# 第一章　引言

## 1.1　研究背景与意义

随着科学技术的进一步发展，我们正逐步从信息时代走入数据时代[11]，全球的数据量正在以一种前所未有的方式增长着。数据的迅速增长，在给人们带来便捷信息的同时，也带来了一个巨大的挑战——面对日益复杂的数据，传统的查询方法不再有效，无法快速检索出相关联数据。面对大量有意义的数据，无奈于查询手段的限制，只能将其简化再进行处理。现在的大数据现状就好似守着一座金山，却不知如何开采。为了进一步挖掘有效信息，加速查询速度，提高信息价值，各种数据查询技术便应运而生。

其中最为热门的就是图数据库的查询。图作为计算机科学中的一个数据结构，其数据表达能力较强，可以很好得表示

# 第二章　背景知识

## 2.1　图基本定义

**定义 2.1** (标号图[6])**.** 一个可以被四元组 $G = (V, E, \Sigma, \lambda)$ 表示的图称为**标号图** ***(labeled graph)***，其中 $V$ 为有限的节点集合，$E$ 为有限的边集合 $\in V \times V$，$\Sigma$ 是标号集合，$\lambda$ 是一个标号函数用于给各个节点与边分配标号 $\lambda : V \cup E \to \Sigma$。

如图2.1就是一个包含六个节点的标号图。需要注意的是标号与标识的区别，标号是图的固有属性，标识只是为了方便使用人为添加的记号。在图2.1中，$v_i$ 就是标识，而 $A, B, C, D$ 则是标号。

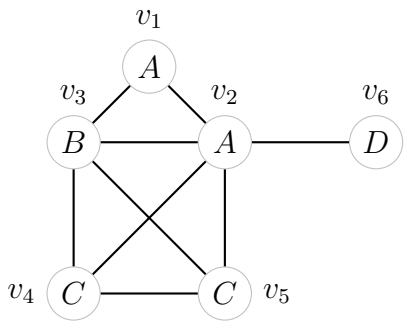



图 2.1 标号图

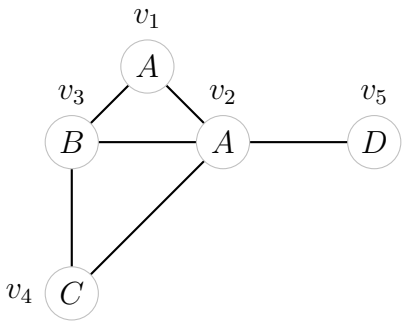



图 2.2 图2.1的子图

**定义 2.2** (子图)**.** 如果一幅图 $G = (V, E, \Sigma, \lambda)$ 和另一幅图 $G' = (V', E', \Sigma', \lambda')$ 有1-1 映射的关系 $f : V \to V'$，那么图 $G$ 就是 $G'$ 的**子图** ***(subgraph)***，用 $G \in G'$ 表示。$f$ 可以有这么几种

- 对于所有 $v \in V, \lambda(v) = \lambda'(f(v))$
- 对于所有 $(u, v) \in E, (f(u), f(v)) \in E$
- 对于所有 $(u, v) \in E, \lambda(u, v) = \lambda'(f(u), f(v))$

换言之，如果一幅图和另一幅图的节点标签，边关系，边标签能一一对应上，那么这副图就是另一幅图的子图。如图2.2就是图一个2.1的子图，节点标签，边关系及边标签均可一一对应，只有标识可以不同。

**定义 2.3** (超图)**.** 如果图 $G$ 是图 $G'$ 的子图，则 $G'$ 是 $G$ 的超图。对于图2.1和图2.2,图2.2是图2.1的子图，所以图2.1是图2.2的超图。

**定义 2.4** (顶点的度[10])**.** 一个顶点 $u$ 的度数是与它相关联的边的数目，记做 $degree(u)$。$degree(v_i) = |E(v_i)|, v_i \in V(i = 1, 2, ..., n)$, 图2.1中 $v_2$ 的度 $degree(v_2) = 5$，图2.2中 $v_2$ 的度为 $degree(v_2) = 4$。

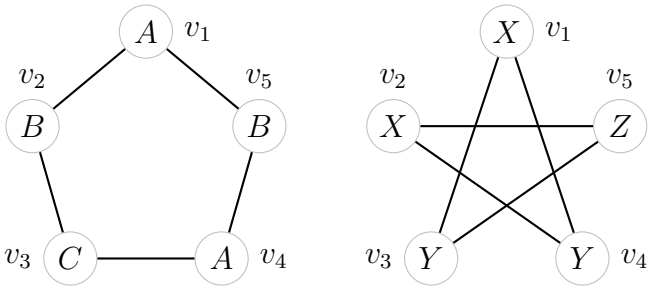
**定义 2.5** (图的尺寸[12])**.** 一般由图中节点数 $|V(G)|$ 定义。因此图2.1的尺寸 *(size)* 为 6，图2.2为 5。

**定义 2.6** (路径)**.** 在图 $G(V, E)$ 中，若从顶点 $v_i$ 出发，沿着一些边经过一些顶点 $v_{p1}, v_{p2}, ..., v_{pm}$，到达顶点 $v_j$, 则称顶点序列 $(v_i, v_{p1}, v_{p2}, ..., v_{pm}, v_j)$ 为从顶点 $v_i$ 到 $v_j$ 的一条路径 *(path)*。如在图2.1中，$(v_1, v_2, v_5)$ 就是一条 $v_1$ 到 $v_5$ 的路径。

**定义 2.7** (图的同构)**.** 给定图 $g$ 和图 $g'$, 若 $g'$ 满足 $g' \equiv_{iso} g$, 则称 $g$ 与 $g'$ 是同构图。$g' \equiv_{iso} g$ 同构，当且仅当存在一个双射函数 $f : V(g) \leftrightarrow V(g')$，其满足下列条件:

