# 1 Hash 函数实现快速图相似性搜索

根据将图中的每个节点利用函数映射到一个特征空间上的这一思想，我们可以设计出一种核函数来进行快速相似性搜索。具体的说，我们有了以下两条发现。

- 当图 $G$ 中节点 $u$ 向量和图 $G'$ 中的节点 $v$ 完全不同，节点 $u$ 和 $v$ 的核值很小基本不影响图核值。所以我们如果仅计算相似点对的核值，得到的核矩阵将和 WA 方法中的相似性度量类似，但耗时很少。
- 如果基于节点向量来构造哈希表的话，相似的对象的位置上将也很靠近。所以我们可以很快速得利用哈希表来寻找相似点对。另外，如果数据库中所有图都存入哈希表中的话，一个位置可能对应不同图中的多个相似节点。因为这些节点都是相似的，所以只要利用其中两个节点进行一次 RBF 核运算就可以代表所有的结果了。节点覆盖给了我们另一个省时间的契机。

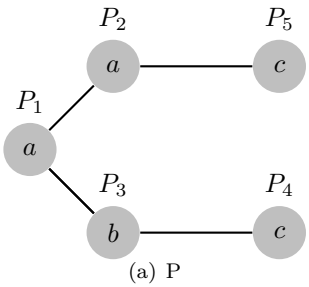
基于以上两条发现，我们引入我们的方法:$k$-NNs 查询 (G-hash).G-Hash 是一种利用 WA 方法进行精确相似度度量，并利用哈希来降低时间复杂度。具体算法见下文。

## 1.1 索引构建

像 WA 方法一样，我们首先需要利用小波变换将数据库中的所有图拆解成点向量的形式。因为 WA 方法对距离参数的选取并不敏感，用不同的小波算法造成的差异也并不大 [1] ，所以我们设定 $h$ 为 2，并才用 *Haar* 小波算法来简化算法

当图完全拆解成节点向量后，我们就可以利用这些向量构建哈希表。这时，我们需要一个哈希函数来把相似的节点哈希到相同的位置。这就意味着我们要基于节点向量构建哈希键。我们可以用和构造节点向量同样的思想构建哈希键，即利用节点标号和邻接信息来构造。我们将每一个节点向量中的特征信息离散成一个整数，并将每个节点标签直接编码为一个 n 位的字符串（除去一个代表本身标号的 1，其他全 0）。其他特征就近似到最近的整数上。当节点向量变成了一个整数列表后，我们再把节点向量变为一个字符串 (用二进制表示每个整数，并用下划线连接)。这样的一个字符串就是对于节点的哈希键。我们以图表 1 的图 P 来举例说明。

例子 4.1. 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。所以我们一共有四个特征。这个图的特征仅属于取值中的一种:*nominal*。我们首先写出节点特征矩阵如图表 *1* 中的 *(B)*。然后用小波函数提取出实际特征值。举例而言，对于 $P_3$ 在用 $h = 0$ 的小波分析提取后，局部特征是 $[b, 2, 0, 1\}$。然后生成的哈希表如图表 *1(C)* 所示。




(a) P

| Nodes | Label | #a | #b | #c |
|---|---|---|---|---|
| $P_1$ | a | 1 | 1 | 0 |
| $P_2$ | a | 1 | 1 | 1 |
| $P_3$ | b | 2 | 0 | 1 |
| $P_4$ | c | 0 | 1 | 0 |
| $P_5$ | c | 1 | 0 | 0 |

(b) 节点特征矩阵

| 哈希键 | 哈希值 |
|---|---|
| a,1,1,0 | $P_1$ |
| a,1,1,1 | $P_2$ |
| b,2,0,1 | $P_3$ |
| c,0,1,0 | $P_4$ |
| a,1,1,1 | $P_5$ |

(c) 图 P 的哈希表

图表 1: 一幅简单图

注意: 在这个算法框架中，我们在构建哈希索引时没有对查询图做任何假设。

## 1.2 *K-NNs* 查询过程

为了获得对于给定图的 *K-NNs*, 我们需要计算它和数据库中其他图的距离。以下是我们定义的利用核函数进行两图距离度量的函数

$$\begin{aligned} d(G, G') &= \sqrt{\|\phi(G) - \phi(G')\|_2^2} \\ &= \sqrt{\langle \phi(G) - \phi(G'), \phi(G) - \phi(G') \rangle} \\ &= \sqrt{\langle \phi(G), \phi(G) \rangle + \langle \phi(G'), \phi(G') \rangle - 2\langle \phi(G), \phi(G') \rangle} \\ &= \sqrt{k_m(G, G) + k_m(G', G') - 2k_m(G, G')} \end{aligned} \tag{1}$$

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。在后文中，我们会介绍该如何计算这些值。

尽管在将查询图节点哈希到哈希表时，我们可以得到核函数

$$k_m(G,G') = \sum_{v\in G', u\in simi(v)} K(\Gamma^h(u),\Gamma^h(v)) \tag{2}$$

$simi(v)$ 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

$$K(G,G') \approx \sum_{v\in G', u\in simi(v)} 1 = \sum_{v\in G'} |simi(v)| \tag{3}$$

$|simi(v)|$ 是在 $simi(v)$ 中的节点数目。这意味着我们只需要统计图 $G$ 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。

在完成上述操作后，我们会得到一个距离向量，其每一个值都对应这一个数据库中的图与查询图的距离，通过对这个距离向量排序，我们就可以得到对于给定的查询的 $K-NNs$。

### 1.3 动态插入与删除

如果要向数据库中插入一副图，我们只需将新图的节点进行哈希，然后加入节点哈希表。这样插入完成后，只有图中所有的节点的位置会发生变化。而且，因为一幅图的节点是有限的，所以插入操作的时间和构建索引的内存也是有限的。删除操作和插入类似，我们只需找到图中节点在哈希表中的对应哈希值，然后删掉即可。

## 参考文献

[1] A. Smalter, J. Huan, and G. Lushington. Graph wavelet alignment kernels for drug virtual screening. *Proceedings of the 7th Annual International Conference on Computational Systems Bioinformatics*, 2008.