈希表

索引键	G_1	G_2	G_3
A,1,1,0	1	4	0
A,1,1,1	0	1	0
B,2,0,1	2	0	6
C,0,1,0	0	5	1
C,1,0,0	3	0	1

(d) 图数据库哈希表示例

图 1.3 一幅简单示例图

G-Hash 算法作为经典的相似性搜索算法，其提出利用核函数计算相似度，成功将结构化特征用数学方法转为数值特征，使得搜索效率有明显提升。但是由于其对哈希函数要求严格，造成编码复杂，而核函数的计算也很耗时，使得此方法实际运用并不多。

1.2.2 Closure tree 算法

2006 年 He 和 Singh 提出了一种基于树结构建立索引的查询算法 C-Tree[?],C-Tree 利用启发式图映射方法来判断编辑距离，进而得到近似图。C-Tree 具有以下几个特点[?]:(1) 每个节点都是其孩子的闭包图，只有作为叶子节点的孩子节点是

数据库中的图数据;(2) 除了根节点每个节点至少包含 m 个孩子 $m \geq 2$;(3) 每个节点最多不超过 M 个孩子，$(M+1)/2 \geq m$。

C-Tree 提出了一种利用图闭包计算编辑距离的方法，提出了邻接子树和邻接子图在相似性搜索中的应用，是图相似性搜索领域一个重要基础算法。但是基于图闭包的计算还是略显复杂，仍须研究。