- 对于所有 $v \in V, \lambda(v) = \lambda'(f(v))$
- 对于所有 $u, v \in V, (u, v) \in E \Leftrightarrow (f(u), f(v)) \in E$
- 对于所有 $(u, v) \in E, \lambda(u, v) = \lambda'(f(u), f(v))$

如图2.3, 图 $G_1$ 和图 $G_2$ 就是一组同构图, 顶点标号 $A \leftrightarrow X, B \leftrightarrow Y, C \leftrightarrow Z$,$G_1$ 中的边与 $G_2$ 的边也形成双射关系




(a) $G_1$ (b) $G_2$

图 2.3 图的同构

## 2.2　图存储表示方式

图存储表示方法有很多种，常用的有 2 种：**邻接矩阵** *(Adjacency Matrix)* 和**邻接表** *(Adjacency List)*。本节还将介绍一种新颖的图表示方法**简化包表示** *(Reduced Bag Representation)*[6]。

### 2.2.1　邻接矩阵

在邻接矩阵存储方法中，除了一个记录各个顶点信息的顶点数组外，还有一个表示各个顶点之间关系的矩阵，称为**邻接矩阵**。设 $G(V,E)$ 是一个具有 $n$ 个顶点的图，则图的邻接矩阵是一个 $n \times n$ 的二维数组，用 $Edge[n][n]$ 表示，定义为:

$$Edge[i][j] = \begin{cases} 1, if(i,j) \in E \\ 0, else \end{cases} \tag{2-1}$$

增加邻接图

### 2.2.2　邻接表

**邻接表**就是将同一个顶点发出的边连接在同一个称为边链表的单链表中。边链表的每个节点代表一条边，称为边节点。每个边节点有两个域：该边终点的序号以及下一个边节点的指针。

增加邻接

### 2.2.3　包表示法

**包表示法**是王教授等人在 *G-Hash*[6] 中提出的一种方法。这种方法的主要思想是将图中的各个节点特征提取出来形成一个列表，然后将这些特征就作为每个节点的标识。这样整幅图就变成了一个字符串的包，也就是一组字符串。通常，我们用这个节点的标号和其邻接节点的不同标号数目为特征。例2.1所示就是一个常用的包表示方法

**例 2.1.** 表2.1是图2.1的包表示。我们以节点标号和其邻接节点各个标号的个数作为特征，因此有五个特征：本身标号，A,B,C,D 分别的个数。所以加上第一列的标识，表一共有六列。而一共有六个不同节点，所以有六行。我们以 $v_3$ 为例，详细说明下抽取特征的步骤。首先 $v_3$ 的标号是 B, 所以 Label 为 B, 然后和 $v_3$ 邻接的共

| Nodes | Label | #A | #B | #C | #D |
|---|---|---|---|---|---|
| $v_1$ | A | 1 | 1 | 0 | 0 |
| $v_2$ | A | 1 | 1 | 2 | 1 |
| $v_3$ | B | 2 | 0 | 2 | 0 |
| $v_4$ | C | 1 | 1 | 1 | 0 |
| $v_5$ | C | 1 | 1 | 1 | 0 |
| $v_6$ | D | 1 | 0 | 0 | 0 |

表 2.1 图2.1的包表示

有四个节点，分别是 $v_1, v_2, v_4, v_5$，其标号分别为 $A, A, C, C$ 所以 $\#A = 2, \#C = 2$ 其余为 0。节点 $v_3$ 的包表示就是“B,2,0,2,0”, 如表2.1所示。需要注意一点的是，包表示不仅仅这一种表示方法，对于选取什么样的特征并没有限制。不过特征必须是离散的，这样才可以用字符串表示。

## 2.3 图查询类型

根据图之间的包含关系，可将图查询分为子图查询和超图查询。根据图查询的精确程度，可以将其分为精确查询和相似查询。

**定义 2.8** (子图查询[5])**.** 子图查询的问题为: 对于给定的一个查询图，返回图数据库中所有查询图的超图。给定一个图数据库 $GD$ 和一个查询图 $q$，子图查询的目的是找出在 $GD$ 中所有包含 $q$ 或者 $q$ 的同构的图的集合，最后返回该超图集合 $Q$。$Q = g | g \in GD \wedge q \subseteq g$。

**定义 2.9** (超图查询[2])**.** 超图查询的问题为: 对于给定的一个查询图，返回图数据库中所有查询图的子图。给定一个图数据库 $GD$ 和一个查询图 $q$，超图查询的目的是找出在 $GD$ 中所有以 $q$ 为超图的图的集合，最后返回该子图集合 $Q$。$Q = g | g \in GD \wedge q \supseteq g$。

图2.4就是同一个查询 $q$ 在同一数据库中子图查询和超图查询的不同结果。

**定义 2.10** (精确查询)**.** 图精确查询的问题是这样定义的: 对于给定数据库 $G = g_1, g_2, ..., g_n$, 查询图 $q$, 返回 $G$ 中与 $q$ 具有子图同构的图集合。具体可以分为子图查询和超图查询。

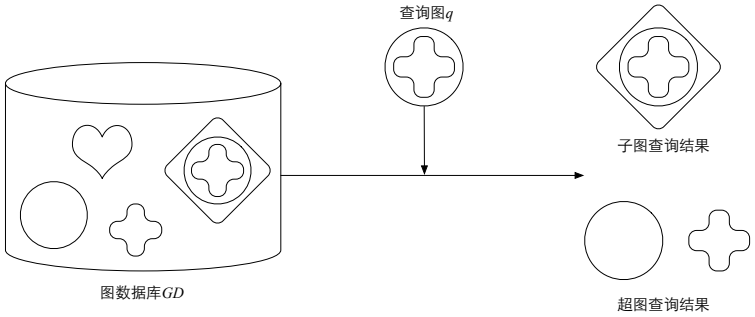



图 2.4 子图查询与超图查询

**定义 2.11** (近似查询)**.** 图近似查询，又称相似性搜索，是这样定义的：对于给定数据库 $G = g_1, g_2, ..., g_n$, 查询图 $q$, 返回 $G$ 中与 $q$ 距离小于预设阀值的图集合。

因此，近似查询中相似性度量方法和近似阀值决定了结果集的大小。阀值越大，候选集越多；阀值越小，候选集越少。当阀值为零是，其结果应该与精确查询一致。

常见的相似性度量方法有两种，一种方法是*编辑距离 (edit distance)*. *编辑距离*就是我们将图 $G$ 通过一系列操作 (如增删点边，重新标号等) 变换为另一个图 $G'$ 所需的操作数。我们可以通过给不同操作分配不同的权值，然后计算权值和作为距离，这样可以让距离更为符合我们实际需求。虽然编辑距离是一种非常直观的图相似性测度方法，但是我们很难计算 (实际上，这是个 NP-hard 问题)。还有一种方法是*最大公共子图 (maximal common subgraph)*[8]，这里就不详细说明了。

# 第三章　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。

## 3.1　图精确查询算法

本节将详细介绍子图查询中的 *GraphGrep* 算法[1] 和 *gIndex* 算法[9]。

### 3.1.1　GraphGrep 算法

2002 年 ShaSha 教授等人提出了 *GraphGrep* 算法[1] 。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3.1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 3.1 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。

### 3.1.2 gIndex 算法

2004 年，Yan 教授等人提出了 *gIndex* 算法[9]。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.

## 3.2　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[6] 和 *C-tree* 算法[3]。

### 3.2.1　G-Hash 算法

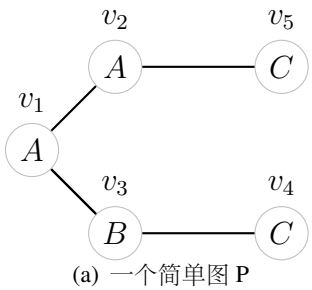
*G-Hash* 算法是 Wang 教授等人于 2009 年提出的一种图相似性搜索方法。相比较其他近似搜索方法，这种方法更加稳定高效。G-Hash 开创性地采用了*简化包表示 (Reduced Bag Represent)* 来表示每个节点特征，并表示成字符串，Hash 存储作为索引。并利用*小波匹配核函数 (Wavelet Graph matching kernels)* 来计算节点间的相似度，从而得到和查询图最为相似的 K 个图。但是 G-Hash 算法的准确度和速度完全取决于相似度度量函数，因此一个好的相似性度量方法尤为重要。

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3.1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 3.1.** 图3.1就是一个 G-Hash 构建索引的实例。图3.1(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3.1(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3.1(c) 所示。而整个数据库生成的哈希表就如图3.1(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。




(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 3.1 一幅简单示例图

图是 G—Hash 查询的一个完整例子。从 XXXX

添加 G-Hash+ 中那副图

此方法还支持动态增删图数据，如例3.1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3.1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。

### 3.2.2 Closure tree 算法

C-tree

# 第四章　基于二次哈希开链法的图精确查询

由于传统算法在利用哈希表存储索引中多采用简单哈希，容易产生冲突问题，导致构建索引效率较低。本章在路径索引的基础上，探究了不同哈希方法对算法速度的影响，并提出一种基于二次哈希开链法的精确查询算法，来减少图查询中的过滤阶段的耗时，以提高查询速度。本章将先介绍下现有的哈希算法，然后详细介绍基于路径的查询方法包括作为此算法验证方法用的子图同构算法 *ULLMANN*[4]。最后是实验结果与分析。

## 4.1　常用哈希方法

本节将介绍几种常用的哈希方法及字符串哈希函数。

补充

### 4.1.1　开放定址法

### 4.1.2　开链法

### 4.1.3　再哈希法

### 4.1.4　常用字符串哈希函数

## 4.2　基于路径的查询算法

本方法同 *GraphGrep* 算法[1]，都是基于路径的精确子图查询算法。算法基本流程如下：(1) 遍历图数据库中图的路径，(2) 利用双哈希构建索引，(3) 遍历查询路径，(4) 利用基本索引特征做先验剪枝，(5) 路径合成进一步筛选候选集，(6) 子图同构确定最终结果。下面我们将分小节详细说明这些步骤。

### 4.2.1　数据库路径遍历

首先，对于每个数据库我们设定一个路径长度的上限 $l_p$，$l_p$ 越大意味着可记录的路径越长，索引集合也会相应增加。随后对于数据库中的每幅图，我们用深度优先搜索遍历每个节点，遍历最大深度为 $l_p$，并记录遍历过程中经过的每一条路径，存成一个列表，用于下一步构建索引。

### 4.2.2 二次哈希开链法索引构建

双哈希是再哈希的一种，其利用两个哈希函数来构造哈希探测序列，大大降低了地址冲突概率。但是传统上，双哈希方法实质上也是开放定址法的一种，也就是如果所需节点数目大于 $Mod$ 时，将完全无法表示。这完全不符合图数据库实际情况，因为作为一个通用算法，我们无法预先确定数据库中最大图的节点数。因此，本文提出了一个变形的双哈希算法，即**二次哈希开链法**来进行哈希定址。

二次哈希开链法就是只进行一次双哈希，然后再有冲突就利用开链法解决。双哈希函数公式如4-1。

$$h(key) = (h_1(key) + h_2(key))\%Mod \tag{4-1}$$

其中 $h_1, h_2$ 为两个不同的哈希函数，$Mod$ 为哈希函数取模值，一般为一个小于但最接近存储空间大小的素数。当 $h_1(key)$ 发生冲突时，再用 $h_2(key)$ 的值作为偏移量来进行探测。如果再有冲突则进行开链法，存成链表。

根据二次哈希开链法的特点，本文设计了一种将路径字符串映射到哈希表的算法，如算法4.1所示。而访问时函数则不需重新设计，直接用开链法原有函数即可。

**算法 4.1** 二次哈希编码

**输入：** 路径字符串 $path\_string$

**输出：** 哈希编码 $code$

```
1: function HASH(path_string)
2:     code ← h1(key)%Mod
3:     if code 有冲突 then
4:         code ← (code + h2(key))%Mod
5:     end if
6:     return code
7: end function
```

在二次哈希开链法中哈希函数可以自选，不过我们推荐选取算法4.2中的两个函数作为 $h_1, h_2$，经过实验这两个函数对于字符串哈希这两个效果最好。

*BKDRHash* 运算简单，速度快，所以作为第一次哈希函数。*APHash* 不易冲突，所以作为第二次。

**算法 4.2** 哈希函数

**输入:** 字符串 $String$

**输出:** 哈希编码 $code$

**function** $h_1(String)$ ▷ BKDRHash
    $char \leftarrow (string.first)$
    $code \leftarrow 0$
    **while** $char \neq 0$ **do**
        $code \leftarrow code << 6 + char$
        $char \leftarrow (char.next)$
    **end while**
    **return** $(code \& 0 \times 7FFFFFFF)\%Mod$
**end function**
**function** $h_2(String)$ ▷ APHash
    $char \leftarrow (string.first)$
    $code \leftarrow 0$
    $i \leftarrow 0$
    **while** $char \neq 0$ **do**
        **if** $i$ 为偶数 **then**
            $code \leftarrow (code \oplus ((code << 7) \oplus char \oplus (code >> 3))$
        **else**
            $code \leftarrow (code \oplus (\sim ((code << 11) \oplus char \oplus (code >> 5)))$
        **end if**
        $char \leftarrow (char.next)$
    **end while**
    **return** $(code \& 0 \times 7FFFFFFF)\%Mod$
**end function**

通过哈希存储，我们可以很便捷得获得各图包含的路径关系表，我们将其存到文件中作为索引，分离查询与建库，进行离线查询加速查询速度。

### 4.2.3 查询图路径遍历

我们对查询图也像数据库中的图一样进行拆分，以 $l_p$ 为路径最大长度遍历出所有路径。然后同样存成一个哈希表，记录着每一条路径出现了几次。为进一步查询做准备。不过和数据库路径遍历有所不同的是，查询图的遍历过程中需要记录不同路径中相同的节点，这个可以通过在路径中添加特定标签实现。

### 4.2.4 先验剪枝

在介绍本算法的先验剪枝步骤之前，我们先介绍一下包含逻辑规则 *(inclusion logic)*。

**定义 4.1** (包含逻辑[7])**.** 对于给定的标号图 $g_1, g_2$，和 $g_1$ 的一个子图 $g'$，若 $g_1$ 是 $g_2$ 的子图，则 $g'$ 必定是 $g_2$ 的子图 $(g_1 \subseteq g_2) \Rightarrow (g' \subseteq g_2)$。反之，若 $g'$ 不是 $g_2$ 的子图，则 $g_1$ 也不可能是 $g_2$ 的子图 $(g' \not\subset g_2) \Rightarrow (g_1 \not\subset g_2)$

从包含逻辑规则中，我们可以得知如果查询图 $g_1$ 中的子图 $g'$ 不是数据图 $g_2$ 的子图，那么 $g_2$ 就不可能是 $g_1$ 的超图，因此可以放心得把 $g_2$ 从候选集中删去。这就大大加速了“过滤 -验证”框架中过滤阶段的过滤速度。

在本算法中，我们从三个方面进行了先验剪枝来缩小索引集个数。三个方面分别是 (i) 节点，(ii) 边，(iii) 路径。如果查询图中的某个节点个数大于数据图中的，那么数据图自然不会包含查询图。如果查询图中有数据图中没有的边，那么自然这幅数据图也不合要求。如果查询图中某条路径的个数大于数据图，那么因为包含逻辑规则，这幅数据图也当从候选集中删去。

经过这三步筛选过程，候选集将会大大减少，降低了后文所述路径合成和子图同构所需时间。

### 4.2.5 路径合成

当完成先验剪枝后，候选集已经相对较小，但是并没有达到最好的情况。我们可以通过路径合成确定其中很多的合理性。路径合成的复杂度远远小于子图同

构，因此最后对候选集做一次路径合成可以大大降低最终复杂度。路径合成，顾名思义就是将多条路径进行合成，具体而言就是将多个有着公共节点的路径合成到一起，通过判定其是不是相同节点来筛选候选集。我们采用的是遍历的方法，只进行两两合成，然后逐一比对。如例4.1所示。

**例 4.1.** 假设有路径 $\bar{A}B\underline{C}\bar{A}$ 和 $\underline{C}B$，其中有相同标记的节点均为同一节点，即在本例中两个 $A$ 是同一节点和两个 $C$ 也是同一节点。

1. 现在假设数据库中有四幅图，其路径信息如下所示，数字代表节点标识:

$$g_1 : ABCA = (1,0,3,1)\ CB = (3,2)$$

$$g_2 : ABCA = (1,2,3,1)\ CB = (3,2)$$

$$g_3 : ABCA = (1,0,3,4)\ CB = (3,2)$$

$$g_4 : ABCA = (1,0,4,1)\ CB = (3,2)$$

2. 在路径合成前，我们就可以利用节点信息删去 $g_3$，因为 $g_3$ 的 $ABCA$ 中两个 $A$ 一个是 1，一个是 4，并非同一节点。

3. 我们将 ABCA 和 CB 合成, 我们知道其中两个 $A$ 是统一节点，两个 $C$ 是同一节点，而两个 $B$ 不是。我们得到合成结果:

$$g_1 : ABCACB = (1,0,3,1),(3,2)$$

$$g_2 : ABCACB = (1,2,3,1),(3,2)$$

$$g_4 : ABCACB = (1,0,4,1),(3,2)$$

4. 显然，$g_2$ 不满足条件，因为它的两个 $B$ 也是一个节点，$g_4$ 也不满足，因为其两个 $C$ 不是同一节点。所以筛选集中只剩下 $g_1$。

可见，通过路径合成，我们大大缩小了候选集，降低了下一步子图同构所需的计算量，加速了整个算法。

### 4.2.6　子图同构

子图同构作为算法最后的验证部分，承担着对于整个算法正确性把关的责任，也同样是个需要消耗大量运算量的部分。由于子图同构是个 $NP-hard$ 问题，所

图 4.1 Ullmann 算法流程图

以目前仍没有一种快速的解法。我们选用经典算法 *ULLMANN* 算法[4] 来解决这个问题。下面我们将大致介绍下 UllMANN 算法。

ULLMANN 算法是 Ullmann 教授 1976 年提出的一种经典图同构算法，其本质是基于一个深度优先搜索树。其算法流程如图4.1所示。首先，根据查询图节点的出入度从数据图中找出候选集。随后，再根据每个节点的邻接节点对候选集进行筛选。最后，通过深度优先查询，一一遍历配对，寻找匹配点。例4.2就是一个子图同构的完整过程。

Ullmann
子

**例 4.2.**

## 4.3 实验结果与分析

### 4.3.1 实验环境

本文提出算法的实验环境为 CPU Intel Core i7, 主频为 1.7 GHz，内存为 8 GB 1600 MHz DDR3，硬盘为 128GB SSD，操作系统为 Mac OS X Yosemite 10.10.3；所有算法均用 C 语言在 clang 600.0.56 环境编译完成。

### 4.3.2 实验数据分析

实验数据全采用真实数据，为 DTP 提供的 AIDS 数据集。可从以下网址得到:`https://wiki.nci.nih.gov/display/NCIDTPdata/AIDS+Antiviral+Screen+Data`。我们从其数据库中随机抽取了 1000,2000,4000,16000 个图作为我们的查询集合。

验

# 第五章　基于节点相似度的图相似性搜索

由前文可知，图数据能表示复杂的数据结构，在诸多领域也得到了广泛应用。从基本的生物，化学的分子结构，到交通网络，人际关系网都可以用图来建模。而对于图数据库的索引也自然成为了热点问题。

上一章我们介绍了精确子图搜索方法，但是由于真实情况下图数据库具有信息不完整，含有杂质等情形，精确搜索容易出现各种不匹配问题，难以得到我们想要的结果。同时，对于查询图的完整信息有时查询者也并不了解。所以相似性搜索的实际应用领域更加广泛。对于实际应用，相似性搜索的研究意义远超过了精确搜索。

传统的图相似性算法虽然运行效率已较为理想，但是编码上过于复杂，也无法做到对所有图数据库良好适配。所以本章我们提出了一种性能上不输于传统算法，但是实现更为简单，并且无需复杂设计即可适用于大量图数据库的基于节点相似度的通用型算法。本章将首先详细介绍下一些相关概念，然后介绍下我们算法的具体思路并给出详细的代码设计方案，最后给出对于真实数据的实验结果与分析。

## 5.1　相关概念

本节主要介绍下下文介绍算法时会用到的一些概念，包括小波图匹配核函数，G-Hash 中的图相似度定义，节点相似度定义三部分。

### 5.1.1　小波图匹配核函数

**小波图匹配核函数 *(Wavelet Graph matching kernel)***[6] 是 Wang 等人在 G-Hash 算法中提出的，用于度量图相似度的一个核心函数。其思想是先通过压缩每个节点周围邻接节点的属性信息，然后应用非递归线性核去计算图之间的相似度。这个方法包含两个重要的概念：***h-hop* 邻域**和**离散小波变换**。用 $N_h(v)$ 标记一个节点 $v$ 的 ***h* 跳邻域**，代表一个距离节点 $v$ 最短距离是 $h$ 跳 (跳过 $h$ 个节点, 也就是 $h+1$ 的曼哈顿距离) 的节点集合。**离散小波变换**涉及到一个小波函数的定义，见下文公

式5-1 , 也用到了 $h$ 跳邻域。

$$\psi_{j,k} = \frac{1}{h+1} \int_{j/(k+1)}^{(j+1)/(k+1)} \varphi(x)dx \tag{5-1}$$

$\varphi(x)$ 代表 ***Haar*** 或者 ***Mexican Hat*** 小波函数，$h$ 是在将 $\varphi(x)$ 在 $[0,1)$ 区间上分成 $h+1$ 个间隔后的第 $h$ 个间隔，$j \in [0,h]$。基于以上两个定义，我们现在可以将小波分析用在图上了。我们用小波函数来计算每个节点的局部拓扑和。公式(5-2)展示了一个小波度量方法，记做 $\Gamma_h(v)$，以图 $G$ 中的一个节点 $v$ 为例。

$$\Gamma_h(v) = C_{h,v} \times \sum_{j=0}^{k} \psi_{j,k} \times \bar{f}_j(v) \tag{5-2}$$

其中,$C_{h,v}$ 是一个归一化因子

$$C_{h,v} = (\sum_{j=0}^{h} \frac{\psi_{j,h}^2}{|N_h(v)|})^{-1/2}, \tag{5-3}$$

$\bar{f}_j(v)$ 是离节点 $v$ 最远距离为 $j$ 的原子特征向量的平均值

$$\bar{f}_j(v) = \frac{1}{|N_j(v)|} \sum_{u \in N_j(v)} f_u \tag{5-4}$$

$f_u$ 表示节点 $v$ 的特征向量值。这样的特征向量值只会是下面四种中的一种：定类，定序，定距，定比。对于定比和定距特征值，直接在上述的小波分析时代入其值即可得到局部特征值。对于定类和定序节点特征，我们首先建立一个直方图，然后用小波分析提取出特征值。在节点 $v$ 分析完成后，我们可以得到一个节点列表 $\Gamma^h(v) = \{\Gamma_1(v), \Gamma_2(v), ..., \Gamma_h(v)\}$, 我们称其为小波测度矩阵。用此方法我们可以将一个图转换为一个节点向量集合。因为小波变换有明确的正负区域，所以这些小波压缩特征可以表示出局部的邻接节点和距离较远的邻接节点的差异。因此，通过小波变换，一幅图的结构化信息可以压缩成节点特征。从而我们可以忽略拓扑结构来专心于节点匹配。核函数就是建立在这些集合上的，我们以图 $G$ 和 $G'$ 为例，图匹配核函数是这样的

$$k_m(G, G') = \sum_{(u,v) \in V(G) \times V(G')} K(\Gamma^h(u), \Gamma^h(v)), \tag{5-5}$$

$$K(X, Y) = e^{\frac{-\|X-Y\|_2^2}{2}}. \tag{5-6}$$

**WA** 方法是一种很好的利用核函数进行图相似度定义的方法。但是这个方法有一个问题，就是小波匹配核的总时间复杂度是 $O(m^2)$，核矩阵的是 $O(n^2 \times m^2)$，$n$

是数据库图个数，$m$ 是平均节点个数。这意味着，当数据库尺寸增加时，计算时间将大幅度增加。

### 5.1.2　G-Hash 中的图相似性定义

图相似性有很多种定义方式有很多，第二章中介绍的图编辑距离和最大公共子图就是两种常用的图相似度度量方法。

G-Hash 中是利用核函数计算两图相似性的。公式5-7就是两图相似度距离的定义。

$$\begin{aligned} d(G,G') &= \sqrt{\|\phi(G)-\phi(G')\|_2^2} \\ &= \sqrt{\langle\phi(G)-\phi(G'),\phi(G)-\phi(G')\rangle} \\ &= \sqrt{\langle\phi(G),\phi(G)\rangle+\langle\phi(G'),\phi(G')\rangle-2\langle\phi(G),\phi(G')\rangle} \\ &= \sqrt{k_m(G,G)+k_m(G',G')-2k_m(G,G')} \end{aligned} \tag{5-7}$$

公式中 $k_m(G,G)$ 代表图 $G$ 和其本身的核函数值，$k_m(G',G')$ 是图 $G'$ 及其本身的值，$k_m(G,G')$ 就是图 $G$ 和 $G'$ 的。

$$k_m(G,G') = \sum_{v\in G', u\in simi(v)} K(\Gamma^h(u),\Gamma^h(v)) \tag{5-8}$$

$simi(v)$ 是一个包含着图 $G$ 和节点 $v$ 哈希到同一个位置的节点集合。我们用以下的解码方式来获取包含这些节点的图号和节点号。

显然，仅利用相似点对而非所有点对来计算两图相似度可以节约很多运算时间。因此为了增加准确度，相似的节点应该被哈希到相邻的位置。在图很大 (如大于 40) 时，我们也要计算相邻位置的节点。

因此在核函数计算时我们只考虑相似点对，在使用 RBF 核的情况下，$K(\Gamma^h(u),\Gamma^h(v))\approx 1$，所以公式 (2) 可以写成

$$K(G,G') \approx \sum_{v\in G', u\in simi(v)} 1 = \sum_{v\in G'} |simi(v)| \tag{5-9}$$

$|simi(v)|$ 是在 $simi(v)$ 中的节点数目。这意味着我们只需要数据图 $G$ 中和查询 $G'$ 相似的点个数，其和就是我们要求的核。同理，我们可以这样计算每个图与其自己的核。显然，每个图与自己的核就是节点数目。

所以公式5-7最终变成公式5-10, 其中 $|V_G|$ 代表图 $G$ 的节点数。

$$d(G,G') = \sqrt{|V_G| + |V_{G'}| - 2\sum_{v \in G'} |simi(v)|} \tag{5-10}$$

### 5.1.3 节点相似度

在上一节我们介绍了 G-Hash 中的图相似性的定义。其中有参数 $simi(v)$ 代表与节点 $v$ 相似的节点，但是何为相似，这个很难定义。G-Hash 算法在这边根据不同图数据库构建了不同类型的哈希，来保证相似的节点都在相邻位置。而这在实际应用中很是困难。因此我们提出了**节点相似度**这一概念。我们用查询图每个节点和数据图中每个节点的节点相似度作为 $simi(v)$ 的值。

**定义 5.1** (节点相似度)**.** 用**简化包** ***(Reduced Bag)*** 表示的两个节点字符串之间的相似度节点相似度称为**节点相似度**。

因为简化包表示的字符串实质上是一个向量，如 $'a,1,0,1'$ 就可以看做一个向量 $\vec{A} = (a,1,0,1)$, 所以对于字符串的距离，也就是节点相似度，可以用公式(5-11)计算。

$$simi(A,B) = \frac{\vec{A} \times \vec{B}}{||\vec{A}| \times |\vec{B}||} \leq 1 \tag{5-11}$$

其中，$|\vec{A}|$ 表示 $\vec{A}$ 的模，$simi(A,B)$ 永远是小于等于 1 的，值越大代表越为相似，为 1 代表两个节点特征完全一样。

## 5.2 基于节点相似度的图相似性搜索

本算法和 G-Hash[6] 基本类似，只是在查询过程计算相似度时没有利用哈希特征找到相似的节点，而是利用节点相似度来计算相似度。这样避免了由于哈希函数选取不当造成的相似节点不在靠近位置的问题。用普通的枚举虽然理论复杂度从 $O(1)$ 变为了 $O(n)$ 但是提高了查询准确度，并且由于哈希存在冲突，而实际图节点一般最多为千个等情况，我们算法的运行效率并不比 G-Hash 低，甚至在某些情况下会略好于 G-Hash 算法，而且编码难度也大大降低。本节将从数据库构建，查询过程，数据库维护，编码设计四个方面详细说明我们提出的基于节点相似度的图相似性搜索算法。

图 5.1 数据库示例

### 5.2.1　数据库构建

首先统计数据库中的不同标号数目，然后将数据库中的每幅图都用简化包表示，选取节点标号和与其直接连接的各标号个数作为特征值。归一化后变成字符串，利用红黑树建立 Bag 编号与字符串的对应关系。之所以选用红黑树而不是哈希表是因为对于纯字符串结构红黑树的表现要好于哈希表。然后再以 Bag 编号为图，图序号为列建立一个二维表，每个值代表在此图中此种节点有几个。最终数据库如图5.1所示。

补充一张
树的图，
数据库 ta

### 5.2.2　查询前 K 个相似图

首先，我们参考 G-Hash 的图距离函数定义了我们的基于节点相似度的图距离函数，即公式(5-12)。

修改公式
改为 eqre

$$d(G,G') = \sqrt{|V_G| + |V_{G'}| - 2\sum_{v\in G', u\in G} simi(v,u)} \tag{5-12}$$

当我们得到一个查询图 $G'$ 时，我们也用简化包将其表示，得到节点字符串集。然后利用公式(5-12)计算查询图和数据库中每副图的距离。随后将结果排序，取出最小的 K 个即为与查询图最相似的 K 个图。

### 5.2.3　数据库维护

当需要增删图的时候，会出现两种情况:(i) 增删图对标号个数没有影响，即没有因为添加图而添加新的标号，也没有标号因为删除图而不被使用，(ii) 增删图对标号个数有影响，即因为添加图而引入了新的标号，或者因为删除图而有的标号不再有图使用。当出现 (i) 情况时，只需要添加一列或者删除一列即可。当出现 (ii) 情况时，则需要重新建立数据库索引，不过无需每次出现都重新建立，每 5-10 次重建一次即可。

### 5.2.4　代码设计

本算法的实现非常简单，不需要任何复杂数据结构，只需要三个类即可实现。类图设计如图5.2所示。

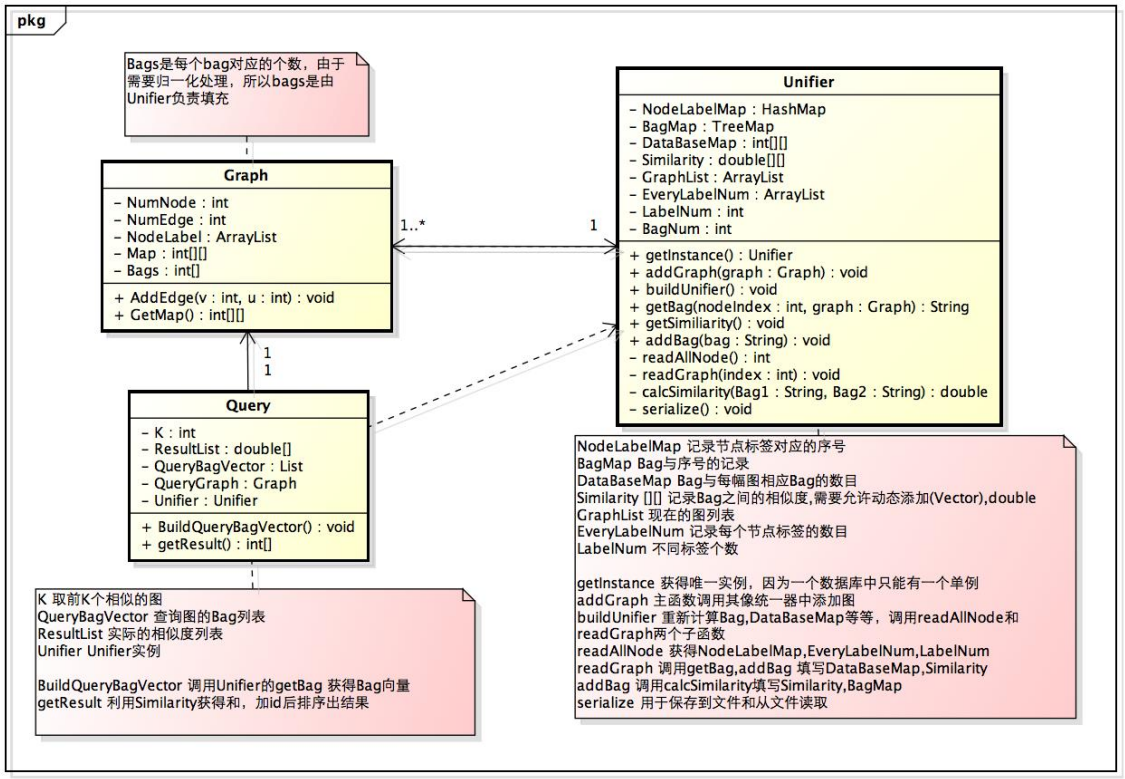



图 5.2 基于节点相似度的图相似性搜索设计类图

其中，$Graph$ 为一个最小单元即一个图，对外提供一些基本数据访问方法。$Unifier$ 顾名思义，是个用于归一化的类，也是数据库类，负责建库，增删图，和数据查询时一些算法定义，像获取简化包表示，图相似度计算就是靠这个类实现的。$Query$ 是一个查询，通过调用 $Unifier$ 实现查询过程，最后输出查询结果。结果表现为图的 id 列表。我们还给 $Unifier$ 添加了序列化接口，实现建库查询分离，加速查询过程。

由上可见，基于节点相似度的图相似算法实现非常方便，可运用在大量环境中。

## 5.3 实验结果与分析

### 5.3.1 实验环境

本文提出算法的实验环境为 CPU Intel Core i7, 主频为 1.7 GHz，内存为 8 GB 1600 MHz DDR3，硬盘为 128GB SSD，操作系统为 Mac OS X Yosemite 10.10.3；所有算法均用 Java 语言在 JDK1.7u71 环境编译完成。

### 5.3.2　实验数据分析

实验数据全采用真实数据，为 DTP 提供的 AIDS 数据集。可从以下网址得到:https://wiki.nci.nih.gov/display/NCIDTPdata/AIDS+Antiviral+Screen+Data。我们从其数据库中随机抽取了 1000,2000,4000,16000 个图作为我们的查询集合。

# 第六章　总结与展望

本章作为全文的最后一章，将对本文的所述内容进行总结，并对下一步的研究工作给出一些意见。

## 6.1　本文总结

主要工作如下:

根据绪论

1. 本文首先介绍了下图的基本定义，存储方法，图查询类型和一些经典的查询算法。

2. 随后，我们对于图精确搜索提出了一种基于二次哈希开链法的搜索算法，有效避免了传统哈希方法冲突频发的问题，加速了整个查询过程。本文中，我们完整得介绍了这种算法，从数据库建库，到查询剪枝，直最后的子图同构检测，并通过实验证明了此算法确实可以加快整个查询过程。

3. 然后，对于图相似性搜索我们提出了一种基于节点相似度的搜索算法。本算法与 G-Hash 算法大致相同，但重新定义了其核心部分——图相似度度量方法，使得编码复杂度大大降低，但同时效率又不低于 G-Hash 算法。本文中我们除了详细介绍了此算法原理，还给出了算法设计类图，并通过实验证明了此算法完全符合我们预期的目标——运行效率不低于 G-Hash，甚至在某些情况下略好于 G-Hash, 但是编码难度大大下降。

综上所述，本文对子图搜索方面做了一定基础性的研究，在认真研究了前人的经典算法基础上，进行了一些新的探索。

## 6.2　进一步的工作

但是，图数据查询仍是图数据管理中的一个重要领域，其中仍有一些问题需要继续去研究与改进。在未来的研究过程中，可以从以下几个方面入手:

1. 目前，时时刻刻产生着大量的图数据信息，如何在图数据库中有效存储，在内存中以何种结构存储图数据，如何用固定的磁盘页面存储不同规模尺寸的图数据，又如何对图数据进行压缩表示，这些基本的物理存储问题将直接决定 I/O 操作的用时。而在实际中对图的存取又是非常频繁的，所以如果能解决存储问题，那么必然能提高整体查询效率。

2. 除了物理存储，逻辑索引显得更为重要，如何高效索引，如何进行维护和更新，这些都是亟待解决的问题。特别是图索引的维护与更新，由于目前大量算法都处于理论阶段而并非实际运用，所以都没有涉及动态维护与更新。但是随着图数据越来越为重要，实际中对图的应用也日益增多。如果能解决这个问题，那么对图从实验室走到实际运用中将会是个强助力。

3. 图查询的计算复杂度通常能达到指数级及以上，单线程的运算速度就成了图查询速度的瓶颈。因此考虑利用 CPU/GPU 异构并行或者多处理器并行，多机并行等也是一个不错的研究课题。

# 参考文献

[1] Rosalba Guigno and Dennis Shasha. Graphgrep : A fast and universal method for querying graphs. 2002.

[2] Shang H, Zhu K, and Lin X. Similarity search on supergraph containment. *ICDE*, 2010.

[3] H. He and A. K. Singh. Closure-tree: an index structure for graph queries. *Proc. International Conference on Data Engineering'06 (ICDE)*, 2006.

[4] J.R.ULLMANN. An algorithm for subgraph isomorphism. *Journal Association for Commputing Machinery*, 23(1):31–42, January 1976.

[5] Kurmochi M and Karypis G. Frequent subgraph discovery. *ICDM*, 2001.

[6] Xiaohong Wang, Aaron Smalter, and Jun Huan. G-hash:towards fast kernel-based similarity search in large graph databases. *ACM*, 2009.

[7] Yan X, Yu P S, and Han J. Graph-based substructure pattern mining. *ICDE 2002*, 2002.

[8] T. Jiang Y. Cao and T. Girke. A maximum common substructure-based algorithm for searching and predicting drug-like compounds. *Bioinformatics, 24(13)*, 2008.

[9] Xifeng Yan, Philip S.Yu, and Jiawei Han. Graph indexing : A frequent structure-based approach. *ACM*, 2004.

[10] 王桂平，王衍，任嘉辰. *图论算法理论, 实现及应用*. 北京大学出版社, 2011.

[11] (英) 维克托 · 迈尔 -舍恩伯格，肯尼思 · 库克耶. *大数据时代: 生活、工作与思维的大变革*. 浙江人民出版社, 2012.

[12] 谭伟，杨书新. 图数据精确查询与近似查询的研究. 2013.